New circuit switching techniques in on-chip networks

SHAOTENG LIU

Doctoral Thesis in Electronic and Computer Systems
Stockholm, Sweden 2015

TRITA-ICT 2015:18
ISSN 1653-6363
ISRN KTH/ICT-15/18-SE
ISBN 978-91-7595-727-2

KTH School of Information
and Communication Technology
SE-164 40 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi torsdagen den 04 Dec 2015 klockan 9.00 i Sal A, Electrum, Kista, Stockholm.

Tryck: Universitetsservice US AB

Abstract

Network on Chip (NoC) is proposed as a promising technology to address the communication challenges in deep sub-micron era. NoC brings network-based communication into the on-chip environment and tackles the problems like long wire complexities, bandwidth scaling and so on. After more than a decade's evolution and development, there are many NoC architectures and solutions available. Nevertheless, NoCs can be classified into two categories: packet switched NoC and circuit switched NoC. In this thesis, targeting circuit switched NoC, we present our innovations and considerations on circuit switched NoCs in three areas, namely, connection setup method, time division multiplexing (TDM) technology and spatial division multiplexing (SDM) technology.

Connection setup technique deeply influences the architecture and performance of a circuit switched NoC, since circuit switched NoC requires to set up connections before launching data transfer. We propose a novel parallel probe based method for dynamic distributed connection setup. This setup method on one hand searches all the possible minimal paths in parallel. On the other hand, it also has a mechanism to reduce resource occupation during the path search process by reclaiming redundant paths. With this setup method, connections are more likely to be established because of the exploration on the path diversity.

TDM based NoC constitutes a sub-category of circuit switched NoC. We propose a double time-wheel technique to facilitate a probe based connection setup in TDM NoCs. With this technique, path search algorithms used in connection setup are no longer limited to deterministic routing algorithms. Moreover, the hardware cost can be reduced, since setup requests and data flows can co-exist in one network. Apart from the double time-wheel technique for connection setup, we also propose a highway technique that can enhance the slot utilization during data transfer. This technique can accelerate the transfer of a data flow while maintaining the throughput guarantee and the packet order.

SDM based NoC constitutes another sub-category of circuit switched NoC. SDM NoC can benefit from high clock frequency and simple synchronization efforts. To better support the dynamic connection setup in SDM NoCs, we design a single cycle allocator for channel allocation inside each router. This allocator can guarantee both strong fairness and maximal matching quality. We also build up a circuit switched NoC, which can support multiple channels and multiple networks, to study different ways of organizing channels and setting up connections. Finally, we make a comparison between circuit switched NoC and packet switched NoC. We show the strengths and weaknesses on each of them by analysis and evaluation.

To mother, father and Jie

Acknowledgment

This is the first chapter of the thesis, but this is the last chapter I write. This is the easiest part for the readers, but this is the hardest one for me — I am indebted. I am indebted to many people who have offered their generous help and genuine concern to me during my PhD study. I would like to try my best to express my sincere gratitude here.

I am especially grateful to my two supervisors: Axel Jantsch and Zhonghai Lu. I would like to thank Axel for giving me the opportunity for being his student and inspiring me with many good ideas and interesting topics. Axel is a very respectable person. He sets a good example to me on how to behave as a researcher. I would like to thank Zhonghai for imparting me the knowledge and skills, for sharing the wisdom of doing research, for motivating me and helping me, for giving the warmth and strength to get through difficulties in both life and research.

I want to specially thank several people who helped my PhD research career. I would like to thank Prof. Elena Dubrova. The talks with her are always very encouraging. She also reviewed this thesis. I would like to thank Graham Schelle for giving me the great opportunity to experience the industrial research in American. I would like to thank my Master supervisor Prof. Jinmei Lai, who inculcated me with the love of research. I would like to thank Prof. Lirong Zheng and Prof. Shili Zhang, who were the first to introduce me about the on-going research topics in KTH and Sweden.

Additionally, I would like to deeply thank many current and former colleagues in my department. I would like to thank Prof. Ahmed Hemani, Ingo Sander, Alina Munteanu and Johnny Öberg for facilitating my PhD study. I would like to thank Ming Liu, Yuan Yao, Jian Wang, Shuo Li, Pei Liu, Yanchen Long, Mohammad Badawi, Xueqian Zhao, Huimin She, Nan Li for the inspiring discussions and laughter we have together. I would also like to thank Jue Shen, Liang Rong, Li Xie, Chuanying Zhai, Junhe Gan, Ning Ma, Kunlong Yang, Geng Yang, Botao Shao, Qiansu Wan, Zhuo Zhou, Jia Mao, Tao Sha and Byron Roberto Navas Viera for all the joys and fun shared with me.

Furthermore, I would also like to extend my gratitude to colleagues and friends in KTH, Xilinx and in SICS. Especially, I would like to thank Prof. Mats Brorson, Patrick Lysaght, Rebecca Stainert, Jan Ekman, Per Kreuger for their help and concern.

Moreover, I would like to give my heartfelt thanks to my friends. Thank Song Lu, Zhiyun Pei, Bo Wei, Jing Sun, Ze Ni, Jitong Sun, Kai Yu, Yueyue Lin, Martin Ericsson and Jing Fu for accompanying me in Sweden for so many years.

Finally, and most importantly, I am indebted to my parents and my wife Jie Ji. Without your consistent and selfless supports, this work will never be finished. Many thanks to my parents. Thanks for your permanent love and good-to-excellent care and concern. Many thanks to Jie. Thanks for being with me all along the way. Thanks for making every day of mine special.

Shaoteng Liu
October 2015, Stockholm Sweden

Contents

List of Figures

List of Abbreviations

ATM	Asynchronous Transfer Mode
AXI	Advanced Extensible Interface
FIFO	First-In-First-Out
HRA	Homogeneous Resource Allocation
HWC	Highway Channel
IP	Intellectual Property
MRR	Massive Round-Robin
MultiCS	Multi-channel and Multi-network Circuit Switched
NI	Network Interface
NoC	Network on Chip
QoS	Quality of Service
SDM	Spatial Division Multiplexing
TDM	Time Division Multiplexing
VC	Virtual Channel

List of Publications

Papers included in the thesis

1. Shaoteng Liu, Axel Jantsch and Zhonghai Lu, "Parallel Probing: Dynamic and Constant Time Setup Procedure in Circuit Switching NoC," *In Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'12), pages: 1289-1294, Dresden, German 2012.*

2. Shaoteng Liu, Axel Jantsch and Zhonghai Lu, "Parallel probe based dynamic connection setup in TDM NoCs," *In Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'14), pages: 239:1–239:6, Dresden, German 2014.*

3. Shaoteng Liu, Zhonghai Lu and Axel Jantsch, "Highway in TDM NoC," *In Proceedings of the Ninth ACM/IEEE International Symposium on Networks-on-Chip (NoCS'15), pages: 15:1–15:8, Vancouver, Canada, 2015 (**The best paper award**).*

4. Shaoteng Liu, Axel Jantsch and Zhonghai Lu, "A Fair and Maximal Allocator for Single-Cycle On-Chip Homogeneous Resource Allocation," *IEEE transactions on very large scaled integration systems (TVLSI), vol 22, no. 10, pages: 2229–2233, 2014.*

5. Shaoteng Liu, Axel Jantsch and Zhonghai Lu, "MultiCS: Circuit Switched NoC with Multiple Sub-Networks and Sub-Channels," *Journal of Systems Architecture (JSA), vol 61, issue 9, pages: 423–434, 2015.*

6. Shaoteng Liu, Axel Jantsch and Zhonghai Lu, "Analysis and evaluation of circuit switched NoC and Packet Switched NoC," *IEEE Euromicro Conference on Digital System Design (DSD'13), pages: 21–28, 2013*

Papers not included in the thesis

1. Meganathan Deivasigamani, Shaghayeghsadat Tabatabaei, Naveed Mustafa, Hamza Ijaz, Haris Bin Aslam, Shaoteng Liu, Axel Jantsch, "Concept and design of exhaustive-parallel search algorithm for Network-on-Chip," *IEEE International SOC Conference (SOCC'11), pages:150–155, 2011*

2. HuaQiu Yang, LiGuang Chen, Shaoteng Liu, HaiXiang Bu, Yuan Wang, JinMei La, "A flexible bit-stream level evolvable hardware platform based on FPGA," *NASA/ESA Conference on Adaptive Hardware and Systems (AHS'09), pages:51–56 2009.*

3. Shaoteng Liu, Jinmei Lai, Liguang Chen, Jiarong Tong, Lichun Bao, "An Evolvable and Reconfigurable Image Filter," *Journal of Fudan University (Natural Science), issue 06, 2010.*

4. Shuo Li, Jamshaid Malik, Shaoteng Liu, Ahmed Hemani. "A code generation method for system-level synthesis on ASIC, FPGA and many-core CGRA," *In Proceedings of the First International Workshop on Many-core Embedded Systems, pages: 25–32, 2013*

Chapter 1

Introduction

This chapter begins with introducing the background and the classification of NoCs. Then, it gives an overview about our works on circuit switched NoC and outlines the author's contributions in the papers enclosed to the thesis.

1.1 Background

The development of semiconductor technology shrinks the feature size of transistors while enlarging the die size. As a result, more devices and Intellectual Property (IP) cores can be integrated on a single chip. However, on-chip interconnects tend to get worse, since the shrinking feature size will cause the increase of wire delay and crosstalk. Besides, as the number of devices increases, more communication bandwidth is required. Therefore, new communication architectures are required to overcome the limitations on wire delay and bandwidth.

In the past, a single bus was used to connect all the devices (IP cores) on a chip, as illustrated in Figure 1.1. Data sent out by a device is broadcast on the bus and can be received by all other devices. At one time, only one device is allowed to transmit data to the bus. An arbiter decides which one can transmit. Bus is not a scalable solution for communication since the service rate of each device decreases as the number of devices connected to the bus increases. Besides, the broadcast way of transmitting data is inefficient. Moreover, as the number of devices grows, the bandwidth of a bus may decrease due to the growing wiring delay, which is the result of

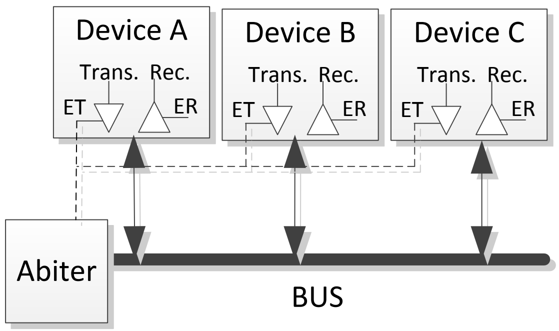


Figure 1.1: The structure of a typical bus

the increase in wire length and capacitance. Consequently, bus becomes the communication bottleneck as the number of devices on a chip goes up.

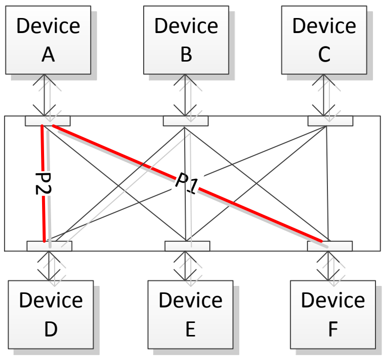


Figure 1.2: AXI interconnect: an example of point-to-point communication architectures

To overcome the limitation of the single bus based communication architecture, crossbar based point-to-point communication architectures are proposed. For example, Advanced Extensible Interface (AXI) interconnect [1] allows point-to-point communications via a big crossbar structure, as depicted in Figure 1.2. This architecture enhances the communication bandwidth by allowing several data flows on-going simultaneously inside a crossbar. However, one significant drawback of this solution is that the crossbar structure does not scale up well. Firstly, the number of wires and transistors consumed by the crossbar and the arbiters scales up with $O(n^2)$, where n is the number of connected devices. Secondly, inside this architecture, the wiring delay between different sources and destinations are non-uniform since the physical distance varies. As illustrated in Figure 1.2, the distance of path $P1$ is longer than that of $P2$. As the number of devices scales up, such non-uniformity issue becomes more severe, resulting in the design of the clock system and synchronization scheme very challenging.

Because of all of these limitations and challenges, more advanced communication architectures are required. Network-on-Chip (NoC) emerges under this background. As a promising technology, NoC is expected to exceed the limitation on the communication bandwidth, overcome the increasing wiring delay problem and become a scalable communication solution. NoC, as its name suggests, intends to build a communication network inside a chip. The concept of communication network is well-known since it has been widely used in our everyday life, for example, telephony network and Internet.

NoC inherits some of the merits from those macro-networks like Internet or Asynchronous Transfer Mode (ATM) network. For example, it also distributes a number of routers between all the terminal devices and interconnects the routers and terminal devices according to a certain topology, as illustrated in Figure 1.3. We name the wires that interconnect two routers/devices as a link. Inside a NoC, point-to-point data delivery is relayed by routers. Thus, instead of connecting source and destination directly by using long wires, data transmitted from a source node takes multiple hops and traverses multiple short links to reach its destination. This approach can solve the complexities caused by long wires and thus increases the scalability. Besides, in many cases, NoCs take a regular topology, e.g. mesh (as illustrated in Figure 1.3) or torus, which are very suitable for expanding. Moreover, inside a NoC, multiple data flows can co-exist on different links, or take turns to use one link. In such a way, the communication resources are exploited very efficiently. When compared with other approaches, NoC can provide larger communication bandwidth with relatively less area cost and less wiring complexity.

NoC also presents some varieties from those macro-networks. For example, NoC design can utilize more wires for a link and operating at very high clock speed since it operates in an on-chip environment. Besides, NoC design focuses more on router delay, area cost, and power consumption. For example, NoC design restricts the usage of complicated routing/arbitration/flow control algorithms, deep buffers and so on. Therefore, existing macro-network solutions are often infeasible for NoC design. We

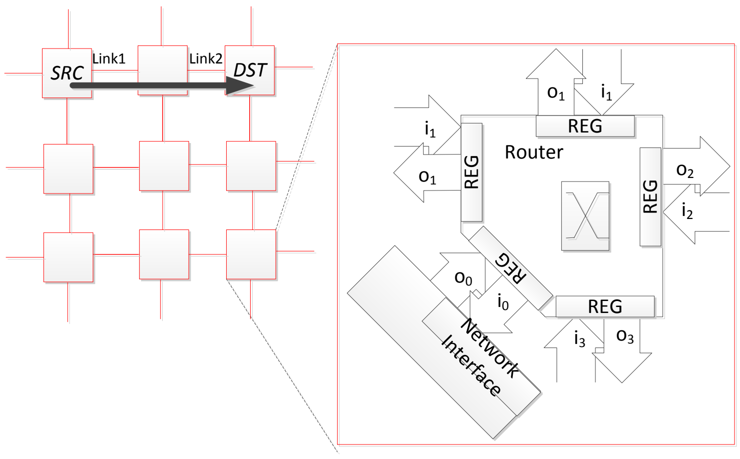


Figure 1.3: The overview of NoC

have to consider new communication architectures and components that are suitable for on-chip environments.

1.2 Classification of NoCs

At present, no well-formulated standard exists for guiding NoC design. There are many published NoC architectures in the literature. Generally speaking, these NoC architectures can be classified into two categories: packet switched NoC [2, 3, 4, 5] and circuit switched NoC [6, 7, 8, 9, 10].

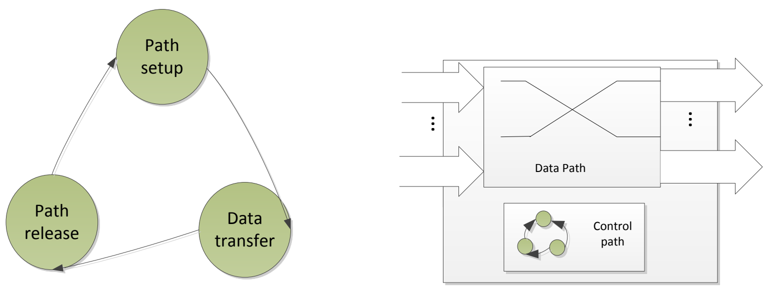
In a packet switched NoC, a data flow is split and wrapped into blocks of a certain size, called packets. Normally, a packet is composed of a header and a payload. The header contains the information needed by the routers to direct the packet to its destination. The payload contains the information a source node wants to deliver. Communication resources, such as buffers and channels inside a router are allocated to individual packets. When a packet arrives at a router, resources are allocated. When the packet leaves, the allocated resources are reclaimed. In packet switched NoCs, flow control is often required between every two hops. This is called hop-by-hop flow control.

Links in a packet switched NoC are normally shared in a work-conserving manner [11] in the sense that a link is never idle if there are packets to transmit. Packets of different data flows take turns to use a shared link. An arbiter decides the order. When a packet gets its turn, a certain amount of its data is delivered by the link. Otherwise, the packet has to be buffered. Depending on the buffering policy and the amount of data allowed to deliver each time, packet switched NoC can further be classified into sub-categories like wormhole [12, 13, 14, 3], virtual cut-through [15, 16], deflective [17, 2, 18] and so on.

The main difference between circuit switched NoC and packet switched NoC is that, a circuit switched NoC pre-allocates a path for a data flow before the data transfer launches. In circuit switched NoC, since all the required communication resources are pre-allocated, there is no contention and thus no need for arbitration during the data transfer process. Besides, circuit switched NoCs often do not need a header for directing individual packets since each data flow is transmitted along a pre-allocated path between the source and the destination.

In a circuit switched NoC, normally links are shared in a non-work-conserving manner [11] between different data flows, if there are link sharings, e.g. in TDM NoC. The bandwidth of a link is pre-allocated to a data flow and each flow can only use its allocated share. In circuit switched NoC, flow control is often only needed between the source and destination. This is called end-to-end flow control [19].

1.2.1 Fundamentals about circuit switched NoC



(a) General operating flow (b) Router architecture overview

Figure 1.4: The overview of circuit switched NoC

As depicted in Figure 1.4a, the general operating flow of a circuit switched NoC consists of 3 phases: *path setup*, *data transfer*, and *path release*. During the path setup phase, link resources are allocated to a data flow to build up a connection from the source to the destination. During the data transfer phase, data is switched by routers on the established path. After data transfer is finished, the routers reclaim the allocated resources. The three phases are distinctively isolated. Thus, the router architecture of a circuit switched NoC can often be divided into a data path and a control path (plane), as suggested by Figure 1.4b. The control plane is used in path setup and reclaim phase. It controls the allocation and configurations of communication resources such as channels and crossbars. The data plane is used in the data transfer phase and responsible for switching data to the established path.

In the past, packet switched NoCs have been explored more thoroughly and intensively. However, circuit switched NoC has some appealing merits and could be preferable under certain traffic or service scenarios. Compared with packet switched NoC, although circuit switched NoC has path setup overhead time, it can offer guaranteed throughput and latency. Besides, it can also present lower hardware complexity and higher energy efficiency, and may work at a higher clock frequency.

Link bandwidth sharing techniques

Time Division Multiplexing (TDM) and Spatial Division Multiplexing (SDM) are the two techniques frequently adopted by circuit switched NoC to share the bandwidth of a link between different connections. With TDM technique, connections can take turns to use a link. Each connection can only use a link for a predetermined fraction of time. With SDM technique, a link is physically divided into sub-links. Each connection reserves a number of sub-links and use them exclusively. We will discuss the two techniques in Chapter 3 and Chapter 4 in detail.

Besides TDM and SDM, some other circuit switched NoC designs [20, 21] utilize per-connection virtual channels and round-robin arbitration to share links. To offer guaranteed throughput for each connection, this technique pre-allocates virtual channels to connections and puts limitations on the virtual channels that share a link. However, virtual channels are expensive resources, since they consist of buffers, multiplexers, demultiplexers and require separate hop-to-hop flow control. Thus, the hardware cost is high with this technique.

1.3 Contributions

In this thesis, we focus on circuit switched NoC. We present our research on advanced path setup methods and data switching techniques. The thesis summarizes a collection of papers, which are grouped into three blocks: *Dynamic connection setup, Time division multiplexing, Spatial division multiplexing*. Each block corresponds to one chapter. We concentrate on introducing the author's contributions in these chapters. In the following, we summarize our contributions:

- Dynamic connection setup
  Paper A Shaoteng Liu, Axel Jantsch and Zhonghai Lu. Parallel Probing: Dynamic and Constant Time Setup Procedure in Circuit Switching NoC. *In proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'12), pages 1289-1294, Dresden, German, 2012.*

  In this paper, we proposed a novel parallel probe based path searching algorithm. This algorithm can search the entire network topology and find an available path from a source to a destination in constant time. We implemented this algorithm in a circuit switched NoC for dynamic connection search and set up, by solving tricky issues like live-lock inside this setup method. We also proposed and evaluated several connection search and setup policies. Compared to previous works, our design can reduce the setup time and enhance the setup success rate. Moreover, the router in our design has a concise structure with an inexpensive and efficient implementation.

  Author's contribution: The author proposed the parallel probing algorithm and utilized this algorithm in a circuit switched NoC for connection setup, made the hardware implementation, established experimental platform to evaluate the design, and wrote the manuscript.

- Time division multiplexing
  Paper B. Shaoteng Liu, Axel Jantsch and Zhonghai Lu. Parallel probe based dynamic connection setup in TDM NoCs. *In proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'14), pages 239:1–239:6, Dresden, German, 2014.*

In this paper, we proposed a double time-wheel technique that used in TDM based circuit switched NoC to facilitate the two-way communication of a TDM connection. A slot-table is shared by both upward and downward messages of a connection. Based on this technique, we introduced a probe based connection setup method into TDM circuit-switched NoC. Our design provides shorter connection setup delay and higher success rate while presenting lower hardware complexity than any previously known method.

Author's contribution: The author proposed the idea and made the hardware implementation of the double-time wheel technique. The author also proposed a probe based path setup method that can be applied in TDM NoC for connection setup. Besides, the author established an experimental platform to evaluate the design and wrote the manuscript.

Paper C Shaoteng Liu, Zhonghai Lu and Axel Jantsch. Highway in TDM NoC. *In Proceedings of the Ninth ACM/IEEE International Symposium on Networks-on-Chip (NoCS'2015), pages: 15:1-15:8, Vancouver, Canada, 2015 (The best paper award)*

In this paper, we proposed a novel highway technique that can enhance the performance of TDM NoCs. The proposed technique can dynamically set up highways without contention. Once highways are built, the delivery of data flows can be accelerated by utilizing unallocated and idle TDM slots of links, while preserving the guarantee on minimum bandwidth and data packet order. This highway technique has no dependency on the TDM NoC architectures and introduces no additional traffic. It is a generic technique that can be applied in many different kinds of TDM NoCs.

Author's contribution: The author developed the concept of highway, defined detailed procedures for highway setup, transfer and release. The author also solved tricky hardware implementation issues, utilized both synthetic traffic patterns and benchmarks for test and evaluation. The author wrote the manuscript.

- Spatial division multiplexing
  Paper D Shaoteng Liu, Axel Jantsch and Zhonghai Lu. A Fair and Maximal Allocator for Single-Cycle On-Chip Homogeneous Resource

Allocation. *IEEE transactions on very large scale integration systems (TVLSI), 22.10, pages: 2229-2233, 2014.*

In this paper, we exposed a special allocation problem in NoC environment and named it as the Homogeneous Resource Allocation (HRA) problem. We developed a novel Waterfall (WTF) allocator for the homogeneous resource allocation. The WTF allocator provides maximal matching quality while keeping the strong fairness guarantee. It can have a loop-free structure and solve a homogeneous resource allocation problem within one clock cycle. It achieves strong fairness and offers better performance and lower area than known solutions.

Author's contribution: The author proposed the homogeneous resource allocation concept and the idea of the waterfall allocator. The author also efficiently implemented the allocator in hardware, proved its advantages in matching quality and fairness by comparising with existing solutions. The author wrote the manuscript.

Paper E Shaoteng Liu, Axel Jantsch and Zhonghai Lu. MultiCS: Circuit Switched NoC with Multiple Sub-Networks and Sub-Channels. *Journal of System Architecture (JSA), vol 61, issue 9, pages: 423-434, 2015.*

In this paper, we explored the methods of organizing multiple physical channels in a circuit switched NoC. We proposed a Multi-channel and Multi-network Circuit Switched (MultiCS) to study channel partitioning and configuration policies. Based on the analysis and experiments results, we revealed the benefits and burden of using different number of channels and configurations.

Author's contribution: The author proposed the MulitCS NoC, designed different channel partitioning and configuration policies, analyzed and evaluated these policies, and wrote the manuscript.

Paper F Shaoteng Liu, Axel Jantsch and Zhonghai Lu. Analysis and evaluation of circuit switched NoC and packet switched NoC. *In proceedings of IEEE Euromicro Conference on Digital System Design (DSD'2013), pages: 21-28, 2013*

In this paper, we compared circuit switched NoC against packet switched NoC. We showed that performance decreases for packet switched NoC as the packet size increases, whereas it increases for circuit switched

NoC. We revealed that circuit switched NoC can operate at a higher clock frequency than packet switched NoC and thus could be better than packet switched NoC in some cases.

Author's contribution: The author designed a circuit switched NoC and compared with packet switched NoC based on speculation, circuit-level analysis, and evaluation. The author built the experiment platform, conducted all the experiments and wrote the manuscript.

The remainder of the thesis is structured as follows. Chapter 2 introduces our research on the dynamic connection setup method. Chapter 3 summarizes our works on TDM based circuit switched NoC. In Chapter 4, we present our works related to SDM based circuit switched NoC. Finally, we conclude the thesis and discuss future works in Chapter 5.

Chapter 2

Dynamic Connection Setup

In this chapter, we introduce how to set up connections in circuit switched NoCs. Particularly, we will present our novel parallel probing algorithm, and show how to use this algorithm to dynamically search and set up connections in a circuit switched NoC [Paper A].

2.1 Introduction

How to set up connections is vital to a circuit switched NoC, since data transfer in circuit switched NoC relies on established connections. Connection setup methods affect the performance and resource utilization of a circuit switched NoC. Connection setup methods can be classified into *static scheduling methods*, and *dynamic setup methods*.

In this chapter, we focus on connection setup methods. In order to simplify our illustration, in the following discussions, we just consider the case that a physical single directional link can only be used by one communication channel without sharing with others. We will discuss link-sharing techniques such as TDM and SDM in Chapter 3 and Chapter 4, respectively.

2.1.1 Static scheduling method

Static scheduling methods schedule the channel resources of a network for connections at compilation time [22, 23]. During the past decade, many NoC architectures supporting static connection scheduling have been proposed

[24, 8, 25, 26], based on a variety of static scheduling algorithms [27, 28, 29, 30, 31, 32, 33, 34, 35, 36].

Z.Lu and A.Jantsch in [37] has formulated a static connection scheduling problem. According to [37], inside a network, all the possible paths of a connection compose a tree-like search space, and a static scheduling algorithm's responsibilities are the following. 1) Explore the solution space and schedule a path for a connection. 2) Make sure all the connections scheduled on a link summed together does not exceed the link's bandwidth. 3) Try to optimize the path schedule. A good schedule should satisfy as many communication requirements as possible, and consume as less communication resources as possible.

The advantage of static scheduling methods is that advanced algorithms can be applied to optimize the scheduling of connections. However, static schedule methods often suffer from the following disadvantages. Firstly, they assume that all communication needs and connection requirements are known at compile time. Thus, they are not well suited for applications like H.264 [38] with requirements for dynamic communication setups or dynamic traffic mixes of applications. Secondly, those static scheduling algorithms are often complicated and take several iterations to produce an optimal solution. As the number of nodes inside a circuit-switched NoC grows up, the required processing time and storage area for scheduling connections can increase exponentially. Thirdly, connections scheduled by static methods cannot be aware of the dynamic fluctuation of network traffic load. Thus, the network bandwidth utilization is often sub-optimal.

2.1.2 Dynamic setup method

Dynamic connection setup methods [39, 40, 41, 42, 43, 44, 45, 46, 47] were proposed to overcome the shortcomings of static scheduling method. Dynamic connection setup methods allocate and release connections at runtime, according to dynamic communication requirements and network status. With dynamic methods, network resources can be dynamically and efficiently allocated and reclaimed.

Dynamic setup methods can further be classified into *centralized* methods and *distributed* methods.

Centralized methods

In centralized methods, a resource allocation algorithm is running inside a special *coordinator node* of a network. This node manages all the network resources and connections. All the other nodes need to send requests to the coordinator node in order to set up or release a connection. When the coordinator node receives a connection setup request, it will make a quick check on available channels and decide how to allocate channels inside the network to the connection. When it receives a connection release request, it will reclaim the allocated channels. The resource allocation algorithm can be either running as a software inside a processor like [39, 40, 41, 48], or running on hardware based accelerators [43, 44]. Normally, hardware accelerators based algorithms are about 100-1000 times faster than running as software inside a processor.

After an allocation decision is made by the allocation algorithm inside the coordinator node, *channel reservation* process starts. The channel reservation process reserves and configures channels in different routers according to the allocation decision on a connection. Two different channel reservation approaches are frequently used in centralized methods. The first approach utilizes a dedicated network to distribute allocation decisions to routers inside a NoC [8, 26], and directly initiates the configuration processes inside each router. With the second approach [43], the coordinator node firstly sends the channel allocation decision to the source node of a connection. Then, the source node will compose and send out a reservation packet to the destination by using contention-free source routing method. This reservation packet carries all the channel reservation and routing information, and reserves the corresponding channels inside each router on its traveling path. When the reservation request reaches the destination, the connection has been established.

The main limitation with centralized allocation methods is the lack of scalability. The coordinator node needs to handle all the setup/release requests and distribute allocation decisions from/to the entire network. Such multiple-to-one and one-to-multiple traffic patterns can become the performance bottleneck. Besides, resource allocation algorithms cannot be occupied by one setup request for a long time. Otherwise, the following setup requests will be delayed and blokced, resulting in the overall performance degradation. For example, if the allocation algorithm fails to find free channels for a setup request, it has to discard the setup request immediately, rather than retry the failed request repeatedly.

Distributed methods

Distributed setup methods [46, 42, 45, 49] are proposed to overcome the scalability issue in centralized methods. With distributed methods, each source node can setup and release a connection independently, without the inquiry of a coordinator node. To establish a connection, the source node generates a set up request and sends it out. Compared with the reservation packets used in centralized methods, the setup request in this case is more concise. It contains only source and destination addresses, instead of all the routing and channel allocating information. The setup request is routed towards the destination by routers. Each router computes the next hop of the setup request, and forwards the setup request to the next router. A channel inside a router is reserved by the setup request when it passes the router. When a setup request reaches the destination, a connection has been established, since the setup request has reserved channels by the routers along one path from the source to the destination. In distributed methods, the channel allocation and channel reservation happen at the same time.

Routing algorithm inside each router used for routing the setup request is crucial to distributed setup methods. It decides the speed and efficiency of setting up a connection to the destination.

In this chapter, we concentrate on dynamic distributed connection setup methods. In the following discussions, the term *path* refers specifically to *minimal path*.

2.2 Parallel probe based connection setup

2.2.1 Problem description

Inside a network, multiple minimal paths exist between a source node and a destination node. As described in Figure 2.1, we use a mesh topology to illustrate such path diversity. For example, from the source $node(0,0)$ to the destination $node(2,2)$, there are multiple path choices. Specifically, since there is 1 path from $node(0,0)$ to $node(1,0)$ and 1 path from $node(0,0)$ to $node(0,1)$, there are in total $1+1=2$ paths from $node(0,0)$ to $node(1,1)$. With this method, since there is 1 path from $node(0,0)$ to $node(2,0)$ and 2 paths from $node(0,0)$ to $node(1,1)$, the number of possible paths from

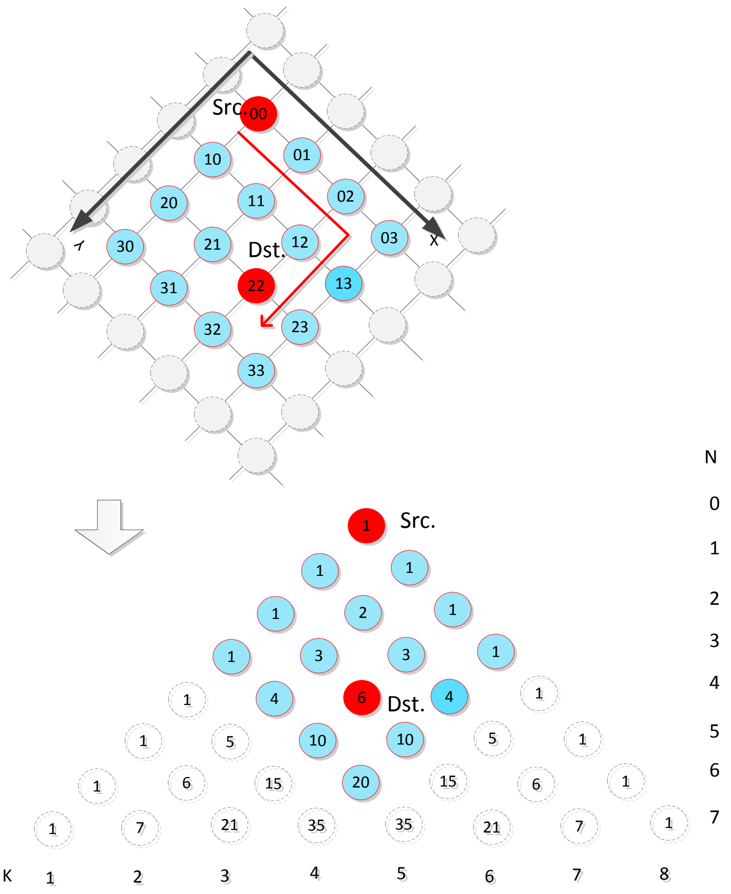


Figure 2.1: Path diversity expressed by using Pascal's triangle

$node(0,0)$ to $node(2,1)$ equals $1+2=3$. Similarly, we can find that there are also 3 paths from $node(0,0)$ to $node(2,1)$. Finally, the number of paths from $node(0,0)$ to $node(2,2)$ is $3+3=6$.

Therefore, we may conclude that the number of possible minimal paths from a source node to a destination node in mesh topology can be expressed by using a Pascal's triangle [50], as Figure 2.1 suggests. Suppose that the source node has coordinate $(0,0)$, and the destination node is (x,y). The Manhattan Distance [51] between source and destination node is D, where $D=x+y$. We have the number of paths m as

$$m = C_D^x = C_D^y = \frac{D!}{x!(D-x)!} = \frac{D!}{y!(D-y)!} \tag{2.1}$$

By using Eq.2.1, we can simply get that there are $C_4^2 = 6$ paths from $node(0,0)$ to $node(2,2)$, which is in accordance with our previous analysis. As the distance between source and destination increases, the number of possible paths increases dramatically. For example, in a 4 by 4 mesh, the number of shortest possible paths from a corner node to reach the node on its diagonal corner is 20. For a 5 by 5 mesh, this number becomes 70. For an 8 by 8 mesh, it becomes 3432.

The routing algorithm used to route setup requests determines how many possible paths can be searched. Since routing algorithms are normally implemented inside individual routers and in hardware, many previous works [42, 45, 46, 52, 53, 54] just seek to apply simple deterministic routing algorithms such as X-Y routing. However, deterministic routing algorithms can only search one fixed path from source to destination. It fails to explore the aforementioned path diversity and thus significantly limits the connection setup successes probability and the setup efficiency.

In order to exploit the path diversity, powerful routing algorithms need to be developed for the setup request. It is possible to implement powerful distributed routing algorithms with reasonable hardware cost. For example, in [49, 55], Pham et al. proposed a depth-first path search algorithm and implemented it in hardware. With this algorithm, a setup request firstly travels along a certain path from source to destination. If the setup request cannot continue along the path, it will backtrack one or several nodes and try another path. This depth-first algorithm can do an exhaustive search among all the possible minimal paths. If a free path exists, it will find it. However, as a depth-first search algorithm, it takes too long time to search over all the possible paths. As Eq.2.1 suggests, the time complexity for the worst case is $O(D!)$, where the D is the distance between source and destination.

Towards a more efficient solution, we propose a more powerful path setup algorithm which is based on a breadth-first way for path search. Unlike the depth-first setup algorithm proposed by Pham et al. [49, 55], our breadth-first setup algorithm can exploit the parallelism of hardware. It can search all the possible paths at one time in parallel. The time complexity for such a search is only $O(D)$. We name our algorithm as *parallel probing*.

2.2.2 The parallel probing dynamic setup algorithm

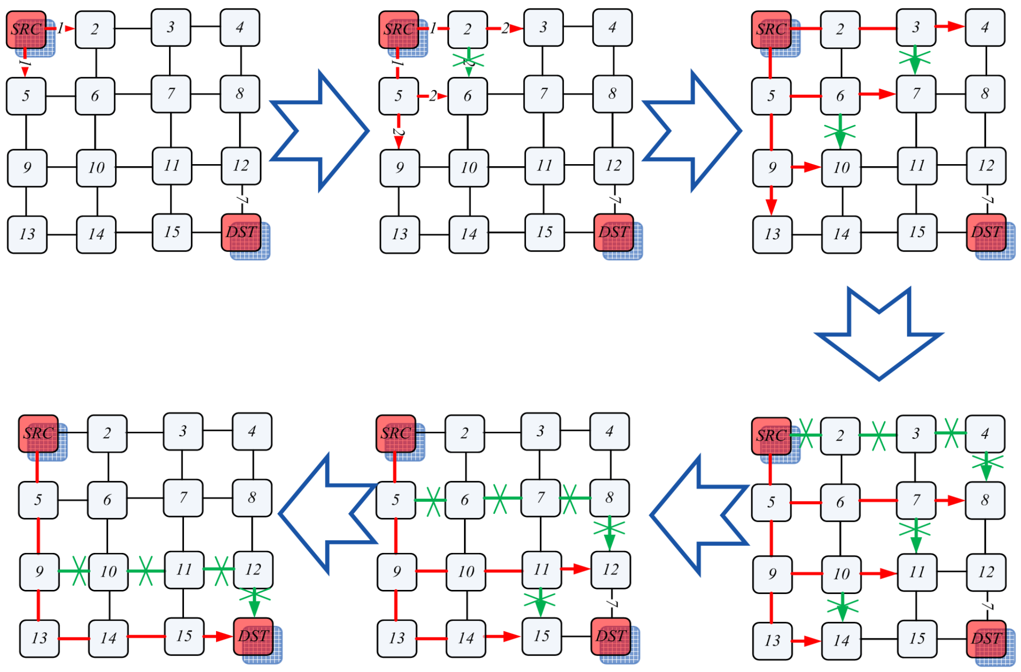


Figure 2.2: Set up a connection with the parallel probing search algorithm

We use an example to illustrate the key idea of our parallel probing path search algorithm, as illustrated in Figure 2.2. In Figure 2.2, a source node (node 1) tries to set up a connection to the destination node (node 16). At the first cycle, the source node sends out two setup probes along two productive directions toward the destination. Each probe consists of source address and destination address to guide its routing. At the second cycle, when the two probes arrive at node 2 and node 5, each probe splits into two probes, so there are four probes in total continuing travelling toward destination along all the minimal paths. Following such a way, a number of probes will be triggered out to search all the possible paths in parallel. As a probe travels, it reserves the channels it has passed.

However, since we only need one path from source to destination in the end, our algorithm eliminates redundant channels as soon as possible. This is how our algorithm is different from the flood based setup algorithms [56]. One policy of our algorithm is that when two probes carrying the same setup request meet at the same node, one probe dies, and the channels reserved by the dead probe alone will be canceled. For example, as the second picture of Figure 2.2 describes, two probes meet at node 6 that one probe dies. With this policy, the channel between node 2 and node 6 will be canceled, whereas the channel between the source node and node 2 remains, because it is still needed by the probe that has travelled further to node 3. We identify these released redundant channels with cross markers, as shown in Figure 2.2. Following this way, in an ideal case (minimum paths exist and no contention happened between probes of different setup requests), only two probes can enter into the destination node, which is node 16. Then one probe dies and cancels its reserved path. In the end, only one path is left between the source and the destination.

We summarize the general rules of our parallel probing algorithm as follows:

1. **Probe split rule**

   It is inspired by the idea "cell division". At first, only one setup probe is sent out by the resource connected to the source node. When a probe enters into a node and finds multiple productive outputs towards the destination (e.g. it can have two productive outputs in a mesh topology), it will split into multiple probes to travel through all these output channels in parallel, with one probe per output for an exhaustive search and setup.

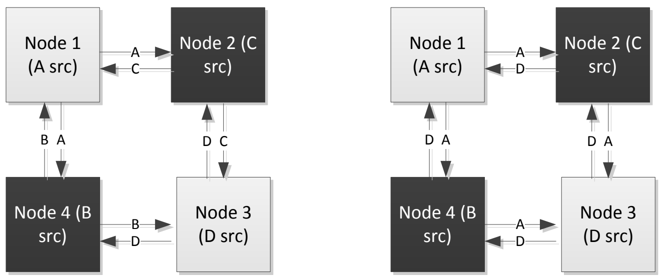
2. **Probe cancel rule**

   This rule reduces the network resource consumption during a connection setup since the probe split rule generates too many probes and occupies too many redundant channels. The probe cancel rule mandates the release of the redundant parallel paths as soon as possible. This rule requires that 1) whenever two probes of the same request meet, one of them dies, 2) if a probe cannot proceed because of contention or obstruction, the probe dies as well. The channels reserved by a dead probe is released and reclaimed immediately. The release action proceeds backward hop by hop along the path reserved the dead probe until reaches a channel that is still in use by an active probe.

Live-lock avoidance

Our parallel probing algorithm has no deadlock [57, 58] issue. This is due to that if a probe cannot proceed, it will be canceled immediately and all the resources reserved by the probe is released and reclaimed. When probes do not hold resources and wait, there is no deadlock.

However, live-lock [59] may exist in our algorithm, since contentions for resources between probes belonging to different connection setup requests may create live-lock. As illustrated in Figure 2.3a, if the four connection setup requests A: 1 → 3 (it means node 1 trying to set up a connection to node 3), B: 4 → 2, C: 2 → 4, D: 3 → 1 are sent out simultaneously, then they will block each other. None of them can succeed. Retrying in a deterministic manner will not help.



(a) Live lock (b) Priority preemption method to solve live-lock

Figure 2.3: Live-lock and priority preemption based live-lock avoidance method

In paper A, we proposed a priority based preemption [60] method to deal with live-lock. With this method, the live-lock illustrated by Figure 2.3a can be resolved. E.g., if setup probe A's priority is higher than C, it can preempt the channel reserved by C between node 2 and node 3.

In paper A, we use a setup request's retry times and its source node ID as priority values. Actually, we can use anything as a priority, just making sure that two conflicting requests do not have the same priority.

2.2.3 Implementation of the parallel probing algorithm

Operating phases in a connection setup process

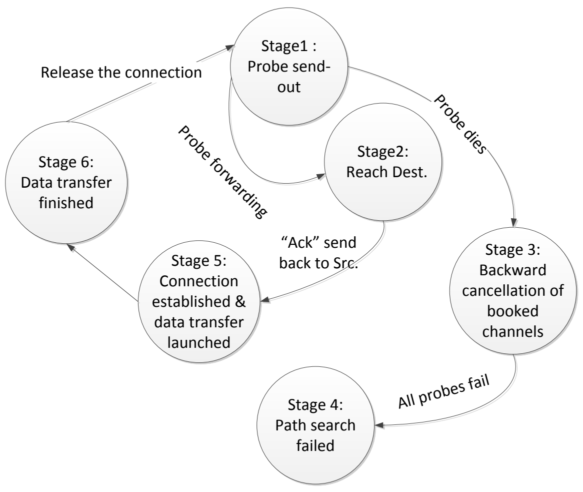

Stage1 : Probe send-out
Release the connection
Probe dies
Stage 6: Data transfer finished
Probe forwarding
Stage2: Reach Dest.
"Ack" send back to Src.
Stage 5: Connection established & data transfer launched
Stage 3: Backward cancellation of booked channels
All probes fail
Stage 4: Path search failed

Figure 2.4: General operating phases of connection setup with parallel probing algorithm

The general operating phases of the parallel probing algorithm is depicted in Figure 2.4. After probes are sent out from the source node, they proceed toward destination while splitting and dying. When a probe reaches destination, "Ack" signals will be sent back to the source node along the established path to confirm channel reservations and launch data transfer. If probes failed, channels reserved by dead probes will be released by "Nack" signal. If all the probes die, depending on different setup polices, the setup request may be retried or dropped.

According to our implementation in paper A, a probe takes 2 cycles per hop to move forward while the backward Ack/Nack takes 1 cycles per hop. Thus, for a single search, it takes $2D$ cycles for a probe to reach the destination and D cycles for sending back the ANS signal. There is also an additional 6 cycles that are spent in the network interface as an overhead (D is the hop distance between a source and a destination). In other words, after sending out probes, a source node is notified about the result of the connection setup endeavour within $3D + 6$ cycles.

Setup policies

We proposed three setup policies, which are *no retry*, *retry for free path* and *retry until success.*

No retry policy only tries once for a setup request, and gives up the request if the endeavor is failed. Therefore, it spends at most $3D + 6$ cycles on a setup request.

Retry for free path policy will retry the failed setup requests several times before give-up. The interval between every two adjacent reties is fixed to $3 * D_{max} + 6$ cycles, where D_{max} is the longest distance between two nodes in a topology. In a $n * n$ mesh, $D_{max} = 2 * (n - 1)$. After each retry, the priority of the setup request is increased by 1. Suppose there are K source nodes inside a network and each node at most serves one setup request at a time. Then after K retries, if the setup request still cannot succeed, it will be dropped. Since after K retries, the setup request will have the highest priority inside the network. It will win every arbitration inside each router according to our priority preemption based arbitration method. If it still cannot succeed, it means that all the possible paths are occupied.

Retry until success policy has no limitation on retry times of a failed setup requests. It keeps retrying a failed request until it becomes successful. The retry interval may be set to zero, which means a source node will retry a failed setup request immediately when it is notified about the failure.

2.3 Future work

Our parallel probing algorithm is an adaptive routing algorithm. It can search over all possible minimal paths and find an available path within a time complexity $O(D)$. Because of these good properties, we can explore the parallel probing algorithm in the following three directions in the future.

1. *Using the parallel probing algorithm in topologies other than mesh.* In Paper A, the parallel probing algorithm is only implemented for a mesh topology. However, dynamic connection setups in regular topologies, such as torus [61], fat tree, benes [62], clos and butterfly [63], should also benefit from our algorithm. The principle of the parallel probing algorithm is to use as many as possible probes to search among all available shortest paths in parallel, while cancelling redundant paths as soon as possible. Thus, on a network topology, as long as each router knows how to forward incoming probes toward the destination along minimal paths, our parallel probing algorithm could work.

2. *Using the parallel probing algorithm for fault tolerance purpose.* It is possible to utilize our parallel probing algorithm for fault tolerance purpose, since our algorithm can search all the minimal paths simultaneously and in parallel. Even if faults happen on some of the paths, our algorithm can still find out the fault-free path to build up a connection.

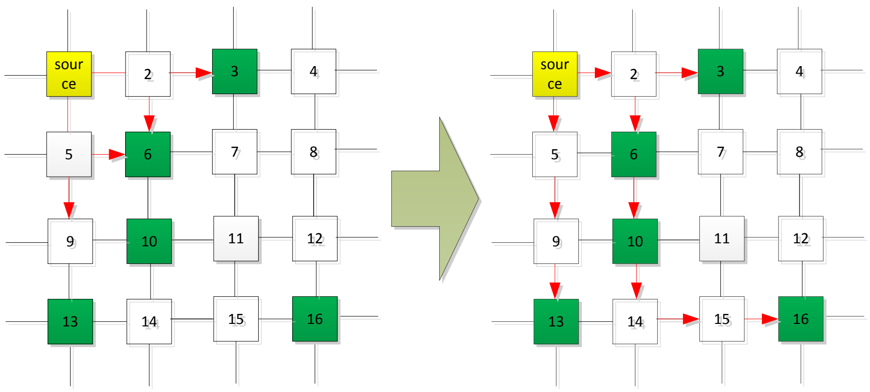


Figure 2.5: Establish a multicast tree by using the parallel probing algorithm

3. *Using the parallel probing algorithm for multicast communication purpose.* Multicast communication refers to that information sent by a source node is addressed to a group of destinations simultaneously. To support multicast in circuit switched NoC, a source node needs to be connected to multiple destinations. Instead of establishing multiple point-to-point connections, multicast can take a tree-like structure [64, 65] to connect the source node as root and the destination nodes

as leaves. In such a way, multicast can use communication channel resources more efficiently, since the consumption of channel resources can be reduced.

Our parallel probing algorithm is suitable to establish such a tree-like communication structure for multicast. As illustrated in Figure 2.5, with one setup attempt, our parallel probing algorithm is possible to establish a tree-like structure which connects all the destinations (green nodes) to the source node (the yellow node).

Chapter 3

Time Division Multiplexing

This chapter summarizes our research on TDM NoCs. Particularly, we will introduce our work on double time-wheel technique and its usage in the setup of TDM based connections [Paper B]. We will also introduce our highway technique and present its advantages [Paper C].

3.1 Introduction on TDM NoCs

3.1.1 Time division multiplexing technique

Time Division Multiplexing (TDM) [32, 35, 37, 24, 66, 34] technique has been widely used for guaranteed data transfer in NoCs. In TDM NoC, a physical link can be contention-freely shared by different connections, with each connection reserving one or multiple specific time slots in a finite repeating time window. Each slot is exclusively reserved by one connection. A connection can span many links from source to destination, by reserving slot(s) at each of the links in a consecutive manner. For example, as illustrated in Figure 3.1, connection v1 spans link 1 and link 2. If slot 0 and slot 2 of link 1 are reserved by v1, then slot 1 and slot 3 of link 2 must be reserved by v1. TDM circuit switching treats the entire NoC as a single shared resource with a single arbiter. In other words, packets wait only at the ingress Network Interface (NI) until their reserved slots come, after which they progress without contention to the egress NIs, with minimal latency [67].

In the following discussions of this chapter, we define a *TDM channel* as a simplex link between two routers with associated buffers in a particular

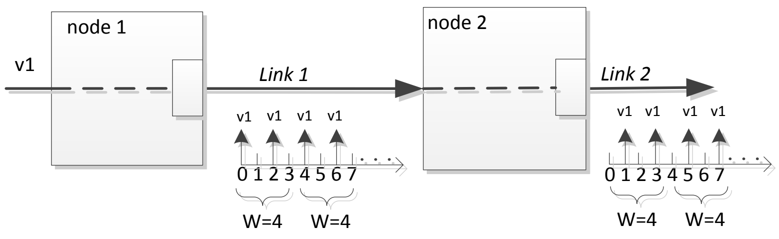


Figure 3.1: Illustration of TDM concept

time slot. Thus, the same link in different time slots belong to different channels.

3.1.2 Data path configuration methods

In TDM NoC, data path configuration refers to how to configure the crossbar inside each router to switch data flits of different traffic flows correctly through the pre-reserved channels. Generally speaking, there are two kind of methods: *source routing based configuration* [68, 67, 69] and *slot-table based configuration* [8, 70, 37].

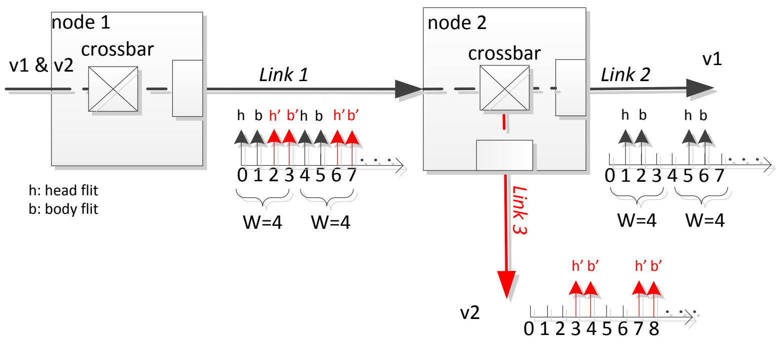


Figure 3.2: Source routing based configuration in TDM NoCs needs reserved slots of a connection to be adjacent

With source routing based configuration method, a data flow is wrapped into packets, each of which contains a head flit. The head flit carries all the crossbars' configurations of a connection. The head flit configures every cross-bar as it travels, with one crossbar per router. Body flits of a packet follow the path paved by the head flit. Thus, it requires that the allocated slots of a connection must be adjacent, so that body flits can follow closely without interval. For example, as Figure 3.2 suggests, if a head flit of connection $v1$ is delivered at slot 0 of link 1, then its body flit must be delivered at slot 1 to follow the head flit closely. Otherwise, if head flit of another flow occupies slot 1 of link 1, the configuration of the crossbar in node 1 is altered and thus the body flit of $v1$ cannot be switched correctly.

Source routing does not require a slot-table inside each router to store configurations in a distributive manner. However, the configuration information of each connection needs to be stored inside the network interface of the source node of each connection.

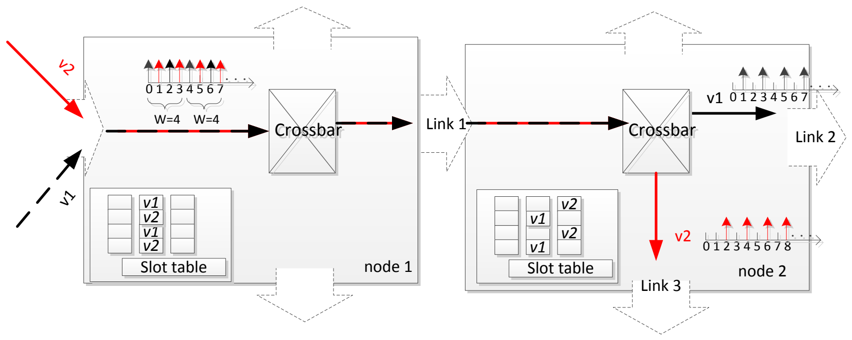


Figure 3.3: Illustration of the slot-table based configuration method

In slot-table based configuration method, the configuration information is distributed and stored inside each router, as suggested by Figure 3.3. For example, the slot-table inside node 2 denotes that slot 0 and slot 2 of link 3 are allocated to connection $v1$ and slot 1 and slot 3 of link 2 are allocated to connection $v2$. Inside each router, a slot counter reads information from the slot-table and updates the configuration of the crossbar at each slot. Slot counters of all the routers inside a NoC should update their values together according to a global time notion. With slot-table based configuration, there is no need to use head flits and thus does not require the allocated slots of a connection to be adjacent.

In the following, we will introduce some new techniques for TDM NoCs proposed by us.

3.2 Double time-wheel based dynamic connection setup

3.2.1 Problem description

Traditionally, dynamic distributed TDM channel allocation algorithms cannot be efficiently implemented in TDM based circuit switched NoCs because of the following reasons:

- During the connection setup process, messages such as setup requests, Ack/Nack need to be sent back to the source node to notify the failure or success of a setup request, and confirm/cancel the reserved channels. Traditionally, the delivery of these messages needs an auxiliary network [71, 53, 72], which adds up the costs and lowers the cost efficiency of a TDM NoC.

- The routing algorithms of the auxiliary network has to be deterministic [71, 53] to guarantee that setup, tear-down, and Ack/Nack messages of a connection are routed along the same path. In this way, the booked channels inside the routers along the path can be reserved/removed correctly. However, deterministic routing algorithms limit the exploration of path diversity. Setup requests routed by the deterministic routing algorithm are always traveled on fixed paths, resulting in low connection setup success probability.

- This auxiliary network is often a packet switched NoC [71, 53, 72]. It utilizes best effort packets to carry messages such as setup, release, Ack/Nack. These packets contend with each other, rendering the setup/tear down delay long and highly unpredictable.

Therefore, in Paper B, we propose a probe based connection setup method which can be used in TDM NoCs to eliminate the auxiliary network and overcome above-mentioned limitations. To support the probe based setup method, we developed a double time-wheel technique.

3.2.2 Probe based connection setup in TDM NoCs

Probe based setup method

Generally speaking, in order to support our probe based setup method, we add control signals *request* and $ANS(answer)$ to a connection. The *request* signal is used to denote the setup, data transfer and release requests for a connection. The ANS signal is used to carry "Ack/Nack" messages. These messages notify the source node whether a setup endeavour has failed or succeeded.

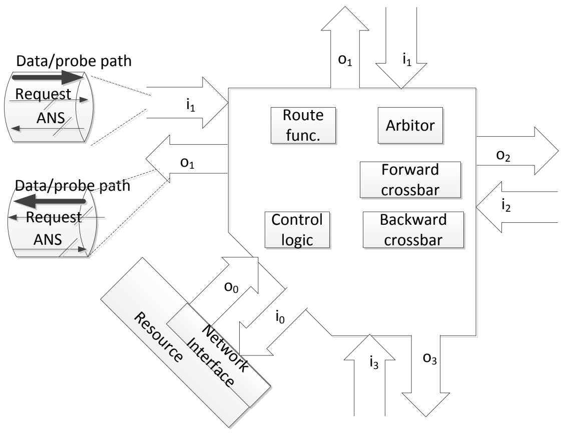


Figure 3.4: The overview of a router which supports the probe based connection setup method

The router used to support our probe-based setup method is illustrated in Figure 3.4. Each link of the router has a 2-bit *request* signal which is in parallel with the data path, and a 2-bit ANS signal goes in the opposite direction of the data path. Note that, all wires of a link, including the data path, the *request* signal, the ANS signal are all shared by a number of channels in a TDM manner.

The setup process has two phases, namely, channel reservation phase and acknowledge phase. During the setup process, setup requests are carried by probes. When a probe arrives at a router, if the next slot of a desired output link is free, the probe will reserve this slot and use this slot to continue its movement toward the destination. Therefore, when a probe arrives at the destination, a connection has been successfully built up and then the acknowledge phase begins. An "Ack" message will be sent back to the source node hop by hop, through the backward ANS signal of these reserved channels.

With our probe based setup method, the setup/release request and data flits of a connection can travel on the same TDM channels. This is possible because setup, transfer and release stages of a connection do not overlap with each other. This is due to 1) before a connection is established by a setup request, no data flits can appear on that connection, and 2) only after the transfer of data flits is finished, release request can then be sent out to release the slots of links reserved by the setup request.

Note that, if a probe fails to reserve a TDM channel in a router, a "Nack" message must be sent back by using the ANS wires of the reserved TDM channels. As the "Nack" messages travel backward, it can also release the reserved channels.

As a result, with the probe based setup method, the auxiliary network for setup is not needed any more. Since both forward and backward messages are transferred over the same TDM channels and along the same path, path search algorithm is no longer constrained to be deterministic.

However, backward "Ack/Nack" messages constitute a design challenge in our probe based setup method, since the "Ack/Nack" messages consist of only 1-2 bits, contain no address information, and need to be routed back following a given path. Moreover, the "ANS" wires used to deliver the "Ack/Nack" messages are also shared by different connections in a TDM manner, which mandates that the usage of correct slots to deliver the "Ack/Nack" messages of connections in order to avoid contentions. To resolve this challenge, we propose the double time-wheel technique.

Double time-wheel TDM technique

The way to route the backward messages (such as "Ack/Nack") of a connection is to rely on the slot table information inside each router along the path of the connection. As illustrated in Figure 3.5, the setup probe reserved time slots of a connection inside each router following the sequence $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$. The slot table of each router on the path records such information to configure its forward crossbar. Thus, if we can correctly read and interpret the information that reside in the reserved slot table cell, the backward crossbar can also be correctly configured to deliver backward messages. For example, if a backward message is sent out at slot 0 in router

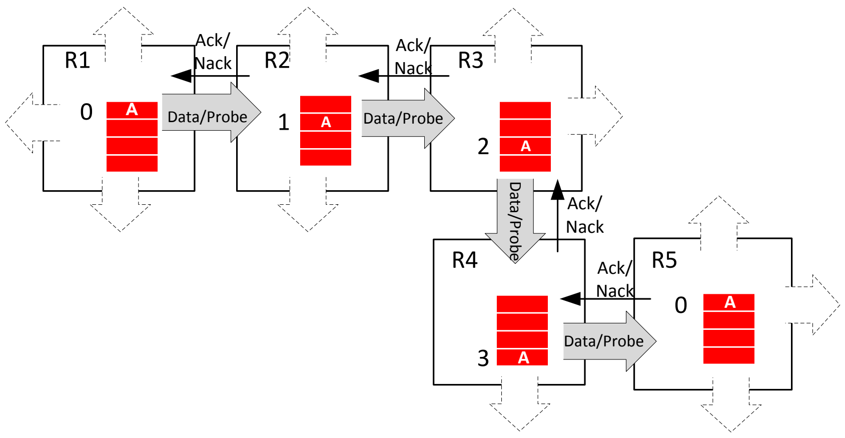


Figure 3.5: Route back Ack/Nack with the double orientation time-wheel technique

5, by reading the information from the slot table to configure the backward crossbar, it can reach router 4 at slot 3. Following the same manner, it can reach router 3 at slot 2, and so forth.

To support this scheme, we propose a double orientation time-wheel technique. With this technique, we apply two slot counters inside each router: one is incremental counter and the other is decremental counter. The incremental slot counter reads the slot table and configures the forward crossbar for data flits and requests. The decremental counter configures the backward crossbar for "Ack/Nack". With the two slot counters, there are two slot access sequences inside a TDM NoC. For example, as Figure 3.5 depicts, the read sequence of a slot-table with an incremental counter is slot $0, 1, 2, 3, 0, 1, 2, 3 \ldots$, while the sequence is slot $0, 3, 2, 1, 0, 3, 2, 1 \ldots$ with the decremental counter. Following this way, if the "Ack/Nack" message is sent out at the correct time slot, e.g. slot 0 from router 5, by using decremental slot counters to read the slot-table and configure the backward crossbar inside each router, it can reach router 4 at slot 3, reach router 3 at slot 2, and so forth. As a result, "Nack/Ack" messages are all delivered by slots of the ANS wires without contention and miss routing. The backward messages of a connection are naturally traveling on the same path that is reserved by the setup request.

An additional benefit of our double time-wheel technique is predictable delays of all kind of messages. Firstly, all the messages travel at a guaranteed speed of one slot/hop since both forward slots and backward slots along the path of a connection are consecutive, according to the requirements of a TDM NoC. Secondly, for "Ack/Nack" messages, although they need to be buffered at a node for a while to ensure that they are sent out at the correct time slot, such buffering delay is also predictable. The maximum buffering delay is the size of a time window. Thirdly, the delay of a single setup attempt is predictable since all kinds of message delays are predictable.

In general, our probe-based double time-wheel TDM NoC can have the following advantages.

- When probe based connection setup method is utilized in TDM NoCs, auxiliary network for connection setup is no longer needed. Instead, the setup/release messages and data flits of a connection share the same TDM channels for traversal.

- The double time-wheel technique can route Ack/Nack messages which are required in the connection setup phase. With this technique, the Ack/Nack messages of a connection are carried by only a few bits of wires and always attached to the path reserved by the setup request.

- By combining the probe-based method together with double time-wheel technique, we can freely choose routing algorithms for the setup probes to explore the path diversity. Besides, in our design, the delivery of all kinds of messages of a connection have predictable delays.

3.2.3 New features in a TDM router

The details about the TDM router supporting our double time-wheel technique have been described in Paper B. We designed a slot-table based router with the double time-wheel technique that supports the probe based connection setup method. In the following, we emphasize the new features added to a TDM router.

ANS signal management

Ack/Nack messages must be sent out at the correct time slot. As Figure 3.6 illustrates, we designed an ANS table at each input port to buffer Ack/Nack messages of different connections until their correct backward time slots

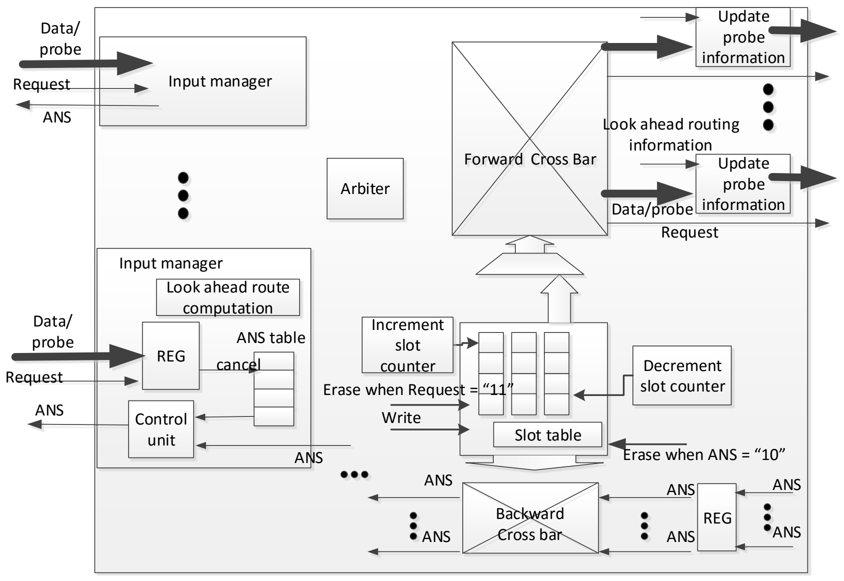


Figure 3.6: Router architecture of the double time-wheel TDM NoC

come. Each ANS table just has one column, with the row number representing different slots of a time window. Thus, if a TDM time window contains K slots, each ANS table should have K rows. In our design, for differentiating different kinds of backward messages, a table cell is 2 bits wide. The ANS signal management principle works as follows.

- When an "Ack/Nack" message needs to send back to the source node to notify the success/failure of a connection setup, it is firstly written into the corresponding ANS table cell pointed by the incremental slot counter. The Ack/Nack message is created in the same slot at which the connection setup probe arrives.

- The read position of the ANS table is pointed by the decremental slot counter. After a cell is read, the buffered message is sent back through ANS wires. Then, the cell will be erased at the beginning of the next cycle.

With this ANS signal management principle, the maximum buffering time of a message is K cycles, where K is the total slot number in the time window.

Slot table

Slot table plays a key role in our double time-wheel technique. Our double time-wheel technique requires that a slot table can be read by using both incremental slot sequence and decremental slot sequence simultaneously. It also requires that a slot table is able to handle written/erased requests on different table cells from both forward *request* signals and backward ANS signals at the same time.

As Figure 3.7 describes, rows in the slot table represent time slots, and columns denote output directions. During the slot reservation process, input port ID is written into a vacant table cell, whose row position is denoted by the increment slot counter and column position is denoted by the aiming output direction of the input probe. For example, the content of row 3 in Figure 3.7 means that at slot 3 the forward crossbar is configured in such a way: input port i_1 connects output port o_0, i_2 connects o_1. Meanwhile, it also denotes that for the backward crossbar, output o_0 connects input i_1, o_1 connects i_2. As Figure 3.7 suggests, when implemented in a mesh topology, the ID of each input port is encoded into 3 bits (suppose each router has 5 input/output ports). Thus, the slot table width equals 15 bits.

Both incremental slot counter and decrement slot counter simultaneously read a slot table, for the configurations of the forward crossbar and backward crossbar, respectively. To configure the multiplexers inside the forward crossbar, a row of table cells can be read to directly get a vector of configuration bits, with 3 bits per group. Each group configures a multiplexer. To configure the backward crossbar, the slot table's row content needs to be translated into another format, in which columns denote input ports and each table cell denotes the output port, as denoted by Figure 3.7. We implement a combinational logic circuit to do such transformation.

In our implementation, a slot table functions like a dual-ported RAM. Both incremental slot counter and decremental slot counter can issue the read addresses. Write access on a slot-table is only issued by forward "setup" probes. However, erase accesses can be issued by both forward "release" signal and backward "Nack" signal.

Therefore, we have to enforce rules for slot-table access, to avoid collisions between different access requests. Our rules are as follows.

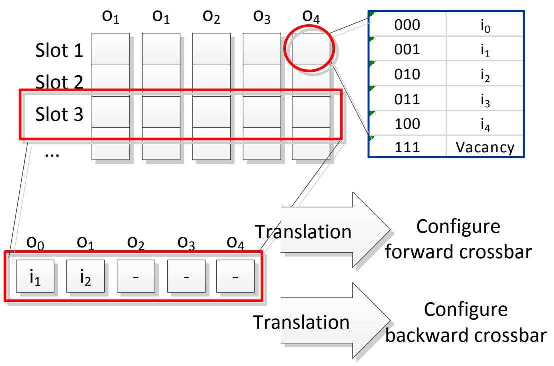


Figure 3.7: The slot-table in a router

1) Read access to a slot table returns a value within a delta delay (some combinational logic gates' delay), while write/erase access will be delayed until the very beginning of the next cycle (or the next slot). Thus, if read and write access to a cell happen at the same time, the read access gets the old value; if write is issued in the previous cycle and read happens in the current cycle, then read access gets the updated value.

2) We use a connection management mechanism to guarantee that write and erase accesses to the same table cell never happen. This mechanism states as follows.

1. For each connection, its forward "setup", "release", and backward "Nack" signals never collide. This is ensured by the non-overlapping phases in the connection setup and release. Therefore, write access and erase access to a table cell can never happen at the same time. Besides, this also guarantees that the erase access issued by forward "release" action of the connection never collides with the erase access issued by its backward "Nack" action.

2. Write accesses between different connections are also impossible to collide, since a slot table cell can only be exclusively reserved by one connection. Before the connection removes its reservation, no other connections can write or erase the table cell.

Flow control with double time-wheel technique

In TDM NoCs, the absence of contention ensures that no link-level flow control is required, but end-to-end flow control per connection is still required. For example, if the receiver's buffer is full, it needs to notify the sender to pause the data transfer. In previous works [25, 68, 8], TDM connections are bidirectional, one way from source to destination for data, and one way from destination to source to deliver end-to-end flow control messages.

However, bidirectional connections have several limitations:

- It imposes more constraints on the connection schedule/setup algorithms. Bidirectional connection setup requires a forward path as well as a backward path available. Thus, the connection setup algorithm suffers more limitations and thus has less probability to successfully build up such a connection.

- If a connection just has data flow on one direction, it is a waste of communication resources and bandwidth.

With our double time-wheel technique, connections can be unidirectional, the flow control messages can be carried by the backward ANS signal (1-2 bits) of a connection. Unidirectional connections can utilize the channel resources of a TDM NoC more flexibly and efficiently.

3.3 Highway in TDM NoC

3.3.1 Problem description

In TDM NoCs, quite often the utilizations of links are low. Because both idle and unallocated slots commonly exist in TDM NoCs.

Unallocated slots refers to the slots that are not reserved by any connection. The generation of unallocated slots is due to the two reasons:

- Successive links on the path of a connection require consecutive slots. Such mandatory sequence makes it difficult to utilize all the slots of links. For example, as illustrated in Figure 3.8, link 2 has one free slot, which is slot 0 (the TDM window size is 4 slots), and link 3 have two free slots, which are slot 0 and slot 2. Suppose that a connection $v4$ wants one slot on link 3 and one slot on link 2, although both link 3 and link 2 have free slots, they cannot be allocated to $v4$ since they are not consecutive.

- When mapping an application onto a TDM NoC, it is common that traffic flows are not uniformly distributed on all the links: some links have more bandwidth reservation requirements and some links less. Such unbalanced bandwidth requirements inside a network also cause slots of some links unallocated.

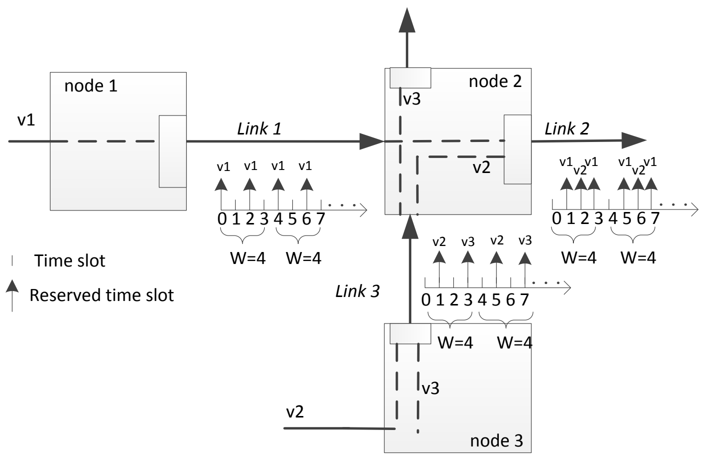


Figure 3.8: Illustration of slot utilization in TDM NoC

Idle slots refers to allocated but unused slots. Considering the practical traffic situations inside a network, it is unlikely that all the connections are busy all the time, especially for those statically scheduled connections. On one hand, idle TDM connections hold all the pre-reserved slots even if they have no data to deliver. On the other hand, busy TDM connections can just use a fixed number of reserved slots for data transfer, no matter how many data flits are buffering and waiting at the traffic sources.

However, is it necessary to enhance the performance of a TDM connection? As we know, TDM connections are used to offer guaranteed services. So, is it necessary to offer more service than guaranteed service?

In the streaming applications such as [73, 74] or H.264 decoding, each connection carries a streaming flow. In these applications, we might want such a property that a lower bound streaming quality for a flow is always guaranteed, whereas enhanced quality can be offered when the system has free resources. To support such a property, we only need the NoC's promise on the lower bound of communication bandwidth to offer guaranteed quality, whereas the upper bound of a traffic flow should be adjustable and adaptive to the traffic situations.

Besides, consider a computer system with a guaranteed connection between an L1 cache and L2 cache. Since cache misses are not completely predictable, often connection bandwidth is over allocated [68]. Thus, we might want a mechanism that can keep the guarantee on minimum throughput and on the predictable traffic flow, while offering additional non-guaranteed bandwidth to enhance the overall system performance by absorbing peaks of less predictable traffic flows.

Therefore, in some situations it would be a merit if a system can not only provide guarantee service, but also offer more and better service than the guaranteed whenever possible. From this perspective, we propose a novel highway technique. When applying this technique, a TDM connection can dynamically acquire both idle and unallocated time slots to enhance its throughput while still keeping the guarantee on lower bound of the bandwidth.

3.3.2 Highway technique for TDM NoCs

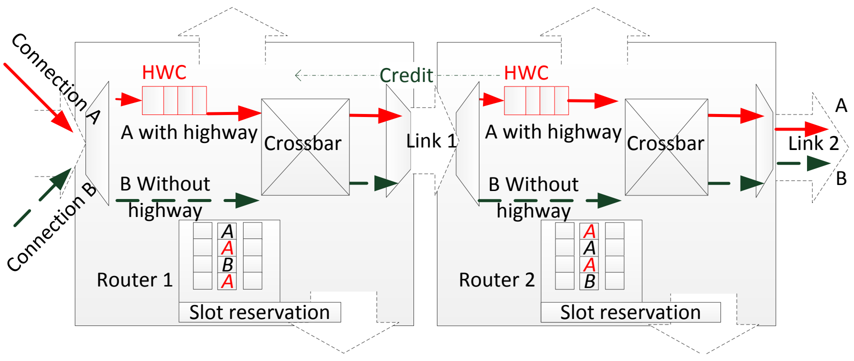


Figure 3.9: Illustration of the function of a highway

Our proposed highway can accelerate the data transfer of an established TDM connection by using unused time slots along the links of the connection.

A highway consists of one buffer queue per router, and along the given path of a TDM connection. We name the buffer queue as Highway Channel (HWC). With these HWCs, a connection can use both unallocated and idle slots of the output links on its path.

The function of a highway is illustrated in Figure 3.9. Both connections, A and B, span two routers by using slots of link 1 and link 2 in a consecutive manner. Connection A books slot 0 of link 1 and slot 1 of link 2; connection B books slot 2 of link 1 and slot 3 of link 2. Connection A also builds up a highway path by using one HWC in router 1 and one HWC in router 2, respectively. Thus, besides slot 0, connection A may also use slot 1 and 3, even slot 2 (if connection B has no data) of link 1. In contrast, without HWCs, connection B can only use slot 2 for data delivery.

In contrast to the free slot utilization methods proposed in [25, 66, 72], our highway technique can guarantee in-order data delivery. In [25, 66, 72], free slots are used to deliver best effort packets. However, the transfer delay of best effort packets are highly unpredictable. As a result, if some packets of a flow are delivered in the form of the best effort packets while the others are delivered by using TDM connections, the arrival order of packets at the destination node may be different from the sending order. In comparison, our highway technique can maintain the packet order of a data flow.

3.3.3 Router architecture overview

A router supporting the proposed highway technique is depicted in Figure 3.10. The input manager at each input link manages all the incoming flits of that input. Each input manager contains one or more HWCs. To support dynamic usage of the highway technique, each input/output link has a flag signal and a credit signal associated with the data path. The flag signal goes in parallel with the data path, while the credit signal is in the opposite direction. The 3-bit wide flag signal is used in highway setup, data transfer, and release. The 2-bit credit signal is used for Ack/Nack purpose in the highway setup phase and for credit-based flow control when in data transfer phase.

3.3.4 Highway setup process

As depicted in Figure 3.11, the procedure of setting up a highway for a 2-hop TDM connection which has 2 reserved slots in a time window of 4 is used as

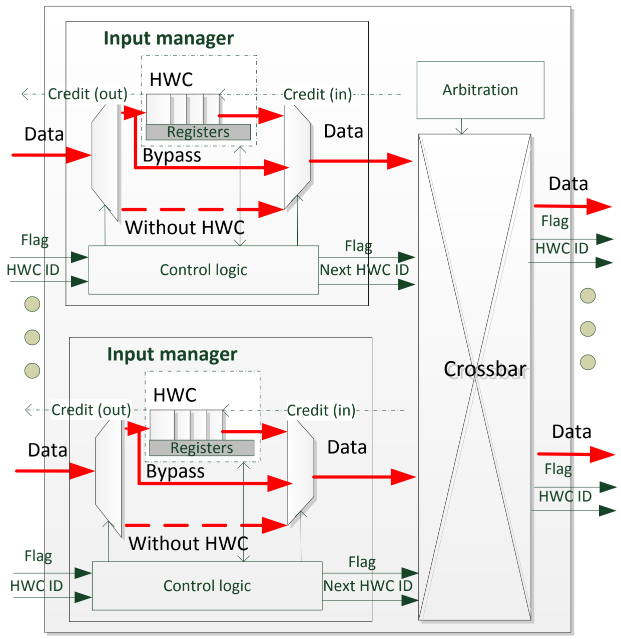


Figure 3.10: The overview of a router with highway support

an example to illustrate the dynamic highway setup process proposed by us. When a reserved slot (slot 0) is coming, the source node of an established TDM connection sends out a data flit carrying the *flag* signal "setup". This flit travels on the reserved slots of links, at a guaranteed speed of one slot per hop. If it can acquire an HWC in a router, its flag signal remains "setup". If the destination receives a flit with flag signal as "setup", "Ack" message will be sent back to the source node hop by hop, along the *credit* signal of each reserved HWC. When the source node receives "Ack", data transfer can use the established highway.

If the flit with flag "setup" fails to reserve a HWC inside a router, the setup process stops. The flit continues its traversal with the flag turning into "No HWC" (The default flag for a flit without highway is "No HWC"). At the same time, a "Nack" signal will be sent back to the source node hop by hop, along the credit signals of the already reserved HWCs along the path. As the "Nack" signal travels, it will release the reserved HWCs.

The proposed dynamic highway method has the following features:

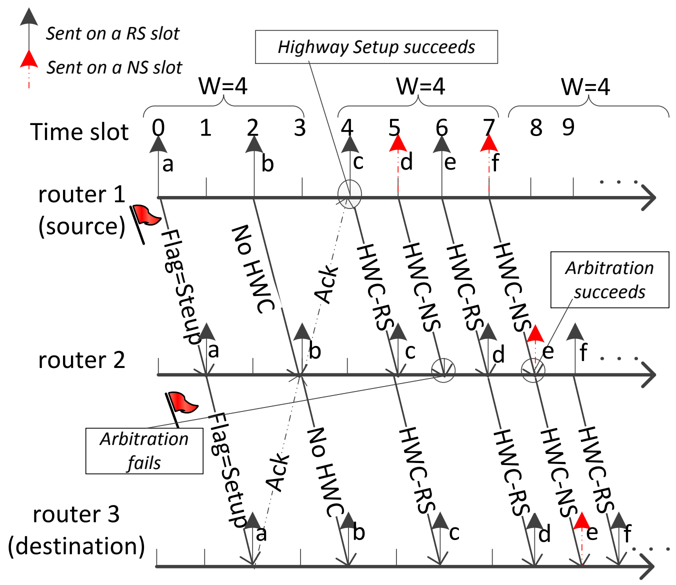


Figure 3.11: Highway setup process

- Highways can be dynamically established and released in a distributive way. When a connection needs to accelerate its data transfer, the source node of the connection can send out a "setup" flag to build up a highway by reserving HWCs in routers along the path of the connection. When a highway is no longer needed, the source node can release it by asserting a "release" flag.

- The highway setup process is guaranteed to be contention free. This is due to the fact that the "setup" flag is delivered by using pre-reserved TDM slots of links. As a result, this contention-free feature simplifies the design of HWC allocation logic since there is no need to do arbitration between setup requests.

- Our highway setup technique is applicable to many kinds of TDM NoCs. It basically has no architectural dependency. We impose no constraints on how connections are established and how crossbars are configured. Thus, this technique can be applied for both statically and dynamically built-up TDM connections. As long as a TDM connection delivers a flit to the destination, and the *flag* of the flit remains as "setup", a highway has been established.

- The highway setup process has no influence on the normal TDM data transfer, since the setup requests are carried by flag signals, and do not affect the data path.

- The setup time of a highway is predictable. This is due to the following reasons. Firstly, a "setup" flag travels at a guaranteed speed of one slot per hop toward the destination, since it is delivered on an established TDM connection with consecutive slots. Secondly, the backward "Ack/Nack" signal also travels at a constant speed per hop, since it is delivered by using the "credit" wires of the reserved HWCs and all these HWCs are allocated exclusive to a connection. Assuming each slot is one cycle and the propagating speed of "Ack/Nack" signal is also one cycle per hop and D is the distance between source and destination, then the total delay of a setup attempt is at most 2D + 2 cycles. The 2D cycles are the time spent on the traversal of the forward "setup" flag and backward "ACK" signal. The additional 2 cycles are regarded as the overhead time spent at network interfaces.

3.3.5 Data transfer with highway

After a highway is built up, accelerated data transfer can be launched. As described in Figure 3.12, when a data flit arrives at a router, the first step is to judge whether the incoming flit has a booked HWC or not. If the flit has no reserved HWC, the incoming flit must be delivered by using pre-reserved slots. Due to the consecutive property of the reserved slots, the incoming flit should be directly forwarded to the downstream node at the next slot.

However, as Figure 3.12 suggests, if the flit has an HWC, then the flit will be dispatched to that HWC first. When entering into an HWC, if the First-In-First-Out (FIFO) buffer queue is non-empty, the flit is pushed into the FIFO. However, if the FIFO is empty and if its requested output link is granted by arbitration, the incoming flit can be directly forwarded by using the *bypass way*, without being buffered into the FIFO. Otherwise, it still needs to be buffered. The flit arbitration rule is described in the next section.

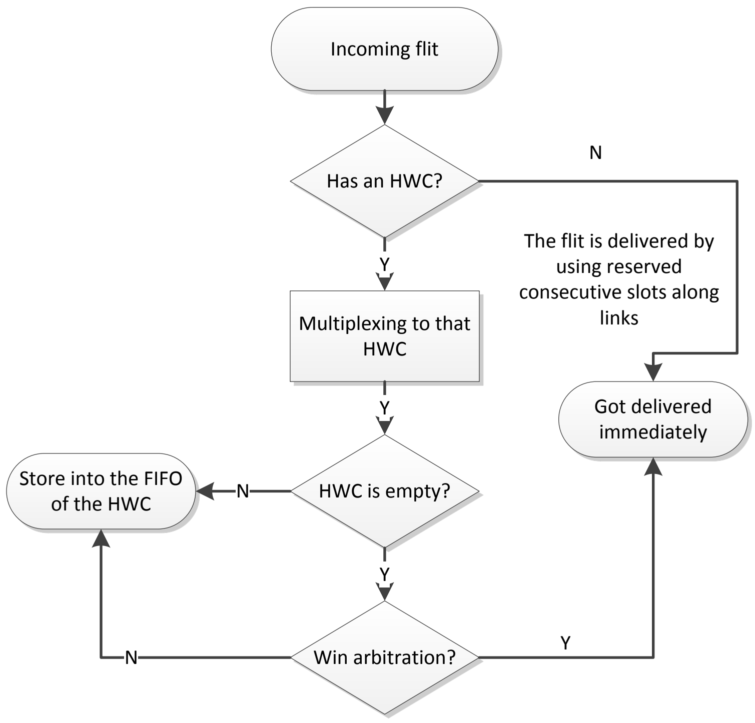

Incoming flit
Has an HWC?
N
Y
The flit is delivered by using reserved consecutive slots along links
Multiplexing to that HWC
Y
Got delivered immediately
Store into the FIFO of the HWC
N
HWC is empty?
Y
Win arbitration?
N
Y

Figure 3.12: Flit storing and forwarding rule

Flit arbitration rule

During the data transfer phase, when an output slot is requested by more than one HWC, arbitration is needed. As suggested by Figure 3.10, each input link of a router has an input manager, and each input manager can have one or several HWCs. Thus, there are two stages of arbitration accordingly: a first stage arbitration between all the HWCs of an input manager, and a second stage arbitration between the input managers, as described in Figure 3.13.

Our arbitration mechanism needs to prioritize flits of a connection which are delivered on the pre-reserved slots of the connection. Only in this way, the lower bound of the bandwidth of a connection can be guaranteed. We use priority based round-robin to implement this scheme.

However, unlike the standard priority based round-robin arbiter, our arbiter design can take advantage of the following properties to simplify its

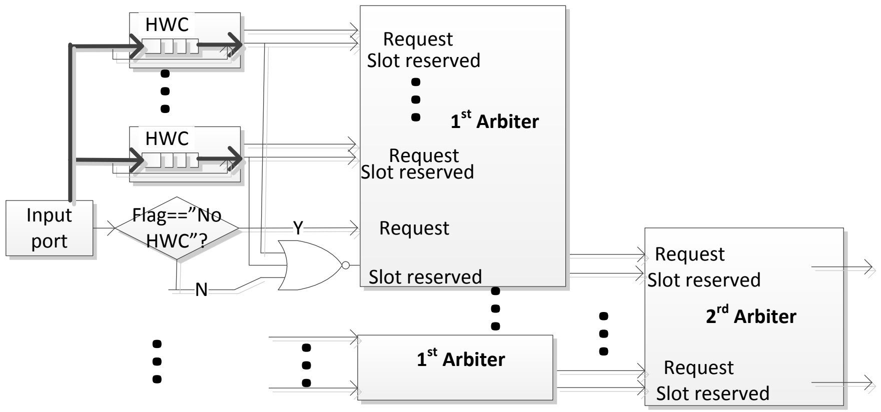


Figure 3.13: The structure of the two-stage arbitrator

circuit logic.

- For the first stage arbitration at each input, at one time slot there is at most one request claiming that it has reserved the current slot.
- For the second stage arbitration for each output link, at one time there is also at most one request claiming that the current slot is reserved by it.

Due to the above two properties, we can simplify the priority based arbiter design for both stages. Our design just adds 2-3 gate-level delay to a classic no-priority round-robin arbiter.

The implementation of the two-stage arbitration is abstracted in Figure 3.13. An HWC asserts its *request* signal when it has flits to deliver. Meanwhile, if the HWC meets one of its reserved slots, it will also assert the *reserved slot* signal. Besides, if the input flit has a flag "No HWC" and no other HWC asserts its *reserved slot* signal[1], the input port can assert the *reserved slot* signal. The asserted *reserved slot* signal of the first stage will be propagated to the second stage.

Both stages of arbitration prioritize the request with an asserted *reserved slot* signal. If no one asserts the reserved slot signal, it will use round-robin rule to decide a winner from all the requesters.

[1]This actually checks whether the incoming flit has an unreleased HWC or not. Refer to Paper C for more details

3.4 Future work

Both our double time-wheel technique and highway techniques are helpful to TDM NoCs. The double time-wheel technique enables two-way communication on a TDM connection with a low cost. The highway techniques can enhance the performance of a TDM NoC without disordering data flows or introducing additional communication overhead. In the future, we plan to explore the two techniques in the following directions:

- Apply the double time-wheel technique for advanced control purpose on the connection setup process. For example, the backward ANS signals can be used to feed back the congestion information about the network to source nodes. Based on the congestion information, a source node can make better decisions on when to send out the new connection setup requests, and how often to retry the failed ones. In this way, we can try to lower down the contention probability between setup requests and enhance the success probability of each connection setup endeavor.

- Apply the double time-wheel technique for error correction purpose. When soft errors [75, 76] occur in a data transfer process and are detected by a destination node, the backward ANS signal path can be used to inform the source node about the error. Then, the source node can re-send the data to correct the errors.

- Design a network interface that can offer better support to the TDM highway technique. As a complete TDM NoC solution, network interface [77] is always indispensable. In order to support the TDM highway technique, the network interface also needs to add in a few new functionalities. For example, there should be a decision-making unit at the network interface to decide when to setup a highway for acceleration and when to cancel it.

Chapter 4

Spatial Division Multiplexing NoC

This chapter summarizes our research on Spatial Division Multiplexing (SDM) based circuit switched NoC. Particularly, we proposed a new allocator with maximal matching quality and strong fairness guarantee [Paper D]. It can be used for allocating channel resources in SDM NoCs. We also proposed a circuit switched NoC which supports multiple channels (SDM) and multiple networks. Based on this NoC, we explored the effects of several channel partitioning and configuration polices [Paper E]. Finally, we analyzed the respective strengths and weaknesses of circuit switched NoC and packet switched NoC [Paper F].

4.1 Introduction

4.1.1 An overview on SDM NoC

As the feature size in semiconductor technology scales down, more wires can be utilized between on-chip routers. Using all the wire resources between two routers as one wide communication channel becomes awkward and inadequate in many situations. Therefore, we may need to think about breaking down the wire resources into several narrower links, which is called *sub-links*, to offer more flexibility. One popular way of organizing these sub-links is called spatial division multiplexing (SDM) [78, 79, 80, 81, 82], which interconnects all the sub-links of a router with one crossbar.

The comparison between TDM NoC and SDM NoC is illustrated in Figure 4.1. With TDM technique, a single link is shared by multiple data flows

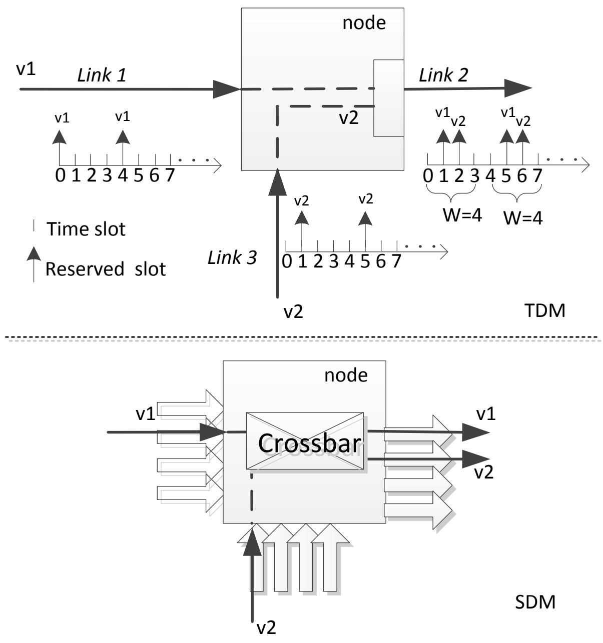


Figure 4.1: TDM NoC and SDM NoC

by dividing the time of the link into many fractions. Each data flow only uses a fraction of the time of the link in an alternating pattern. In comparison, SDM physically divides a link into sub-links. Each data flow can occupy one or several sub-links. All the flows are physically and spatially isolated.

4.1.2 Properties of SDM NoCs

The advantage of SDM NoC is that established connections are physically isolated. In other words, after a connection is established, the sub-links reserved by the connection are exclusively utilized. Thus, advanced end-to-end communication techniques such as source synchronized data transfer technique [83, 84, 85, 6, 86] can be utilized in SDM based circuit switched NoCs. With such techniques, data transfer can suffer less clock uncertainties and thus benefit from a higher clock frequency[1].

[1]With source synchronous data transfer technique, the sender sends a clock together with the data. Since the clock and data travel along the same path, they experience the

The disadvantage of SDM NoCs is that, as the number of sub-links increases, the area spent on the crossbar of a router increases dramatically. Moreover, if dynamic connection setup method is applied, each router needs an allocator for sub-link allocation. However, both the area and the critical timing path of the allocator increases significantly, when the number of sub-links grows up.

4.2 An allocator for channel allocation

In this section, we present a new allocator designed by us. This allocator can make an allocation decision within one clock cycle. It also guarantees maximal matching quality and strong fairness.

4.2.1 Background and problem description

On-chip communication networks often use allocators for resource allocation purpose. For example, if we want to dynamically set up connections in a distributive way, we have to allocate output channels in each router individually according to the setup requests [87]. Besides, in virtual channel based packet switched NoCs, an allocator is also needed inside each router to allocate virtual channels for packets during run time [3].

We illustrate the allocation problem in Figure 4.2. Suppose two channel setup requests arrive at a router simultaneously and each of them demands an output channel from East output direction. If we define the setup requests as requesters and the output channels of as resources, an allocation problem is about how to allocate resources to their requesters.

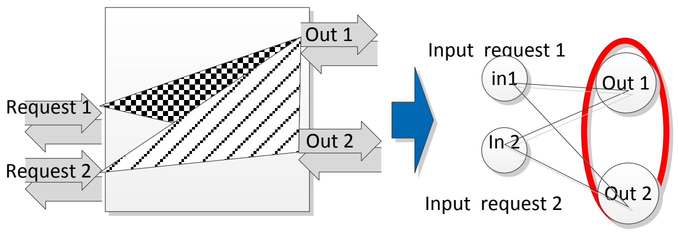


Figure 4.2: Illustration of a channel allocation problem in NoC

similar delay and jitter. At the receiver side, data is re-sampled with the incoming clock.

Allocator inside a system is responsible for solving such allocation problems. It performs a matching between resources and the requesters. A matching is a distribution of resources to requesters satisfying the following three rules [88]:

- Only if the corresponding request exists, a resource can be granted to the requester.
- Each resource is at most granted to one requester.
- Each requester is at most granted once.

Two criteria [89, 88] are often used to evaluate an allocator. One is *matching quality*, and the other is *fairness*.

Matching quality refers to how well resources can match the needs of the requesters. It can be classified into *Maximum matching* and *Maximal matching* (refer to Paper D and [88] for detailed definitions). Maximum matching is often too costly to be realized in hardware. However, maximal matching is achievable. Maximal matching refers to that the resources are distributed in such a way that no additional requests can be served without removing existing grants.

Fairness [89] can be classified into strong fairness and weak fairness [90, 91]. Intuitively, fairness is about how requesters are served in proportion to their relative request rates. In practice, strong fairness often means that persistently active requesters are served in a periodic sequence equally often, while weak fairness only guarantees that each active requester is eventually granted, without any guarantee on the service behavior and the service rate.

Allocators designed for NoC confront many constraints. Firstly, since they are hardware based, the allocation algorithms have to compromise for hardware cost. Secondly, the allocator is desirable to work out an allocation decision within one clock cycle [89, 3]. This is due to the performance of a NoC is very sensitive to the delay in each router.

Because of these constraints, allocators used in NoC often suffer from drawbacks in either matching quality or fairness. For example, on one hand, the separable-input-first (SIF) and separable-output-first (SOF) [3, 92, 89] allocators utilize two-stage round-robin arbitrations to ensure strong fairness. However, they cannot guarantee the maximal matching quality. On the other hand, the wave-front (WVF) [93, 94, 95, 92], rectilinear-propagation-arbiter

(RPA) and diagonal-propagation-arbiter (DPA) [96] allocators provide maximal matching quality. However, they have no or only weak fairness guarantees.

Although other kinds of allocators, such as SPAA [97], iSlip [98], D2DDR [99], provide both maximal matching quality and strong fairness, they take too many clock cycles to work out an allocation, since these allocation algorithms need several iterations to optimize their allocation decisions. As a result, it is infeasible to use them in NoC environment.

For general matching problems, no existing solution is known to meet the constraints on timing while overcoming the shortcomings on matching quality and fairness.

4.2.2 A fair and maximal allocator for HRA

By taking a close look at the allocation problems in NoCs, frequently we encounter a special kind of allocation problems, which is named as *homogeneous resource allocation* (HRA) problem. For this kind of problems, we propose a single-cycle allocator to guarantee both maximal matching and strong fairness.

HRA problems in NoCs

The term *homogeneous resources* in our definition refers to a class of resources that have the same functionality and can be used interchangeably. *homogeneous resource allocation* problem obeys two more rules besides the three rules aforementioned in section 4.2.1:

- For each requester, all desired resources should belong to the same class;
- Any resource in a class can be granted to the requester which asserts requests on this class.

As Figure 4.3 suggests, HRA problems commonly exist in the systems which can be modeled by a multi-queue and multi-server model. In such a system, the allocation of servers to serve input queues constitutes an HRA problem if all servers are identical and can be used interchangeably. In this model, the matching quality of the allocator describes how well idle servers can discover and serve non-empty queues efficiently and without conflicts.

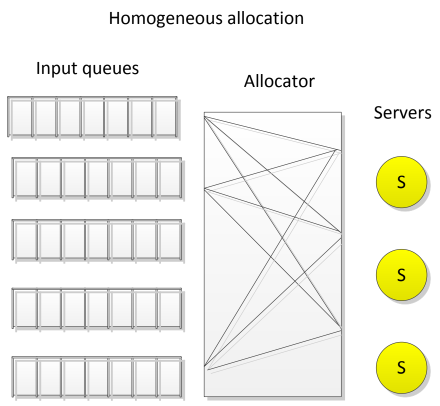


Figure 4.3: Illustration for homogeneous resource allocation problem

The fairness of the allocator affects the service sequence and service frequency on input queues. This example suggests that the HRA problem revealed by us exists in different kinds of NoCs as well as in other on-chip applications, as long as they can obey the multi-queue and multi-server model. In NoC design practices, HRA may exist in circuited switched NoC, for example, SDM based channel allocation; or in packet switched NoC, such as virtual channel allocation.

We can use a matrix with rows representing requesters and columns representing resources to express a homogeneous resource allocation problem. For example, as illustrated in Figure 4.4, input channels 0, 1, 3 (marked as ch0, ch1, ch3 in the figure) each requests a channel from output direction 1. Input channel 2 (ch2) requests a channel from output direction 2. It is an HRA problem since 1) each input channel (requester) just has one desirable output direction (resource class), and 2) any channel (resource) of a requested output direction can satisfy the need of a requester.

An HRA problem has the properties that 1) the request matrix can be split into sub-matrices based on resource classes, and 2) each of the sub-matrices can be merged into a single column, as described in Figure 4.4. As a result, we can assign an allocator to each reduced sub-matrix to solve the allocation individually. In such a way, the complexity of the allocator design can be reduced.

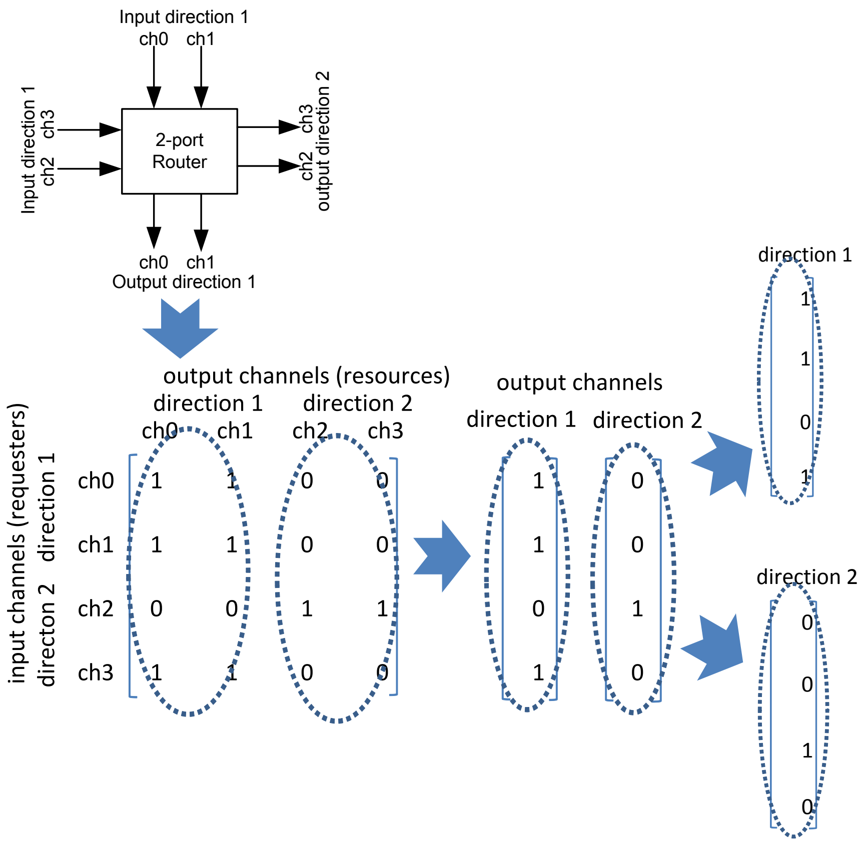


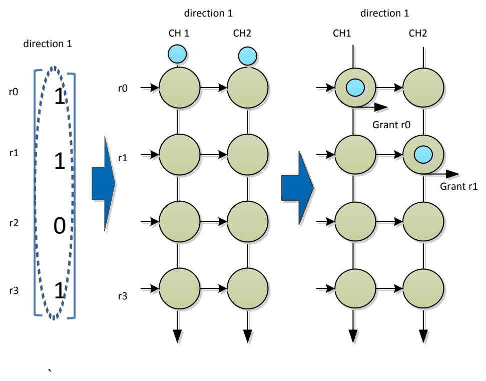
Figure 4.4: Reduction of the request matrix

4.2.3 Single-cycle fair and maximal allocator

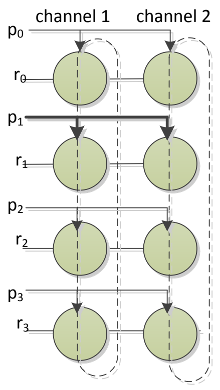
We proposed a single cycle fair and maximal allocator to solve each reduced sub-matrix. The allocator used to solve each reduced matrix contains two parts: the resource allocation logic part and the priority updating logic part.

Overview of the resource allocation logic

The resource allocation logic makes resource allocation decisions. As depicted in Figure 4.5a, a matrix-based structure is applied for making resource allocation decisions. We name it as "waterfall" (WTF) because it finds the matching with the help of consecutive rows of arbitration cells. For an n-requester allocator, it requires n rows.



(a) The water-fall allocation concept



(b) Rotation of the start row

Figure 4.5: Overview of the resource allocation logic

We use an example to illustrate how an allocation decision is made by using the WTF allocation logic. Suppose the allocation logic is used to solve the reduced request matrix at the output direction 1 in Figure 4.4. Since it has 2 resources and 4 requesters, the allocator is also composed of 2 columns and 4 rows of arbitration cells. 3 of the 4 requesters are active, marked by an "1" of the reduced matrix. The two small dots in Figure 4.5a denote the availability and grant decision of the two channels of output direction 1, respectively. In Figure 4.5a, the allocation starts from the top to the bottom, so that requester r_0 get channel 1, then requester r_1 get channel 2.

However, in order to guarantee the fairness, we have to roll the allocation start and the end row at each round. Thus, the structure of the allocation logic is modified into Figure 4.5b. The allocation start row i can be selected by assert signal p_i. If row i is selected as the start row, due to the looped structure in Figure 4.5b, the end row becomes $(i + n - 1) \bmod n$, where n is the total number of rows.

From Figure 4.5b, it seems that the allocation logic needs a looped structure to roll the start row and end row. However, we proposed a technique to make it loop-free, as illustrated in the left part of Figure 4.6. For an n-row allocator, the loop-free structure needs $2n - 1$ rows. The general idea is that using the bottom $n-1$ rows replicates the top $n-1$ rows, from row 0 to row $n-2$. In this way, rolling of the start and end row is equivalent to selecting an active area. For example, if an allocation starts in a looped structure starts from row 1 and ends at row 0, it is equivalent to activate the area in the loop-free structure from row 1 to row 5, since row 5 replicates row 0, as illustrated in the left part of Figure 4.6. The right part of Figure 4.6 depicts the detailed circuits of the loop-free structure. It is implemented entirely by combinational logic gates, which means that the entire allocation behavior takes no more than one clock cycle.

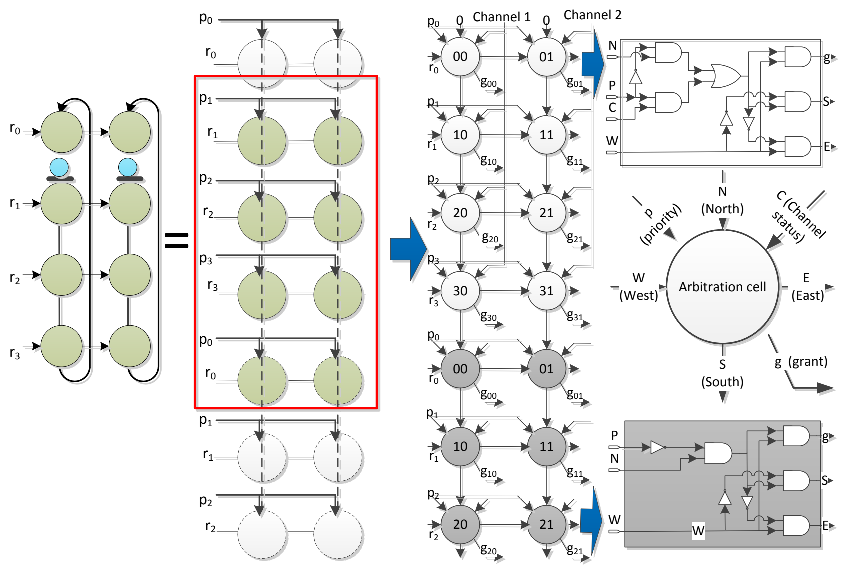


Figure 4.6: The loop-free structure of the WTF allocator

Massive round-robin fairness policy

To guarantee the strong fairness, we proposed a Massive Round-Robin (MRR) policy. Briefly speaking, our MRR policy states that the last granted requester of the current round will become the end row (has the lowest priority) in the next round. The start row is acquired by incrementing the end row by 1. If no request is granted in the current round, the start row and the end row remain unchanged in the next round.

The traditional round-robin policy [100, 101, 102] operates on the principle that a request that was granted in the current round should have the lowest priority in the next round. However, such a fairness policy only works in the situation that no more than one requester granted in a round. In comparison, our MRR policy extends the round-robin policy and makes it suitable in the situation that multiple requesters can be granted in a round.

Suppose there are m available resources and n requesters (numbered from 0 to n-1). Given current grants g_i, $0 \leq i \leq n-1$ for each requester ($g_i = 1$ means granted), and the start row of round t is row k. Then the detailed algorithm to find the start row $k\prime$ of round t+1 is depicted in Algorithm 1.

Algorithm 1 Priority updating of MRR policy in $m \rightarrow n$ allocation

```
Require:
    Exist unique $k$, that $k \in \{p_k = 1, k \in [0, n)\}$ in round $t$

Ensure:
    Find unique $k\prime, k \in [0, n)$, that $p_{k\prime} = 1$ in round $t+1$
    for $b = n-1; b \geq 0; b = b-1$ do
        if $g_{(b+k \bmod n)} = 1$ then
            $k\prime = (b+k) \bmod n$
            return $(k\prime + 1) \bmod n$
    return $k\prime = k$
```

We have implemented the MRR policy on hardware with a loop-free architecture to control the selection of the start row and the end row of each allocation round.

For more detailed description of the allocator's architecture, the proof and evaluation of the fairness mechanism and the performance, please refer to our Paper D.

4.3 Sub-networks and sub-channels

4.3.1 Problem description

We need to think about three questions when splitting a link into multiple physical sub-links. (For convenience, in the following discussions, we use the term *sub-link* and *channel* interchangeably.)

1. What would be an optimal number of sub-links: as many as possible, or there are certain limitations?

2. What is the optimal way of organizing the physical sub-links (channels) of a router? One possible way is to use a crossbar to interconnect all the channels. This method is called spatial division multiplexing (SDM). Another possible way is assigning the channels to different sub-networks.

3. How to dynamically set up a connection which needs multiple sub-links (channels) to satisfy its bandwidth requirement.

4.3.2 Circuit switched NoC with multiple channels and multiple networks

Overview

We propose a multi-channel and multi-network circuit switched NoC (MultiCS). It combines spatial division multiplexing [78, 79], which we call sub-channels, with sub-networks, as described in Figure 4.7. Sub-channels divide the wires between two nodes that then can be allocated separately and independently. Sub-networks are independent parallel physical networks that connect to the same nodes of a network [103, 104]. A connection between two nodes can utilize one or several sub-channels, and span one or several sub-networks.

In Figure 4.7, we use switch block diagrams [105] to reveal the differences between sub-channel organization and sub-network organization. With sub-channel organization, data from an input channel of a router can be switched to any channel of the desired output direction. However, with the sub-network organization, data from an input channel can only be switched to the output channels of the sub-network that the input channel belongs to. We can observe that organizing the same number of channels into sub-channels offers better switching flexibility [105] than organizing them into sub-networks, at the cost of a more complex crossbar and channel allocator.

Generally speaking, with the sub-channel organization, the critical timing path increases as the number of channels grows up. More specifically, the critical timing path for the control path scales up with $O(n)$, where n is the number of channels. The critical timing path of the data path scales up with

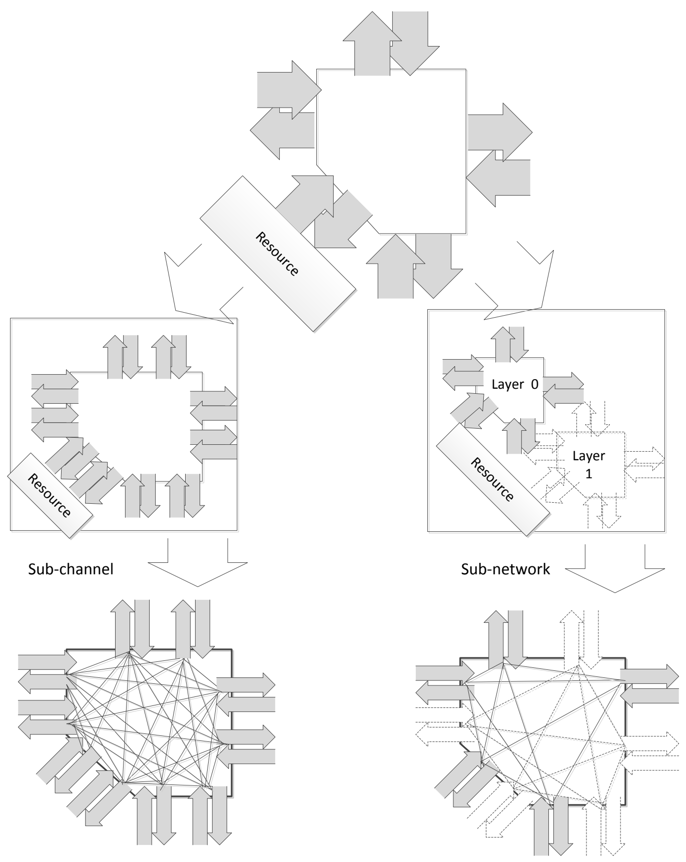


Figure 4.7: Channel organization methods

$O(log(n))$. Besides, the area of both control path and data path scales with $O(n^2)$, due to the scale up of the allocator [106] and the crossbar [82, 78].

With sub-network organization, the critical timing path is irrelevant with the number of channels, while the area increases linearly. Thus, compared with the sub-channel organization, although the sub-network organization is inferior in switching flexibility, it is superior in clock frequency and area cost.

Dynamic connection Setup

We introduce the parallel probe based method Paper A for connection setup in MultiCS. As a result, the dynamic connection setup no longer relies on an auxiliary packet switched NoC which was commonly used in [81, 87, 46, 72, 80].

However, the parallel probing algorithm proposed in Paper A utilizes priority based arbitration method to do channel allocation. This allocation method is not suitable for a router with multiple channels per direction, since the circuit logic used for priority comparison becomes slow and cumbersome as the number of channels increases. Thus, we utilize the allocator introduced in Paper D for channel allocation purpose. Besides, for live-lock avoidance purpose, since the priority preemption policy in Paper A can no longer be used, we adopt a method which randomizes the retry interval for failed probes as a replacement.

Set up connections with multiple channels

We proposed two schemes for setting up connections that have a width requirement of more than one channel width. For example, suppose a connection has the width requirement of 4 bytes, while the width of each channel is 2 bytes, then the connection should consist of 2 one-channel connections.

The first scheme is called Deterministic Channel Allocation (DCA). DCA imposes a mandatory requirement on the connection width. For example, if a connection has a width requirement of 4 bytes and the width of each channel is 2 bytes, then the connections must consist of 2 one-channel connections; if the width of each channel is 1 byte, the it should be composed of 4 one-channel connections; any allocation below or above this figure is unacceptable.

The second scheme is called Adaptive Channel Allocation (ACA): ACA scheme has no hard connection width requirement. During the setup phase, a setup request tries to build as many one-connections as possible from the source to the destination as possible. However, the final connection width is determined by the number of successfully established one-channel connections, which depends on the run-time congestion situation of a network.

4.4 Circuit switching versus packet switching

4.4.1 Problem description

Traditionally, circuit switched NoC is considered as a method for offering Quality of Service (QoS) guarantee for communication, e.g. guaranteed throughput or guaranteed delay. However, circuit switched NoC can also achieve high communication performance. Circuit switched NoC has, when compared to packet switched NoC, a longer setup time, but lower HW complexity and higher clock frequency [107, 108, 109, 110]. Thus, depending on packet size and throughput requirements, it may exhibit better or worse average performances.

4.4.2 Analysis and comparison of packet switched NoC and circuit switched NoC

Intuitions on packet switched NoC

Large packets are detrimental to packet switched NoC. It will incur unbalanced usage of links. As illustrated by Figure 4.8, intuitively, when each packet just contains one flit, the span of idle periods are just made up of a few cycles and equally distributed between output flow A and flow B. However, when packet size increases, the average span of idle periods grows wider and wider. Within a short period and with large packets, we may observe that output flow A is quite busy, while output flow B is almost idle. Such unbalanced usage of links is undesirable, since it is not good for the exploitation of the bandwidths of a network.

If the packet size is small, we can deal with the burstiness by increasing the buffer depth of a Virtual Channel (VC), and re-balance the traffic flows by adding more VCs [111]. With the same injection rate, large packets are detrimental to packet switched NoC since they increase the needs on both sides. The longer the packets, the more VCs and buffers are required to compensate the performance loss. Unfortunately, both VC and buffer are expensive, especially that adding virtual channels can lower the clock frequency. Thus, we can imagine that, if packets are long enough, they are almost impossible to be well handled by packet switched NoC with acceptable cost.

Intuitions to circuited switched NoC

On the other hand, circuit switched NoC prefers large packets. As we know, in order to deliver data with a circuit switched NoC, we need to have two phases.

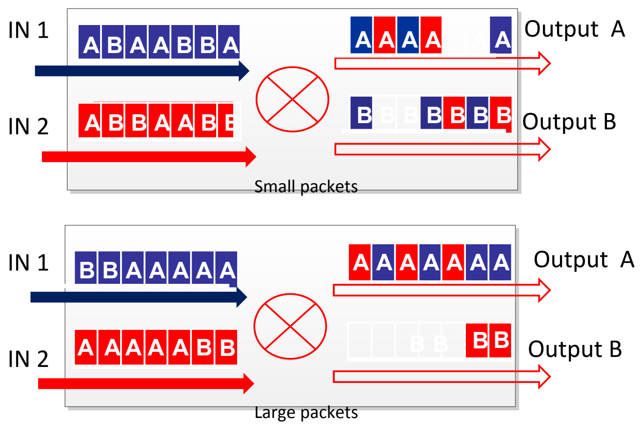


Figure 4.8: Unbalanced traffic caused by large packets in packeted switched NoC using round-robin arbitration

- Connection setup phase. A connection setup attempt is not guaranteed to succeed. To successfully set up a connection, it needs to retry failed setup attempts.

- Data transfer phase. Data transfer can be launched only when the connection has been established.

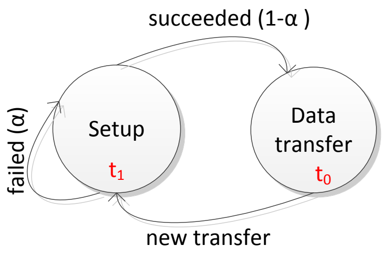


Figure 4.9: Markov model for circuit switched NoC

We can use a Markov model to describe the connection setup and data transfer phases of a circuit switched NoC, as shown in Figure 4.9. Suppose for each setup attempt, the failure probability is α, and the average time spent on each setup attempt is t_1. Suppose further that the average data transfer takes time t_0. Then, we have normalized throughput as

$$TH = \frac{t_0}{t_0 + \frac{t_1}{1-\alpha}}$$

As the packet size grows, data transfer time t_0 increases accordingly. Therefore, we can observe that the normalized throughput of circuit switched NoC goes up as the packet size increases.

Thus, we may conjecture that, as the packet size increases, the performance curve of circuit switched NoC and performance curve of packet switched NoC may have a cross point, since one rises and the other falls.

Architectural analysis of packet switched NoC and circuit switched NoC

We compare the structure of packet switched NoC with circuit switched NoC. For a packet switched NoC, the critical timing path length t_p depends on the virtual channel (VC) allocator. Even with the most advanced allocator design [112, 113], the latency of an allocator still scales up with $O(log(mq))$, where m is the number of output ports and q is the number of virtual channels per port.

For a circuit switched NoC, assuming data path and control path can have different clocks (applying preferrable source synchronized data transfer). For control path, its critical timing path length t_c is made up of an m-port arbiter (suppose no SDM channel sharing), plus some additional combination logic circuit delay. For the critical timing path of data path t_d, it is mainly an m-port crossbar delay, which is about $O(log(m))$.

Therefore, the control path length t_c of a circuit switched NoC can be close to, or shorter than the critical timing path length t_p of a packet switched NoC, while the data path t_d of a circuit switched NoC is much smaller than t_p.

Conclusions about the comparison

We study the performances of circuit switched NoC and packet switched NoC by assigning practical values to t_p, t_c, t_d getting from example designs. For packet switched NoC, we use a classical input-buffering virtual channel (VC) wormhole router architecture with a SIF VC allocator. For circuit switched NoC, we use our circuit switched NoC with a dynamic connection setup method.

The evaluation results obey our intuition and analysis. The general conclusion is that circuit switched NoC has better performance when the packet size is large; whereas packet switched NoC favors small packets. For medium size packets, packet switched NoC is better than circuit switched NoC only when the overall traffic load is high.

4.5 Future work

In this chapter, we have introduced our works related to multi-channel circuit switched NoCs. The future work could be focused on the following aspects:

- In-depth optimization of the architecture of the water-fall allocator. Currently, the critical timing path of the water-fall allocator scales up with $O(n)$, where n is the number of requesters. We will try to explore techniques which can reduce it to $O(log(n))$.
- Research on the techniques that enhance the success probability of multi-channel connection setup. We may apply a certain control mechanism based on feedback information to avoid the collision between setup requests and thus enhance the success probability.
- Implementation and evaluation of mixed packet and circuit switched NoCs. We have evaluated the pros and cons of circuit switched NoC and packet switched NoC, respectively. We think that in real applications, combining them together may achieve an even better performance. It will be an interesting topic on how to efficiently design and utilize packet switched and circuit switched NoCs together.

Chapter 5

Summary

This chapter summarizes the thesis and points out the future directions.

5.1 Thesis summary

Compared with packet switched NoC, circuit switched NoC has advantages in QoS guarantee, separation of control path and data path, smaller cost on buffers, and sometimes a high operating clock frequency. However, in the past decade, circuit switched NoC is not studied as intensively as packet switched NoC. In this thesis, we introduce our researches on circuit switched NoCs. We have focused on *dynamic connection setup*, *time division multiplexing*, and *spatial division multiplexing*.

- Traditionally, distributed dynamic connection setup in circuit switched NoC tries to use routing algorithms designed for packet switched NoC to search paths for connections. However, this results in low setup success probability, since the path diversity is failed to be explored. We propose a parallel probing algorithm to explore the path diversity and enhance the connection setup success rate. The merits of our algorithm are that, on one hand, it tries to search as many as possible paths in parallel. One the other hand, it tries to reduce the overhead of a connection setup attempt by removing the redundant paths as soon as possible. We have efficiently implemented this algorithm in hardware.

- TDM technique has been widely used in circuit switched NoCs for link sharing. However, the usage of TDM technique makes the establishment of connection difficult and the link bandwidth utilization low. We propose a double time-wheel technique to enable the utilization of probe based setup method in dynamic TDM connection setup. With this technique, the difficulties on TDM connection setup can be overcome and the hardware cost can also be reduced. Besides, we proposed a highway technique to enhance the link utilization of TDM NoCs.

- SDM is another popular link sharing method for circuit switched NoC. To better support the dynamic connection setup in SDM NoCs, we propose a single cycle allocator that guarantees both maximal matching quality and strong fairness. We design a MultiCS NoC which supports both SDM and sub-network to organize channels. A parallel probe based connection setup method is implemented in this NoC. Based on this NoC, we have investigated which is the optimal way of organizing multiple channels. Moreover, we have made a comparison between circuit switched NoC and packet switched NoC.

5.2 Future directions

Based on our current research results and according to our current considerations, we can continue on the following directions.

- Extending the parallel probing algorithm. Currently, our parallel probing path search algorithm is designed for mesh topology. However, this algorithm can be utilized for other topologies as well. As along as multiple minimal paths exist between two nodes on a certain topology, our parallel probing algorithm benefits the path search and connection setup.

- Exploring fault tolerance techniques in circuit switched NoC. First, we can extend the parallel probing algorithm for fault tolerance purpose. The parallel probing algorithm can tolerate a certain magnitude of path failures, since it searches the entire possible minimal paths between a source and a destination. It can find a fault-free path as long as the path exists. Second, the double time-wheel technique can also be utilized in error correction. For example, if a transient fault happens on a connection and causes a flit error, when the destination node detects the error, it can send a message through the backward path to notify the source node about the error. Then, the source node can resend the flit for error correction purpose.

- Providing multicast support in circuit switched NoC. Multicast communication is required by many practical applications, e.g. shared cache invalidation in a many-core system. In current stage, our work mainly focuses on the setup of unicast connections. However, it is possible to build up multicast connections by extending our current work. For example, with the parallel probing algorithm, a tree based multicast structure can be readily established with a single setup endeavour. However, in order to implement this feature, we need to solve tricky issues like contentions of setup requests, flow control on the multicast tree and so on.

- Exploring TDM techniques for asynchronous routers. In practice, quite often the NoCs are mesochronous, or even asynchronous. In this situation, keeping all the routers having a unanimous agreement on the slot time is not an easy task. There are tricky issues in slot counter resetting and updating [8, 114, 115, 116]. However, we might be able to relax the requirement on slot time synchronization. For example, instead of requiring a unanimous agreement on the global notion of time, we might just demand all the neighboring routers updating their slot time together. In such a way, we might be able to reduce the effort on slot time synchronization.

- Enhancing the connection setup success probability. Intuitively, if we spawn many setup requests at the same time or within a small period, the success probability of each setup request decreases because of the massive contentions between the setup requests. Therefore, if every source node can be aware of the contention situation of the network and schedule the setup requests to reduce contentions, the success probability of each setup attempt can be enhanced, setup delay can be reduced and throughput can be increased due to the reduction in the time spent on retrying failed setup requests.

- Application of the proposed techniques. In this thesis, we have showed our novel techniques such as parallel probing, double time-wheel, TDM-highway, WTF allocator and so on. In the future, we will study which kind of real applications can be benefited from our techniques, and how to deploy our techniques in real applications.

Bibliography

[1] K. DeHaven, "Extensible processing platform ideal solution for a wide range of embedded systems," *Extensible Processing Platform Overview White Paper*, 2010.

[2] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The nostrum backbone-a communication protocol stack for networks on chip," in *Proceedings of 17th International Conference on VLSI Design*, 2004, pp. 693–696.

[3] S. Park, T. Krishna, C.-H. Chen, B. Daya, A. Chandrakasan, and L. Peh, "Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45nm SOI," in *Proceedings of ACM/EDAC/IEEE Design Automation Conference (DAC'12)*, 2012, pp. 398 –405.

[4] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of Design, automation and test in Europe conference Exhibition (DATE'00)*, 2000, pp. 250–256.

[5] F. Moraes, N. Calazans, A. Mello, L. MÃűller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69–93, 2004.

[6] P.-H. Pham, J. Park, P. Mau, and C. Kim, "Design and implementation of backtracking wave-pipeline switch to support guaranteed throughput in network-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 2, pp. 270 –283, 2012.

[7] P. Wolkotte, G. Smit, N. Kavaldjiev, J. Becker, and J. Becker, "Energy model of networks-on-chip and a bus," in *Proceedings of International Symposium on System-on-Chip*, 2005, pp. 82 –85.

[8] R. A. Stefan, A. Molnos, and K. Goossens, "dAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up," *IEEE Transactions on Computers*, vol. 63, no. 3, pp. 583–594, 2014.

[9] P.-H. Pham, P. Mau, J. Kim, and C. Kim, "An on-chip network fabric supporting coarse-grained processor array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1 –5, 2012.

[10] C. Hilton and B. Nelson, "A flexible circuit switched NOC for FPGA based systems," in *Proceedings of International Conference onField Programmable Logic and Applications*, 2005, pp. 191 – 196.

[11] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, pp. 1374–1396, 1995.

[12] Y. Ben-Itzhak, I. Cidon, and A. Kolodny, "Delay analysis of wormhole based heterogeneous NoC," in *Proceedings of the fifth IEEE/ACM International Symposium on Networks on Chip (NoCS'11)*, 2011, pp. 161–168.

[13] I. Nousias and T. Arslan, "Wormhole routing with virtual channels using adaptive rate control for network-on-chip," in *Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06)*, 2006, pp. 420–423.

[14] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual channels in networks on chip: implementation and evaluation on HERMES NoC," in *Proceedings of the 18th annual symposium on Integrated circuits and system design*, 2005, pp. 178–183.

[15] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (NoC) architectures & contributions," *Journal of engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[16] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of systems architecture*, vol. 50, no. 2, pp. 105–128, 2004.

[17] A. Jantsch *et al.*, "The nostrum network-on-chip," *Royal Institute of Technology, Stockholm*, 2003.

[18] C. Feng, Z. Lu, A. Jantsch, J. Li, and M. Zhang, "A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for network-on-chip," in *Proceedings of the Third ACM International Workshop on Network on Chip Architectures*, 2010, pp. 11–16.

[19] M. Coenen, S. Murali, A. Ruadulescu, K. Goossens, and G. De Micheli, "A buffer-sizing algorithm for networks on chip using TDMA and credit-based end-to-end flow control," in *Proceedings of the 4th ACM International conference on Hardware/software codesign and system synthesis*, 2006, pp. 130–135.

[20] C. Schuck, S. Lamparth, and J. Becker, "artNoC - a novel multi-functional router architecture for organic computing," in *Proceedings of International Conference on Field Programmable Logic and Applications(FPL'07)*, 2007, pp. 371–376.

[21] N. Kavaldjiev, G. J. Smit, P. G. Jansen, and P. T. Wolkotte, "A virtual channel network-on-chip for GT and BE traffic," in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, 2006, pp. 6–pp.

[22] K. Goossens, J. Dielissen, O. P. Gangwal, S. G. Pestana, A. Radulescu, and E. Rijpkema, "A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification," in *Proceedings of the conference on Design, Automation and Test in Europe (DATE'05)*, 2005, pp. 1182–1187.

[23] K. Goossens, "Formal methods for networks on chips," in *Fifth International Conference on Application of Concurrency to System Design ACSD'05*, 2005, pp. 188 – 189.

[24] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens, “A TDM NoC supporting QoS, multicast, and fast connection set-up,” in *Proceedings of the conference on Design, Automation and Test in Europe (DATE’12)*, 2012, pp. 1283–1288.

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

[26] A. Hansson, M. Subburaman, and K. Goossens, “aelite: A flit-synchronous network on chip with composable and predictable services,” in *Proceedings of the conference on design, automation and test in Europe (DATE’03)*, 2009, pp. 250–255.

[27] J. Hu and R. Marculescu, “Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints,” in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE’14)*, vol. 1, 2004, pp. 234–239.

[28] A. Hansson, K. Goossens, and A. Radulescu, “A unified approach to constrained mapping and routing on network-on-chip architectures,” in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2005, pp. 75–80.

[29] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, and H. Corporaal, “Dynamic time-slot allocation for QoS enabled networks on chip,” in *Proceedings of the 3rd IEEE Workshop on Embedded Systems for Real-Time Multimedia*, 2005, pp. 47–52.

[30] R. Stefan and K. Goossens, “A TDM slot allocation flow based on multipath routing in NoCs,” *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 130–138, 2011.

[31] S. Stuijk, T. Basten, M. Geilen, A. H. Ghamarian, and B. Theelen, “Resource-efficient routing and scheduling of time-constrained network-on-chip communication,” in *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD’06)*, 2006, pp. 45–52.

[32] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, "A statically scheduled time-division-multiplexed network-on-chip for real-time systems," in *Proceedings of the Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS'12)*, 2012, pp. 152 –160.

[33] R. Stefan and K. Goossens, "Multi-path routing in time-division-multiplexed networks on chip," in *Proceedings of International Conference on Very Large Scale Integration (VLSI-SoC)*, 2009, pp. 109–114.

[34] L. Tong, Z. Lu, and H. Zhang, "Exploration of slot allocation for On-Chip TDM virtual circuits," in *Proceedings of Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD'09)*, 2009, pp. 127–132.

[35] Z. Lu and A. Jantsch, "Slot allocation using logical networks for TDM virtual-circuit configuration for network-on-chip," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD'07)*, 2007, pp. 18–25.

[36] B. Akesson, A. Minaeva, P. Sucha, A. Nelson, and Z. Hanzalek, "An efficient configuration methodology for time-division multiplexed single resources," in *Proceedings of Real-Time and Embedded Technology and Applications Symposium (RTAS'15)*, 2015, pp. 161–171.

[37] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 1021 –1034, 2008.

[38] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H. 264/AVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, 2007.

[39] R. Stefan, A. B. Nejad, and K. Goossens, "Online allocation for contention-free-routing NoCs," in *Proceedings of ACM Interconnection Network Architecture: On-Chip, Multi-Chip Workshop (INA-OCMC'12)*, 2012, pp. 13–16.

[40] R. Stefan, *Resource Allocation in Time-Division-Multiplexed Networks on Chip.* Delft University of Technology, 2012.

[41] O. Moreira, J. J.-D. Mol, and M. Bekooij, "Online resource management in a multiprocessor with a network-on-chip," in *Proceedings of the 2007 ACM symposium on Applied computing*, 2007, pp. 1557–1564.

[42] D. Wiklund and D. Liu, "SoCBUS: switched network on chip for hard real time embedded systems," in *Proceedings of Parallel and Distributed Processing Symposium*, 2003, pp. 8–15.

[43] M. Winter and G. Fettweis, "Guaranteed service virtual channel allocation in NoCs for run-time task scheduling," in *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE'11)*, 2011, pp. 1–6.

[44] ——, "A network-on-chip channel allocator for run-time task scheduling in multi-processor system-on-chips," in *Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD'08)*, 2008, pp. 133 –140.

[45] D. Liu, D. Wiklund, E. Svensson, O. Seger, and S. Sathe, "SoCBUS: The solution of high communication bandwidth on chip and short TTM," in *Proceedings of Real-Time and Embedded Computing Conference*, 2002.

[46] A. K. Lusala and J. Legat, "Combining SDM-based circuit switching with packet switching in a NoC for real-time applications," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'11)*, 2011, pp. 2505–2508.

[47] J. Heisswolf, R. König, and J. Becker, "A scalable NoC router design providing QoS support using weighted round robin scheduling," in *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, 2012, pp. 625–632.

[48] F. Pakdaman, A. Mazloumi, and M. Modarressi, "Integrated circuit-packet switching NoC with efficient circuit setup mechanism," *The Journal of Supercomputing*, pp. 1–21, 2014.

[49] P.-H. Pham, P. Mau, J. Kim, and C. Kim, "An on-chip network fabric supporting coarse-grained processor array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, vol. 21, no. 1, pp. 178–182, 2013.

[50] A. M. Hinz, “Pascal’s triangle and the tower of hanoi,” *American Mathematical Monthly*, pp. 538–544, 1992.

[51] P. E. Black, “Manhattan distance,” *Dictionary of Algorithms and Data Structures*, vol. 18, p. 2012, 2006.

[52] A. K. Lusala and J. Legat, “Combining circuit and packet switching with bus architecture in a NoC for real-time applications,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS’10)*, 2010, pp. 2880–2883.

[53] A. K. Lusala and J.-D. Legat, “A SDM-TDM-based circuit-switched router for on-chip networks,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 3, p. 15, 2012.

[54] A. K. Lusala and J. Legat, “A hybrid router combining SDM-based circuit swictching with packet switching for on-chip networks,” in *Proceedings of International Conference on Reconfigurable Computing and FPGAs (ReConFig’10)*, 2010, pp. 340–345.

[55] P.-H. Pham, J. Park, P. Mau, and C. Kim, “Design and implementation of backtracking wave-pipeline switch to support guaranteed throughput in network-on-chip,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 2, pp. 270–283, 2012.

[56] M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, “Fault tolerant algorithms for network-on-chip interconnect,” in *Proceedings. IEEE Computer society Annual Symposium on VLSI*, 2004, pp. 46–51.

[57] V. Rantala, T. Lehtonen, and J. Plosila, *Network on chip routing algorithms.* Citeseer, 2006.

[58] Y. Xu, J. Zhou, and S. Liu, “Research and analysis of routing algorithms for NoC,” in *Proceedings of the 3rd International Conference on Computer Research and Development (ICCRD’11)*, vol. 2, 2011, pp. 98–102.

[59] U. Y. Ogras, J. Hu, and R. Marculescu, “Key research problems in NoC design: a holistic perspective,” in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2005, pp. 69–74.

[60] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, "Fixed priority pre-emptive scheduling: An historical perspective," *Real-Time Systems*, vol. 8, no. 2-3, pp. 173–198, 1995.

[61] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, p. 1, 2006.

[62] J. Zhang and H. Gu, "A partially adaptive routing algorithm for Benes network on chip," in *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT'09)*, 2009, pp. 614–618.

[63] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, 2005.

[64] F. Samman, T. Hollstein, and M. Glesner, "Adaptive and deadlock-free tree-based multicast routing for networks-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1067–1080, 2010.

[65] W. Hu, Z. Lu, A. Jantsch, and H. Liu, "Power-efficient tree-based multicast support for networks-on-chip," in *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, 2011, pp. 363–368.

[66] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: goals, evolution, lessons, and future," in *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC'10)*, 2010, pp. 306–311.

[67] K. Goossens, J. Dielissen, J. v. Meerbergen, P. Poplavko, A. Rădulescu, E. Rijpkema, E. Waterlander, and P. Wielage, "Guaranteeing the quality of services in networks on chip," in *Networks on Chip*. Springer US, 2004-01-01, pp. 61–82.

[68] T. Marescaux and H. Corporaal, "Introducing the SuperGT network-on-chip; SuperGT QoS: more than just GT," in *Proceedings of the*

44th ACM/IEEE Design Automation Conference (DAC'07), 2007, pp. 116–121.

[69] Y. Wang, K. Zhou, Z. Lu, and H. Yang, "Dynamic TDM virtual circuit implementation for NoC," in *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems (APCCAS'08)*, 2008, pp. 1533–1536.

[70] J. Sparsø, E. Kasapaki, and M. Schoeberl, "An area-efficient network interface for a TDM-based Network-on-Chip," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'13)*, 2013, pp. 1044–1047.

[71] N. Concer, A. Vesco, R. Scopigno, and L. P. Carloni, "A dynamic and distributed TDM slot-scheduling protocol for QoS-oriented networks-on-chip," in *Proceedings of IEEE 29th International Conference on Computer Design (ICCD'11)*, 2011, pp. 31–38.

[72] J. Yin, P. Zhou, S. S. Sapatnekar, and A. Zhai, "Energy-efficient time-division multiplexed hybrid-switched NoC for heterogeneous multicore systems," in *Proceedings of IEEE 28th International of Parallel and Distributed Processing Symposium*, 2014, pp. 293–303.

[73] U. M. Mirza, F. Gruian, and K. Kuchcinski, "Design space exploration for streaming applications on multiprocessors with guaranteed service NoC," in *Proceedings of ACM Sixth International Workshop on Network on Chip Architectures*, pp. 35–40.

[74] N. Ma, Z. Lu, and L. Zheng, "System design of full HD MVC decoding on mesh-based multicore NoCs," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 217–229, 2011-03.

[75] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. Das, "Exploring fault-tolerant network-on-chip architectures," in *Proceedings of International Conference on Dependable Systems and Networks (DSN'06)*, 2006, pp. 93 –104.

[76] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks on chips," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 434–442, 2005.

[77] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage, “An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration,” in *Proccedings of Design, Automation and Test in Europe Conference and Exhibition (DATE’04)*, vol. 2, 2004, pp. 878 – 883.

[78] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest, “Spatial division multiplexing: a novel approach for guaranteed throughput on NoCs,” in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS’05)*, 2005, pp. 81–86.

[79] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, “Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip,” *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1182 –1195, 2008.

[80] J. Lim, E. Hunt Siow, Y. Ha, and P. Meher, “Providing both guaranteed and best effort services using spatial division multiplexing NoC with dynamic channel allocation and runtime reconfiguration,” in *Proceedings of International Conference on Microelectronics (ICM’08)*, 2008, pp. 329 –332.

[81] M. Modarressi, H. Sarbazi-Azad, and M. Arjomand, “A hybrid packet-circuit switched on-chip network based on SDM,” in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE’09)*, 2009, pp. 566–569.

[82] C. Gomez, M. Gomez, P. Lopez, and J. Duato, “Exploiting wiring resources on interconnection network: Increasing path diversity,” in *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP’08)*, 2008, pp. 20 –29.

[83] P. Ou, J. Zhang, H. Quan, Y. Li, M. He, Z. Yu, X. Yu, S. Cui, J. Feng, S. Zhu *et al.*, “A 65nm 39gops/w 24-core processor with 11tb/s/w packet-controlled circuit-switched double-layer network-on-chip and heterogeneous execution array,” in *Proceedings of IEEE International Conference on Solid-State Circuits Conference Digest of Technical Papers (ISSCC’13)*, 2013, pp. 56–57.

[84] D. Schinkel, E. Mensink, E. Klumperink, E. van Tuijl, and B. Nauta, "Low-power, high-speed transceivers for network-on-chip communication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 12 –21, 2009.

[85] D. Walter, S. Hoppner, H. Eisenreich, G. Ellguth, S. Henker, S. Hanzsche, R. Schuffny, M. Winter, and G. Fettweis, "A source-synchronous 90gb/s capacitively driven serial on-chip link over 6mm in 65nm CMOS," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC'12)*, 2012, pp. 180 –182.

[86] A. Mandal, S. P. Khatri, and R. N. Mahapatra, "A fast, source-synchronous ring-based network-on-chip design," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE'12)*, 2012, pp. 1489–1494.

[87] A. K. Lusala and J.-D. Legat, "Combining SDM-based circuit switching with packet switching in a router for on-chip networks," *International Journal of Reconfigurable Computing*, vol. 2012, pp. 1–16, 2012.

[88] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* Morgan Kaufmann, 2003-12-18.

[89] D. U. Becker and W. J. Dally, "Allocator implementations for network-on-chip routers," in *ACM Conference on High Performance Computing Networking, Storage and Analysis (SC'09)*, 2009, pp. 52:1–52:12.

[90] G. Costa and C. Stirling, "Weak and strong fairness in CCS," *Information and Computation*, vol. 73, no. 3, pp. 207–244, 1987-06.

[91] R. J. van Glabbeek and P. Höfner, "CCS: It is not fair!" *Acta Informatica*, vol. 52, no. 2-3, pp. 175–205, 2015.

[92] Y. Tamir and H.-C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, pp. 13 –27, 1993.

[93] J. G. Delgado-Frias and G. B. Ratanpal, "A VLSI wrapped wave front arbiter for crossbar switches," in *Proceedings of the 11th Great Lakes symposium on VLSI (GLSVLSI '01)*, 2001, pp. 85–88.

[94] J. Delgado-Frias and G. Ratanpal, “A VLSI crossbar switch with wrapped wave front arbitration,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 1, pp. 135 – 141, 2003-01.

[95] W. Olesinski, H. Eberle, and N. Gura, “PWWFA: The parallel wrapped wave front arbiter for large switches,” in *Proceedings of workshop on High Performance Switching and Routing (HPSR’07)*, 2007-06-30, pp. 1 –6.

[96] J. Hurt, A. May, X. Zhu, and B. Lin, “Design and implementation of high-speed symmetric crossbar schedulers,” in *Proceedings of IEEE International Conference on Communications (ICC ’99)*, vol. 3, 1999, pp. 1478 –1483.

[97] S. S. Mukherjee, F. Silla, P. Bannon, J. Emer, S. Lang, and D. Webb, “A comparative study of arbitration algorithms for the Alpha 21364 pipelined router,” in *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems (ASPLOS’02)*, 2002, pp. 223–234.

[98] N. McKeown, “The iSLIP scheduling algorithm for input-queued switches,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188 –201, 1999-04.

[99] S. S. Yeob, S. Y. Nam, and D. K. Sung, *Desynchronized Two-Dimensional Round-Robin Scheduler for Input Buffered*, 2002.

[100] E. L. Hahne, “Round-robin scheduling for max-min fairness in data networks,” *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, vol. 9, pp. 1024–1039, 1991.

[101] M. A. Iannone, “Round robin schedules.” *Mathematics Teacher*, vol. 76, no. 3, pp. 194–95, 1983.

[102] E. L. Hahne, “Round robin scheduling for fair flow control in data communication networks.” DTIC Document, Tech. Rep., 1986.

[103] Y. J. Yoon, N. Concer, M. Petracca, and L. Carloni, “Virtual channels vs. multiple physical networks: a comparative analysis,” in *Proceedings of IEEE Design Automation Conference (DAC’10)*, 2010, pp. 162–165.

[104] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni, "Virtual channels and multiple physical networks: Two alternatives to improve NoC performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 12, pp. 1906–1919, 2013.

[105] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 3, pp. 277 –282, 1991.

[106] S. Liu, A. Jantsch, and Z. Lu, "A fair and maximal allocator for single-cycle on-chip homogeneous resource allocation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2229–2233, 2014.

[107] M. Angermann, "Differences in cost and benefit of prefetching in circuit-switched and packet-switched networks," in *Proceedings of International Conference on Telecommunications (ICT'03)*, vol. 2, 2003, pp. 1084–1090.

[108] K.-C. Chang, J.-S. Shen, and T.-F. Chen, "Evaluation and design trade-offs between circuit-switched and packet-switched NOCs for application-specific SOCs," in *Proceedings of ACM/IEEE Design Automation Conference (DAC'06)*, 2006, pp. 143 –148.

[109] N. Chin-Ee and N. Soin, "Qualitative and quantitative evaluation of a proposed circuit switched network-on-chip," in *Proceedings of IEEE International Conference on Semiconductor Electronics (ICSE'10)*, 2010, pp. 108–113.

[110] ——, "A study on circuit switching merits in the design of network-on-chip," in *Proceedings of International Conference on Computer and Communication Engineering (ICCCE'10)*, 2010, pp. 1–5.

[111] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of Design Automation Conference (DAC'01)*, 2001, pp. 684–689.

[112] H. J. Chao, C. H. Lam, and X. Guo, "Fast ping-pong arbitration for inputâĂŞoutput queued packet switches," *International Journal of Communication Systems*, vol. 14, no. 7, pp. 663–678, 2001.

[113] J. M. JOU and Y. L. LEE, "An optimal round-robin arbiter design for NoC," *Journal of information science and engineering*, vol. 26, pp. 2047–2058, 2010.

[114] E. Kasapaki and J. Sparsø, "Argo: A time-elastic time-division-multiplexed noc using asynchronous routers," in *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'14)*, 2014, pp. 45–52.

[115] I. Kotleas, D. Humphreys, R. Sorensen, E. Kasapaki, F. Brandner, and J. Sparsø, "A loosely synchronizing asynchronous router for tdm-scheduled nocs," in *Proceedings of IEEE/ACM International Symposium on Networks-on-Chip (NoCS'14)*, 2014, pp. 151–158.

[116] Z. Lu, "Cross clock-domain TDM virtual circuits for networks on chips," in *Proceedings of IEEE/ACM International Symposium on Networks on Chip (NoCS'11)*, 2011, pp. 209–216.

Paper A

Parallel Probing: Dynamic and Constant Time Setup Procedure in Circuit Switching NoC

Shaoteng Liu, Axel Jantsch and Zhonghai Lu

Presented at the Design, Automation Test in Europe Conference Exhibition (DATE'12), Dresden, German, 2012. Included in the proceedings, pages: 1289–1294.

Parallel Probing: Dynamic and Constant Time Setup Procedure in Circuit Switching NoC

Shaoteng Liu, Axel Jantsch, Zhonghai Lu
KTH Royal Institute of Technology, Sweden

***Abstract*- We propose a circuit switching Network-on-chip with a parallel probe searching setup method, which can search the entire network in constant time, only dependent on the network size but independent of the network load. Under a specific search policy, the setup procedure is guaranteed to terminate in time 3D+6 cycles, where D is the geometric distance between source and destination. If a path can be found, the method succeeds in 3D+6 cycles; if a path cannot be found, it fails in maximum 3D+6 cycles. Compared to previous work, our method can reduce the setup time and enhance the success rate of setups. Our experiments show that compared with a sequential probe searching method, this method can reduce the search time by up to 20%. Compared with a centralized channel allocator method, this method can enhance the success rate by up to 20%.**

1. INTRODUCTION AND RELATED WORK

Several NoCs offer guaranteed services to meet the QoS demand of applications [1][2][3][4][5]. For example, some of them utilize packet switching mechanism with time division multiplexing channels [2][4][5], while others utilize pure circuit switching mechanism [1][3]. They all adopt the idea of setting up a dedicated path before data can be transferred. A main challenge is how to efficiently search a path and allocate the communication resources for it.

Some methods try to solve this problem at compilation time [6][7]. However, as mentioned in [8], they face the difficulty that applications like H.264 [11] and the possibility of several applications running in parallel do not allow an efficient static policy with task mapping and channel allocation. Thus, dynamic path searching methods are a flexible alternative.

Winter and Fettweis [2][8] developed a *centralized* way to realize dynamic path searching and channel allocation. They designed a Network-on-chip Channel Allocator for the Aetherial NoC [5]. One of the nodes is designated as the "NoCmanager". Inside this node, a special component called Hardware Graph Array (HAGAR) is used. HAGAR stores all channel usage information of the network. If other nodes try to set up a guaranteed path, they must first send their requests to the NoCmanager node via a best effort network, which is the Aetherial packet switching network. Then the NoCmanager node starts to deal with the request and uses HAGAR to compute the path. After computation, the NoC manager will send back the routing information in order to set-up the guaranteed service path as requested. The NoC Channel Allocator is a centralized solution for dynamic path configuration. The advantage is that the NoCmanager node can work very fast. The disadvantage is that it is not a scalable solution. Also, since requests are sent via a best effort network to the "NoCmanager", the delay of setting up a path is neither predictable nor guaranteed.

To overcome such a scalability issue, distributed path searching method was developed. Pham et al. [1] designed a Backtracking Wave-pipeline circuit switching NoC, which supports sequential probe search. During the path searching phase, a probe is sent out. As the probe travels from the source node towards the destination, it reserves the channels it has passed for future data transfer. When this probe encounters congestions, it will backtrack one hop, cancel the last channel it has booked, and try another way. When this probe finally reaches the destination, a path is established and the data transfer phase can be launched. If no path can be found, eventually the probe will backtrack to the source node. This distributed method with circuit switching mechanism has the advantage of supporting many nodes searching their path in parallel. But the disadvantage is that when the majority of circuit links are already in use, the backtracking based search may take a long time.

In this paper, we develop a parallel probe searching approach. Our work has the following properties:

1) The parallel probing can be combined with several different retry policies, that lead to different trade-offs and properties.
2) Under no-retry policy, if a shortest path connection can be found, it is guaranteed to be found in exactly 3*D +6 cycles; if the search fails, it fails in maximum 3*D+6 cycles.
3) Under retry-for-free-path policy, if a free path exists, it will always be found in maximum $6N^{1.5}$ cycles, with N being the number of nodes.
4) On average our method has shorter setup latency than previous methods.
5) The switch has a simple structure with an inexpensive and efficient implementation.

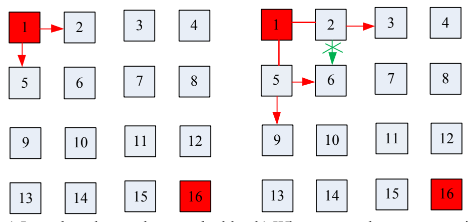
We have simulated our design and compared it with above mentioned work of [1] and [2]. The results show that our work has advantages in delay, success rate and area.

2. DETAILS OF PARALLEL PROBING

2.1. Intuition

As shown in Fig. 1, node 1 tries to set-up a path to the destination node 16. During the first cycle the source node sends out two probes to the neighboring nodes 2 and 5. In the second hop each probe splits into two probes and both continue to travel towards the destination along all possible minimum paths.

In the third hop node 6 receives two probes from the same setup request. One of them is cancelled and all the channels it has booked before are released. However, the channel between node 1 and node 2 is not released, because it is still needed for the probe that has travelled further to node 3. In this way a wavefront of probes travel through the network and reach the destination on a minimal path. The time is exactly 3D, where D is the distance in terms of hops, and it takes 3 cycles to traverse each hop. When a probe successfully reaches the destination, an acknowledgement is sent back to the source node.



a) In each node a probe may double. b) When two probes meet, one is cancelled.

Fig. 1 An example of the parallel probe searching method.

Whenever two probes of the same request meet, one of them is regarded as redundant and is canceled, as shown in Fig. 1 b), and all channels used only by the canceled probe are released. The cancellation process proceeds backwards hop by hop. The switch does a cancellation based on the stored connectivity information that binds an input port to an output port. When a cancel signal appears on an output port from a downstream switch, the corresponding input port is looked up, the connection is canceled, and the cancel signal is forwarded upstream to the input port. Applying this mechanism, if several possible paths exist, one and only one of them can be finally booked, just as desired.

2.2. Structure of the switch

Fig. 2 shows the interface of each switch to the neighboring switches and to the local node. In a mesh topology, every switch is connected to its four neighbors and to the local resource node. Each link has a duplex data channel. This data channel is used for carrying the probe during setup and for transmitting data when a connection has been established. Each probe is one "flit" length. Every data channel is associated with an answer (ANS) signal consisting of 2 bit, which goes in the opposite direction to data channel, and 1 bit for a Request signal, which travels in the same direction as data channel. When the request signal is logic '1', a probe search is running or data transfer is active. When request signal is '0', it denotes idle state, and an established path will be released. The usage of ANS signal is listed in TABLE 1, which will be introduced in the following section.

TABLE 1 THE USAGE OF ANS (2 BITS)

Value	**Usage**
00	Path search continue/Destination is idle
01	Path cancel due to contention
10	Path cancel due to blockage
11	Path established/Busy destination

2.3. Operation flow

As shown in Fig. 3, our circuit switching network has six operation phases. The details are explained in the following.

2.3.1. Probe sendout

In this phase probes are generated and sent out. The request signal is set to '1'. The probe format is shown in TABLE 2, which contains source node address, destination node address, high level priority and low level priority. As a probe travels inside the network, it books the data channels together with the associate ANS and request signal. The probe itself is forwarded to the next node or nodes towards the destination. When a probe can move forward, the ANS to its previous node is set to "00".

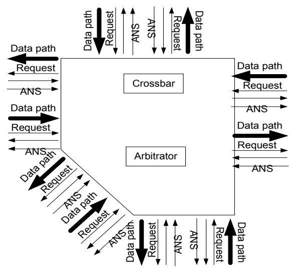


Fig. 2 Signal connection of a 2*2 mesh

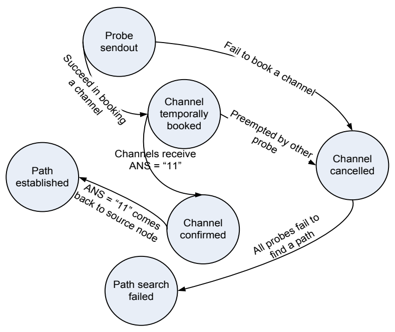

Probe sendout
Fail to book a channel
Succeed in booking a channel
Channel temporally booked
Preempted by other probe
Channel cancelled
Channels receive ANS = "11"
Path established
ANS = "11" comes back to source node
Channel confirmed
All probes fail to find a path
Path search failed

Fig. 3 Phases of operation

TABLE 2 Probe format

Src.Addr	Dest. Addr	High Priority	Low Priority

2.3.2. Channel temporally booked

A channel has 3 states, which are free, booked and confirmed. When a probe enters into a node, after winning arbitration, then

1)If the channels demanded by this probe are in free state, these channels are booked and switched into booked state.

2)If the channel demanded by this probe has already been booked by another probe, but this probe has a higher priority, it can preemptively book this channel. In this case, the path booked by the old probe will be cancelled.

If a probe succeeds in booking one channel, the ANS signal will remain "00".

2.3.3. Channel cancelled

Since a probe can have up to two desired channels in different directions when it enters a node, if and only if the probe is unable to book any channels, then we regard this probe as "failed". When a probe fails, the channel between the previous node and the current node is cancelled. Several factors can cause a probe to fail:

1) Its priority is lower than the probes which are demanding the same channel. This situation is caused by either the

probe loosing arbitration, or the channel booked by this probe is preempted by others.

2) Two probes carrying the same set-up request meet at the same node. One of them succeeds and the other fails.
3) All desired channels are used up by other connections and are in confirmed state.

If the failure is caused by the case 1), the ANS signal will be set to "01". If the failure is due to case 2) or 3), the ANS signal will be set to "10". Both "01" and "10" ANS will inform the previous node to release the channel.

Since a probe may have two desired directions, it is possible that a probe has case 1) failure in one direction, and case 2) or 3) failure in the other. In this situation case 1) always has higher priority.

2.3.4. Channel confirmed

When a probe reaches its destination, the ANS signal is set to "11" and transferred back. Channels along this probe's path will turn into "confirmed" state after receiving ANS "11". Confirmed channels can no longer be preempted.

2.3.5. Path established

Finally, when a source node receives a "11" ANS signal, then a connection is established and data transfer can commence.

2.3.6. Path search fail

When the source node receives a "01" or "10" ANS signal, the path search request has failed, and the reasons are distinguished as follows:

1) If the ANS is "01", it means that one of its probes has once contended with other active probes, and lost because of its low priority.
2) If the ANS is "10", it means that the probe has searched the entire network, but no minimum path is currently available.

Using this distinction, different policies can be applied to achieve different effects, e.g. see Fig. 11.

2.4. Detailed switch structure

According to the operation flow, the internal structure of a switch is shown in Fig. 4. It is divided into two parts: *control path* and *data path*. The *data path* transfers data through the configured *data path* crossbar. The *control path* is used to set up or tear-down a *data path*. The *control path* and *data path* share the same input and output wires.

In the *control path*, there are two crossbars, internal probe crossbar and control signal crossbar. Besides the crossbars, there is one arbiter, 5 input and 5 output controllers.

The probe crossbar is used to transfer a probe from one input to one output. The control signal crossbar is used to transfer requests and the ANS signal to the corresponding output.

The arbiter is used to solve contention between input probes and probes that already book a channel. The arbiter compares their priority and decides which probe wins.

Inside the input controller there is a channel monitor, a failure type monitor and an FSM. The channel monitor records the channels booked by the current probe. If the number of channels booked by the current probe becomes zero, then the probe is regarded as "failed". Whether the ANS signal transfers back a "01" or a "10" is decided by the failure type monitor. The failure type monitor remembers the cause of a failure, as described in section 2.3.3. The possible FSM states are *idle, prepare, booked, cancellation* and *fixed*. Its state transition graph is shown in Fig. 5 a). For example, when the ANS signal "11" is received, the input controller changes its state to "fixed" and transfers the "11" ANS signal to the previous node.

The output controller monitors and changes the states of the corresponding channel. An FSM is used inside the output controller; its states are shown in Fig. 5 b).

A probe needs two clock cycles to travel through the entire *control path.*

2.5. Priority strategy

As we mentioned above, contention between probes carrying different requests is a key point in this parallel probing method. Therefore, we have to wisely design our priority policy to solve contention.

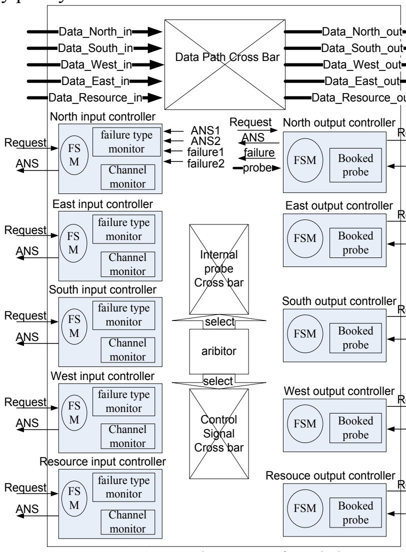


Fig. 4 Internal structure of a switch

We propose the following priority strategy:

1) The older the "age", the higher the priority. The "age" here can be understood as the time (in number of clock cycles) between current time and the first sent-out time of the request. The two level priorities are used to represent age.
2) Probes with higher priority can preempt channels booked by lower priority probes, if channels are in booked state.

This policy is used to avoid live lock cause by mutual blockage. Consider four requests A:1→3, B: 4→2, C: 2→4, D: 3→1 (Fig. 6). If these four requests are sent out at the same time, then they will block each other. Request A booked channel 1→4 and 1→2, then attempts to take channel 4→3 and 2→3. However, channel 4→3 has been booked by request B, and 2→3 by request C. Thus A is blocked by requests C and B. The situation is similar for requests B, C, and D. Without preemption, none of these probes can proceed, and retrying in a deterministic manner will not help. Thus, we use preemption to ensure at least one of them can preempt the channels booked by the others and proceed to its destination.

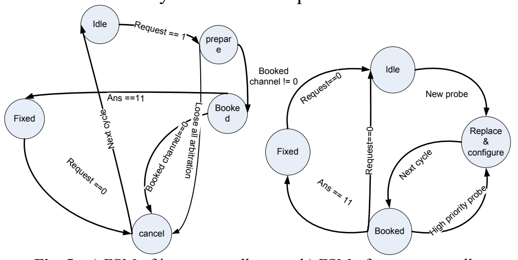


Fig. 5 a) FSM of input controller b) FSM of output controller

3) The source node id is used to avoid the stalemate when two different probes have the same priority. Rather than randomly selecting, the winner will be the one with the largest node id. This determinism leads to a winner-gets-all arbitration, which is required by retry-for-free-path policy (introduced in next section). Winner-gets-all arbitration is depicted in Fig. 6 b), suppose probe A and B with the same priority are contending for both channels 1 and 2. In winner-gets-all arbitration, one of them will win both channels, avoiding the situation that A gets channel 1 and B gets channel 2 which may for instance happen under random selection.

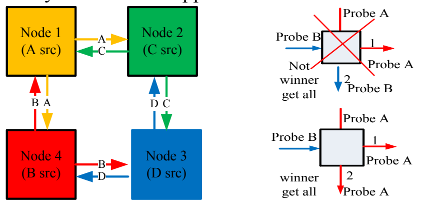


Fig. 6 a) live lock of probes **b)** Winner-gets-all

2.6. Time consumed in parallel probe search.

For every switch, it takes two cycles for a probe to traverse a switch, and it takes 1 cycle for the ANS signal to transfer back. So, it takes at most 3*D+6 cycles for a probe to travel from source to destination and send back the ANS signal (D is the hop distance between source and destination). 6 cycles is the overhead consumed in the source and destination nodes. Therefore, in an n*n mesh the worst case for a single search takes 3*(2*n-2)+6 cycles, no matter if the result is a success or a fail. In other words, it means the time for every single search is predictable and bounded, and has a complexity of O(n).

We have studied several policies.

1) **No-retry**. If a source node receives ANS “01” or “10”, it will mark the request as “failed”, then pick new request from the queue and send it out. Since every request just takes one single search, the maximum set-up time for a request in a n*n mesh is 3*(2*n-2)+6 cycles.

2) **Retry-for-free-path**. If the source node receives ANS “01”, the probe failed due to contention with other active probes. This means there might be a free path but the probe failed to find it. After some delay, the source will retry the request until the ANS becomes “10” or “11”, see Fig. 7Fig. 6. In experiments the retry interval is fixed to 3*(2*n-2)+6 cycles.

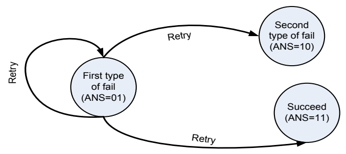


Fig. 7 Retry-for-free-path policy

The maximum number of retries required for a single search using retry-for-free-path policy can be calculated. Since the priority is increasing with the “age” of a probe, as the number of retries increases, the priority will also increase. Besides, one source node can only send out one request at a time, thus during every retry interval, there must be a node with the highest priority which never loses arbitration. Suppose at time t, $\alpha*n^2$ nodes are sending out requests, where α is the percentage of nodes which enabled to send out set-up requests (called *master percentage*). So the max retry times for a request to finish a retry-for-free-path search is $\alpha*n^2$, because during previous $\alpha*n^2$ -1 retry intervals, $\alpha*n^2$-1 other requests have finished their search, and this one has become the “oldest” one with the highest priority. It will finish a retry-for-free-path search without failure due to contention. In this case the set-up time spent for a single search using this policy is $\alpha*n^2$ *[3(2n-2)+6] cycles. And the time complexity is $O(n^3)$.

3) **Retry-until-success**. In this policy, the source node keeps retrying a request until it successfully sets up a connection. In this case the worst search time is unbounded, because it is unknown when a free path becomes available.

3. EXPERIMENT AND SIMULATION RESULTS

3.1. Simulation method

As in Fig. 8, in each node a request generator generates set-up requests according to certain probability and pushes them into a queue. A FSM take a request out of the queue and send it out when the previous request has been accomplished or abandoned. After sending out a request, the FSM waits for the ANS signal to decide what to do next. In our experiment, we have studied the three kind of policies mentioned above.

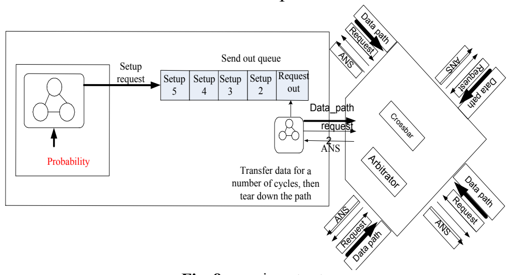


Fig. 8 experiment setup

3.2. Experiment result

In order to compare with the single probe searching technique with backtracking, we simulated an 8*8 mesh network. The simulation was performed under uniform random traffic of requests. The duration of data transmission (lifetime) was 400 probe cycles after the path had been established. The inter-arrival times of requests obey a Poisson distribution. We used the retry-until-success policy. Each source sends out 3000 set-up requests, and first 300 and last 300 are discarded because of warm-up and tail phases. The *total delay* includes the waiting time for a request in the queue and the *setup delay* for a request, which extends from first time sending until the final success.

The average *total delay* versus offered load is shown in Fig. 9. The delay data for sequential probe is extracted from [1]. Here *offered load* refers to the duration of a path times injection rate. Suppose injection rate of requests is 1/2000 cycles, and the duration (lifetime) of a path is 400 cycles, then the offered load is 400/2000=0.2.

As shown in Fig. 9, our parallel probing outperforms sequential probe searching with backtracking. For example, the turning point in our case is delayed until offered load is 0.25, and the delay at 0.2 load is 21% reduced.

In order to compare with the centralized HAGAR solution [2], we use *request success rate* versus *route rate*. The *request success rate* denotes the ratio between established and desired paths and indicates how many of the requested paths could be established. *Route rate* refers to the portion of clock cycles in which a node is used for transferring data [2]. And *master percentage* means the percentage of nodes which can send out set-up requests. They are uniformly randomly distributed in the system.

$$route\ rate = \frac{paths\ wanted * lifte\ time}{simulation\ cycles}$$

We simulated 5,000,000 cycles, of which the first 1,000,000 cycles were discarded as warm up.

However, the success rate should be related to the setup delay to make it a useful metric. In [2][8] setup delay data is not reported. In Fig. 10, we use the retry-for-free-path policy to compare with HAGAR[2]. Our method has a better success rate when route rate is between 0.1 and 0.8. In the range 0.1-0.2 parallel probing has a 20% higher success rate.

We also compared a 6*6 network with 200 cycle lifetime. In this case, parallel probing outperforms HAGAR at every point. Our method has around 50% improvement over HAGAR in terms of success rate at route rates 0.8-1.0. Also, in a 16*16 network with 1000 cycles lifetime parallel probing is superior by 50% for route rates 0.8-1.0. Due to page limitation, figures are not listed here.

In addition, we compared the three mentioned policies of our parallel probe searching method. Here we define *send-out success rate* as the ratio between succeeded requests and the requests sent out from the queue. It indicates the success probability of a single request after sending it out. The relationships between route rate and *send-out success rate* and delay are shown in Fig. 11 and Fig. 12, respectively. Fig. 12 shows the delays only up to the saturation point. We find that, 1) retry-until-success policy has a 100% *send-out success rate*, at the expense of long latency at high network loads. Even in a saturated network every request eventually gets served, but the delay is unbounded. 2) The *send-out success rate* of retry-for-free-path policy stays around 0.54 when route rate is greater than 0.3. This is because the maximum speed of sending out requests becomes less than the speed of generating requests. The average delay of a request waiting in the queue keeps increasing. Although more requests are generated, a limitation exits for the requests that can be sent out during a fixed time interval. Therefore, the *send-out success rate* stabilizes even if the route rate still grows. 3) No-retry policy has the worst *send-out success rate* but the best delay performance. In our experiments the requests generation rate never exceeds the requests are sent out rate, even as route rate reaches 1.0.

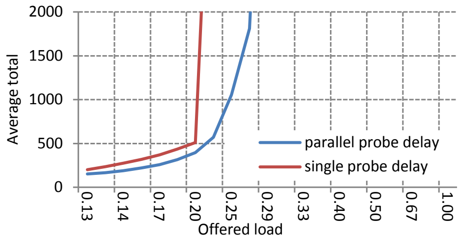


Fig. 9 Path set-up latency performance (8*8 mesh, lifetime 400)

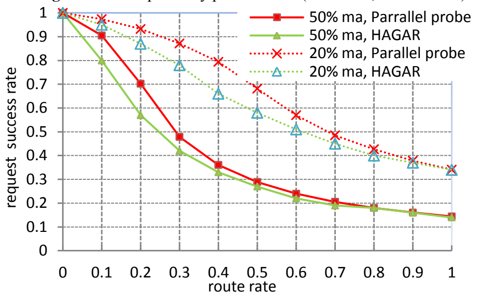


Fig. 10 Comparison of the different Path searching method, for lifetime 200 cycles at 16*16 mesh and master percentage 20% and %50

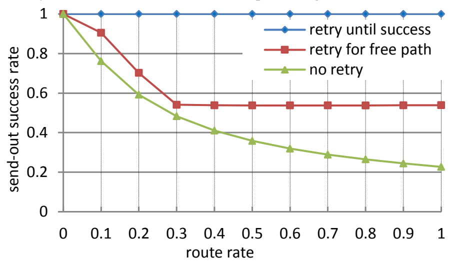


Fig. 11 success rate of the 3 policies of parallel probe searching method for a lifetime of 200 cycles with a 16*16 mesh and master percentage 50%

As mentioned, the *total delay* of a request is composed of delay of waiting in the queue and the *setup delay*, which extends from the first the time a request is sent out until the time the request is completed.

For retry-until-success policy, all delay types increase with the route rate without upper bounds. Some part of the average *total delay* and worst *total delay* curve is not shown in Fig. 12 when the delay was unbounded at that point. The delay in the queue keeps increasing and dominates at route rates above 0.2.

For the retry-for-free-path policy the average total delay and worst total delay also goes up with the route rates, and become unbounded above a route rate of 0.3. But the worst *setup delay* is bounded, which is $\alpha * n^2 * [3(2n-2)+6] = \alpha * 6N^{1.5}$ with $N=n^2$ being the number of nodes. For a 16*16 mesh with α (*master percentage*) %50, it is 12288 cycles. As shown in Fig. 12, the worst setup delay observed in our simulations is 2200 cycles. According to our experience, the theoretical worst case has a very low probability to occur.

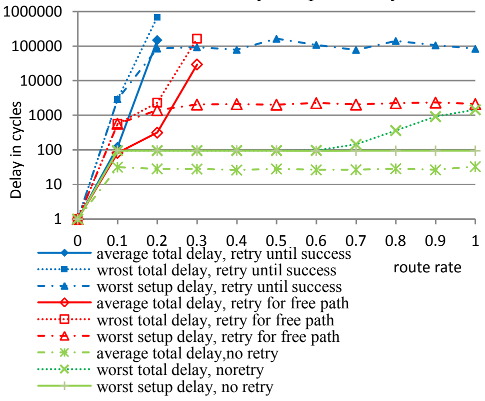


Fig. 12 Delay of the 3 policies of parallel probe searching method for lifetime 200 cycles at 16*16 mesh and master percentage 50%.

For the no-retry policy, the average total delay is around 36 cycles, and goes up slowly with the route rate. The worst case *setup delay* is also bounded. In theory it is 3*(2*n-2)+6=6n=96 cycles in a 16*16 mesh. In our simulations the worst case observed is exactly 96 cycles.

3.3. Synthesis results

Our switch has been synthesized by using Synopsys Design compiler with SMIC 90 nm library. The maximum clock frequency (probe clock) for the *control path* is 570MHz, the maximum clock frequency for *data path* is 1.8 GHz. This dual-clocking scheme has been well described and implemented in the work of Pham et al. [1]. It means that the switch operates at most at 570MHz during probe search stage. When the path has been established, it can use 1.8 GHz clock frequency to transfer source synchronized data [1]. In comparison, the sequential probe of [1] uses 0.18 um process and can work at 345MHz *control path* frequency and 923 MHz *data path* frequency. HAGAR [2] has been synthesized with FARADAY's 130nm UMC library and work at 200 MHz in an 8*8 mesh and at 50 MHz in a 16*16 mesh.

The area consumption for each switch node is 18733 NAND gates for a data path width of 64bits. Of that the *control path* consumes 10,364 NAND gates, and the *data path* consumes 8369 NAND gates. Hence, the per-bit area is 18733/64=292 gates. In comparison the sequential probe switch of [2] uses 12460/16=778 gates per bit.

Compared with the centralized solution with HAGAR, which has a payload of 68bit (32bit for address, 32 bit for data. 4bit for read/write/ mode), our solution is also better in terms of area, see Fig. 13.

4. CONCLUSION

We have proposed a circuit switched NoC with parallel probing, a parallel method for connection setup. Our simulation results demonstrate improvements in terms of setup delay, and success rate compared to previous work at comparable or reduced area.

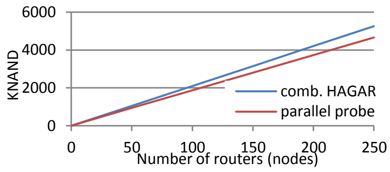


Fig. 13 Area consumption of all combined switches.

The special property of our switch is that the path search results can be acknowledged within a predictable, very low time limit under certain policies like retry-for-free-path or no-retry. In other words, worst case delay can be bounded in those policies. This property is important for real-time based applications. Hence, we have shown that parallel probing is an efficient and cost effective set-up procedure for circuit switched NoCs that can be used for dynamic circuit configuration in real-time and high performance applications.

However, due to the relative long setup time and the high resource usage during setup, it is only suitable for certain applications. For example, when life time of a path is short, a packet switching network will outperform our circuit switching. In future, we will do a comparison to identify the suitable application domain for circuit switched networks.

Furthermore, we plan to expand the flexibility of our method. . Currently entire links are reserved for a connection even if only a fraction of the link bandwidth is required. In future work we will consider the support for multiple sub-networks, which allows a connection to use only a fraction of a link.

5. REFERENCE

[1] P.-H. Pham, J. Park, P. Mau, C. Kim."Design and Implementation of Backtracking Wave-Pipeline Switch to Support Guaranteed Throughput in Network-on-Chip." *IEEE Trans. VLSI*, vol. 99, 2010.

[2] M. Winter and G.P. Fettweis "Guaranteed service virtual channel allocation in NoCs for run-time task scheduling." Design, Automation & Test in Europe Conference & Exhibition (DATE), Page 1-6, March 2011.

[3] D. Wiklund and L. Dake, "SoCBUS: Switched network on chip for hard real time embedded systems." In Proc. Int. Parallel Distrib. Process.Symp., 2003, p. 8.

[4] M. Millberg et al. "Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip." In Proc. of DATE, pages 890–895, February 2004.

[5] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: Concepts, architectures, and implementations." IEEE Des. Test. Comput., vol. 22, no. 5, pp. 414–421, 2005.

[6] A. Hansson, K. Goossens, and A. Radulescu. "A Unified Approach to Constrained Mapping and Routing on Network-on-Chip Architectures". In Proc. of 3rd Int. Conf. on HW/SW Codesign and System Synthesis, pages 75–80, 2005.

[7] J. Hu and R. Marculescu. "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints." In Proc. of DATE, pages 234–239, February 2004.

[8] M. Winter and G. Fettweis. "A Network-on-Chip Channel Allocator for Run-Time Task Scheduling in Multi-Processor System-on-Chips." In Proc. of 11th Euromicro Conference on Digital System Design (DSD), pages 133–140, September 2008.

[9] N. Ma, Z. Lu, L. Zheng "System design of full HD MVC decoding on mesh-based multicore NoCs." Journal Microprocessors & Microsystems Volume 35 Issue 2, March, 2011

Paper B

Parallel probe based dynamic connection setup in TDM NoCs

Shaoteng Liu, Axel Jantsch and Zhonghai Lu

Presented at the Design, Automation Test in Europe Conference Exhibition (DATE'14), Dresden, German, 2014. Included in the proceedings, pages: 239:1-239:6.

Parallel Probe Based Dynamic Connection Setup in TDM NoCs

Shaoteng Liu (liu2@kth.se), Axel Jantsch (axel@kth.se) and Zhonghai Lu (zhonghai@kth.se)
KTH Royal Institute of Technology

***Abstract*—We propose a Time-Division Multiplexing (TDM) based connection oriented NoC with a novel double time-wheel router architecture combined with a run-time parallel probing setup method. In comparison with traditional TDM connection setup methods, our design has the following advantages: (1) it allocates paths and time slots at run-time; (2) it is fast with predictable and bounded setup latency; (3) it avoids additional resources (no auxiliary network or central processor to find and manage connections); (4) it is fully distributed and therefore it scales nicely with network size.**

Compared to a packet based setup method, our probe based design can reduce path setup delay by 81% and increase network load by 110% in an 8x8 mesh, while avoiding the auxiliary network. Compared to a centralized method, our solution can double the success rate, while eliminating the central resource for path setup and reducing the wire overhead. Synthesis results suggest that our design is faster and smaller than all comparable solutions.

I. INTRODUCTION

Circuit switching (CS) is frequently adopted for guaranteed data transfer, since it is a cost-effective technique in real-time communication [1]. CS means that resources are allocated exclusively to a particular connection for its entire lifetime. Since the exclusive allocation of a link is very inflexible and potentially blocks other communications, in the on-chip context two main variants have been explored: (1) in *Time-Division-Multiplexing (TDM)* based CS, resources are allocated exclusively only in specific time slots, while the other time slots of a finite, repeating time window can be used by other communications [2]; (2) in *Spatial-Division-Multiplexing (SDM)*, only part of a link with its corresponding buffers are exclusively allocated to a connection, while the remaining wires of the link can be used by other communications [3]. Because SDM link sharing introduces large crossbars with low clock frequency and high hardware costs, TDM based CS is more popular and e.g. used in Æthereal [1], dAElite [2], Nostrum [4], and so forth.

A main challenge for TDM based CS NoCs is to set up a contention free path and to allocate the time slots. There are two categories of techniques for path setup: one is static scheduling [5]–[7], the other is dynamic (run-time) searching. Static methods schedule connections at compile time, based on the premise that all communication requirements are known beforehand. Thus, they are not well fit for applications like H.264 with requirements for dynamic communication setups or dynamic mixes of applications.

Thus, we concentrate on dynamic setup in TDM NoCs and propose a probe based dynamic path searching method with guaranteed setup delay. In particular, our contributions are:

- We propose a probe based setup method for TDM based CS NoC. Our method does not resort to an auxiliary network for connection configuration and imposes no limitation on search algorithms.
- We present a double orientation time-wheel technique to enable two-way communication in the probe based setup. A slot-table is shared by both forward and backward messages.
- We implement a parallel probing search to find a free path. This algorithm guarantees that, if a shortest path is available, it is found within $2 \times D + K + 6$ time slots, where D is the distance between source and destination, and K is the total number of time slots in a slot table.

Taking all together, our design provides shorter critical timing path, less wire overhead, shorter setup delay and higher success rate at lower hardware cost than any previously known method.

II. BACKGROUND AND RELATED WORK

Dynamic path searching methods can be divided into centralized [2], [8] and distributed solutions [9], [10]. In centralized solutions a special node is used to schedule all the connections in the network. This coordinating node can be a processor or a special hardware accelerator. Path scheduling algorithms running on hardware accelerators such as HAGAR [8] are about 100-1000 times faster than running as software on a processor [11], [12]. However, centralized setup methods suffer from the lack of scalability. As the network grows, the coordinator node becomes the bottleneck [8], [12]. Also, since retrying of failed requests causes the blockage of the following requests, failed setup requests are usually dropped in centralized setup methods. Furthermore, centralized solutions often depend on an additional network for delivering the configuration information to the routers.

Traditionally, distributed solutions in TDM NoCs are implemented with configuration packets [9]. Configuration packets such as setup, tear-down and Ack/Nack, require a separate *Best Effort (BE)* network during the connection establishment procedure. This approach suffers from three major drawbacks. Firstly, using an additional BE network for the connection setup is an unnecessary overhead. Secondly, the routing algorithms have to be deterministic to ensure setup, tear-down and Ack/Nack packets of a connection are on the same route so that the booked slots inside the routers along the route can be read/removed correctly, thus significantly restricting the path searching space. Thirdly, compared with our probing search, tear-down and Ack/Nack signals have to be sent in the form of packets. These packets are often underutilized and contend with setup and other packets. There is no delay guarantee for configuration packets, rendering the setup delay unpredictable.

Another kind of distributed path searching method is probe based searching, although it was only used in CS NoCs without TDM or SDM link sharing. For NoCs, the concept of probing was firstly proposed by Wiklund et al. [13]. Pham et al. [10] developed a backtracking path searching algorithm, which has better performance than Wiklund's. Liu et al [14] developed a parallel probing method for CS NoC. It can complete a search over all possible shortest paths within $O(D)$ time complexity where D is the geometric distance between source and destination. They demonstrated superior performance of this parallel

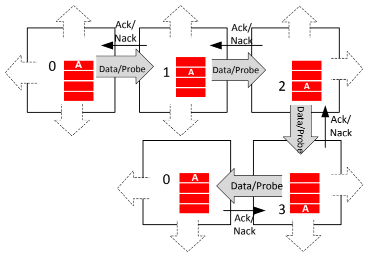


Fig. 1. The usage of double orientation time wheel

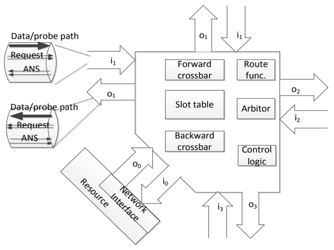


Fig. 2. Overview of the router

probing algorithm compared to Pham's backtracking algorithm. We adapt Liu's parallel probing algorithm for TDM based NoCs.

Another track of work uses per-connection *Virtual Channels (VCs)* and round-robin arbitration to share links and provide communication guarantees (e.g. [15], [16]). VCs are expensive resources, since they consist of buffers, multiplexers, demultiplexers and require separate flow control. The number of VCs per router per direction suggested by the authors is limited to 4, which limits the number of simultaneously supported connections. In section IV-C and table V we compare the hardware costs of artNoC [15] to our solution, showing better performance at half the area.

III. Motivation and Design Outline

A number of previous solutions rely on a BE NoC to support dynamic establishment of connections. For distributed dynamic solutions [5], [9], the BE NoC is not only used for delivering configuration packets, but also for path searching. Centralized dynamic solutions also depend on a BE NoC [8] for delivery of configuration information. Stefan et al. eliminate the BE NoC in dAElite but then they add a tree shaped dedicated configuration network instead [2]. In AElite certain time slots of a router are still reserved for configuration message delivery [6]. In addition, the configuration time of hundreds of cycles is very long.

An additional network for configuration is not cost-effective [1]. The to-be-allocated resources of the CS network have to be free at the time of path search and setup such that they can be used for the very task of connection setup. Besides, delivering all kinds of messages for connection establishment as BE packets is also inefficient and limits the selection of routing algorithm, as we mentioned in Section II.

To avoid these drawbacks, we propose a probe based solution for TDM NoC, where the probe searches through the network to find a free path, selects the time slots, and allocates the network resources for the required slots as it moves forward. When it arrives at the destination, the path is set up. As tear-down, Ack/Nack messages are tightly combined with the probe, our solution is not restricted to deterministic path searching algorithms. We use the parallel probing algorithm in [14] but apply it to TDM NoCs. The probe uses the same wires and buffers as the data uses after the path is set up.

In the following description a *channel* denotes a simplex link between two routers together with associated buffers in a particular time slot; hence, the same link in different time slots belong to different channels. A probe is sent out by a source node and travels through free channels, moving from one router to the next towards the destination. If a free channel is available, the probe will reserve the channel for future data transfer and go through the channel to the next router to continue the search. If at some point no free channel is found, the path search fails. Ack/Nack messages must be sent back to inform the source node whether a search failed or succeeded. Besides, Nack signals also tear down the reserved channels of a connection. To reduce wire costs, backward Ack/Nack messages are 1-2 bits. In probe based setup methods both forward (data/probe) and backward (Ack/Nack) signals are needed.

Backward Ack/Nack messages constitute a design challenge, since they associate to a connection, consist of only 1-2 bits, contain no address information and share wires in TDM manner. Thus, they must arrive at the right router in the right time slot and rely on the slot table's information of the router to move back towards the source. In figure 1 a connection spans 5 routers. For the forward data/probe path, the reserved time slots inside each router follow the sequence $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$. The slot table of each router records the crossbar configuration information. To ensure that backward Ack/Nack signals correctly read the information inside each slot table for traversal, the backward signals should be sent-out at slot 0 in router 5, then reach router 4 at slot 3, reach router 3 at slot 2, and so forth.

To address this challenge we introduce a double orientation time wheel. Inside each router, there are two slot counters, of which one is incremented and the other is decremented; both start from slot 0. The incrementing slot counter uses the slot table to configure the forward crossbar for data or probe; the decrementing counter configures the backward crossbar for Ack/Nack. In this way, if the Ack/Nack signal is sent out at the correct time slot, it will be correctly routed back to the source hop by hop.

IV. Router design and implementation

A. Control signals

The implementation uses a mesh topology with 5-port routers where each port consists of two physical links in opposite direction. Each link contains a data path, which is used for delivering the probe during the setup phase and for data transmission after a connection has been established. Every data path is coupled with 4 control bits: a 2-bit Request signal in parallel to the data path, and a 2-bit answer (ANS) signal in the opposite direction of the data path. Their usages are listed in tables I and II, respectively. Ack/Nack messages are carried by the ANS signal. In the data transfer phase the same ANS signal can be used for end-to-end flow control, as shown in table II.

TABLE I
The usage of Request signal

Request	Usage
00	Idle/data transfer
01	Unused
10	Probe comes in
11	tear-down established connections

TABLE II
THE USAGE OF ANS SIGNAL

ANS	Usage in setup	Usage in data transfer
00	Idle	Ready to receive data
01	Unused	Unused
10	Nack (Path search failed)	Unused
11	Ack (Path established)	Receiver buffer full

TABLE III
THE PROBE FORMAT

Lookahead routing (5 bits)	Dest.addr (6 bits)	Src.addr (6 bits)

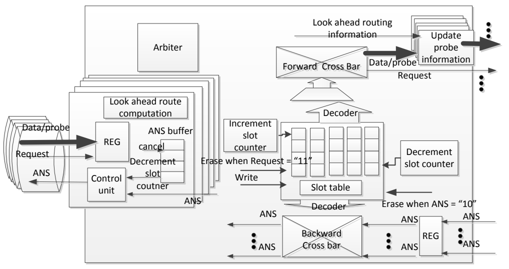


Fig. 3. Detailed router architecture

In total there are only 4 control wires for each channel, which, we believe, is the minimum overhead for probing based setup. The probe format is also compact. A probe just contains the destination address and look-ahead routing information. In an 8×8 mesh, the minimum width of a probe is 11 bits. Since the source node address is required by certain path searching algorithms (e.g. our parallel probing in this paper), a probe can become 17 bits in this case, as shown in table III.

B. Detailed router architecture

The slot-table structure is illustrated in figure 3. Rows in a slot-table represent time slots, and columns denote output links. The input channel id (In a 5 port router, it equals to $\lceil \log 2^5 \rceil = 3$ bits) is written into a vacant cell to book an output link for a certain slot. Each router has 5 outputs and thus the slot table width is 15 bits. This slot-table information, accessed by the incrementing slot counter, is used for forward crossbar configuration, while the decrementing slot counter is in charge of the backward direction. The cell content including its column number can be translated into two sets of configuration bits for the configuration of the forward crossbar and the backward crossbar, respectively. The translation logic is a simple logic decoder.

Inside each router, all input signals are latched. Then, the request signal is checked as follows:

"00" **data:** the data is directly forwarded according to the the corresponding slot table cell;

"11" **erase connection:** This signal will be firstly delivered through the crossbar according to the reserved slot table's information. Then, the corresponding slot table cell can be cleared at the beginning of the next cycle.

"10" **setup probe:** Based on the look-ahead routing information of the probe, arbitration for output channels commences. If a probe fails, backward ANS is used to notify the source node and cancel the reserved slot table cells hop by hop. We will explain this complicated process later. If a probe succeeds in acquiring one cell, it is delivered through the crossbar to the next router immediately. The slot table update is removed from the critical timing path and scheduled at the very beginning of the next cycle.

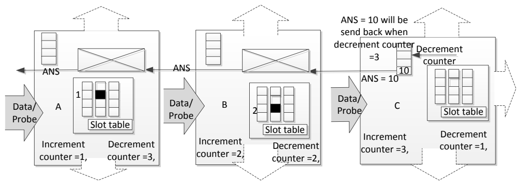


Fig. 4. ANS signal management

To reduce the critical timing path, look-ahead routing is used. The lookahead information denotes the desired output directions of the next router of a probe. This information is pre-computed in parallel with the arbitration and cross-bar traversal process of a probe. After the crossbar traversal, the probe will be updated and carry the new information before it reaches the next router.

The proposed router takes one slot per hop for all messages: request, probe/data as well as Ack/Nack. In our implementation, by default each time slot is one cycle.

Several aspects of our router deserve to be noticed.

1) Backward ANS signal management: We have mentioned in Section III that Ack/Nack signals must be sent out at the correct time slot. In the following we explain how to realize this scheme.

As figure 4 illustrates, a probe arrives when router C at incrementing slot counter is 3 and decrementing slot counter is 1. However, when it fails in router C, a signal ANS="10" will be sent back. The ANS should use slots $3 \rightarrow 2 \rightarrow 1$ to pass routers C, B, and A, respectively. To this end, ANS is buffered in router C for 2 slots, until the decrementing slot counter becomes 3.

We designed an ANS table at each input port for such buffering purpose. Each table cell is 2 bits wide. The rows represent time slots. The writing position of the table is pointed at by the incrementing slot counter; the reading position is pointed at by the decrementing slot counter. After a cell is read and ANS is sent back, it will be erased at the beginning of the next cycle. The maximum buffering time is K cycles, where K is the total slot number in the time window.

2) Predictable delay and setup polices: All kinds of message delays in our NoC are predictable, which are proportional to the distance between source and destination, with 1 cycle per hop (by default a slot corresponds to 1 cycle).

Moreover, the delay of a single search is also predictable. Suppose the distance between source and destination is D and the total slot number in the time window is K. Assume minimal routing. Then it takes at most $2D + K + 6$ cycles for the source to receive an Ack/Nack. D cycles are for sending the probe from source to destination, $D + K$ cycles for returning the ANS signal, and 6 cycles are consumed in the source and destination nodes.

We study two connection setup polices (adopted from [14]):

Retry until success: In this policy, the source node keeps retrying a request until it successfully sets up a connection. In this case the worst delay for setup is unbounded, because it is unknown when a free path becomes available.

Retry before deadline: In this policy, a deadline is attached to each setup request, which denotes the time when the connection has to be set up. The *residual time (RT)* is the deadline minus the current time. Since it can take $2D + K + 6$ cycles to set up a connection, a new connection is launched or an old is retried only when $RT > 2D + K + 6$. Otherwise, the request is discarded.

3) Path searching algorithms: Unlike other distributed path setup solutions [9], our probe based solution imposes no constraint on the routing algorithm. Hence, any probe searching algorithm with reasonable hardware cost is applicable. However, to achieve high performance we implemented the adaptive parallel probing algorithm proposed by Liu et al. [14]. Parallel probing searches all shortest paths between a given source and destination in parallel. If at least one shortest path is free it will be found and allocated in constant time. To achieve this, a probe is copied from the input of a router to up to two outputs if there are two productive directions towards the destination. In the process many parallel paths may be allocated by the travelling probes, but all of them except one will be de-allocated as quickly as possible.

However, we replaced Liu's complicated priority comparison arbitration mechanism with a smaller and faster round-robin arbitration. Since each probe may have two productive output directions and may book two slot cells, the corresponding 2-bits ANS cell is used to record the number of booked slots. Hence, the value of a cell will be decreased when a Nack signal from the downstream router is sent back. Thus, it requires that a cell can be written by both the incrementing and decrementing counter. Fortunately, our router operating mechanism can guarantee that there is no conflict. A Nack signal is only returned to its upstream router when its own ANS cell value becomes zero.

4) Timing, synchronization and scalability considerations: Our design minimizes the critical timing path. The critical path length consists of an input probe checking logic (4 gate delays), a round-robin arbiter (8 gate delays), a multiplexer (2 gate delays), a crossbar (4 gate delays), and a look-ahead routing information updating logic (2 gate delays). Thus, our router is very fast even though it only contains 1 pipeline stage.

Our design can also be applied in a mesochronous or asynchronous environment by adding synchronization tokens for slot update handshaking, similar to the technique used in Æthereal [1]. Our double time-wheel design does not impose any additional requirement on synchronization. If such synchronization efforts are required, we need to add 1 pipeline stage in the router to compensate the latency introduced by synchronization.

Considering hardware scalability, the main cost of adding a time slot is the increase in buffers. For each router, this will raise 25-bit buffer space in total (15 bits for the slot table, 10 bits for the 5 ANS tables in total). In our current design, we use registers for storage.

C. Implementation costs and comparison

The router synthesis results with TSMC 90nm technology is listed in table III. In our default settings, the effective link width for data is 32 bits, and the total link width is 36 bits. The additional wires are always 4 bits. This value does not increase with network size or the number of slots[1].

TABLE IV
THE ROUTER SYNTHESIS RESULTS WITH 90NM TECHNOLOGY

Slots in the time window	Critical path length (ns)	Area (um^2)	Power (mW)
1	0.7	8730	5.8
4	0.7	12226	6.8
16	0.7	22608	9.7

As table IV suggests, the router area goes up linearly with the slot number. One additional slot increases the area by 991 um^2.

We compare our synthesis results with other works reported in the literature. A conclusive comparison is difficult to perform because the different NoCs support different features. Nevertheless, we try to present a fair comparison based on available data.

The synthesis results of our router with TSMC 65nm, 90nm and 130nm, with different effective data path width and different slot numbers, are listed in table V and compared with others' results by assuming that all the routers are used for an 8×8 mesh. Generally speaking, our work has the shortest critical timing path[2]. Lusala's work [9] has the same critical timing path length but it is a mix of SDM channels and TDM channels. Its hardware cost is 5 times higher but offers more routing flexibility[3]. The area of our implementation is only slightly bigger than dAElite. However, dAElite [2] does not include the hardware cost for the scheduler. Finally, our work has the smallest additional wires per link for control/configuration[4].

TABLE V
COMPARISON WITH OTHER PUBLISHED DESIGNS BY USING THE SAME DATA PATH WIDTH AND SILICON TECHNOLOGY

	Critical path length (ns)	Area	Additional wires per link	Auxiliary network
Data width 16 bits, 130 nm technology				
artNoC [15] 2-flit buffers, 4 VCs	2	0.06mm^2	8	No
Our work 8-slot	1.3	0.03mm^2	5	No
Data width 48 bits, 65 nm technology				
Lusala [9] 3 SDM lanes, 3 time slots	0.5	0.05mm^2	20	Yes
Our work 9-slot	0.5	0.01mm^2	4	No
Data width 64 bits, 90 nm technology				
dAElite [2], 4 slots	above 1.08	0.016mm^2	10	Yes
Our work, 4-slot	0.7	0.017mm^2	4	No
Data width 68 bits, 130 nm technology				
HAGAR [8], 1 slot	2	20 kNand	5	Yes
Our work, 1-slot	1.3	5.5 kNand	4	No

V. PERFORMANCE EVALUATION

A. Experiment settings

Each resource node generates setup requests according to a Poisson distribution and pushes them into a queue. Uniform random, shuffle, and other traffic patterns are used for evaluation. An FSM pops a request from the queue and sends it out when an output slot is available. Then the FSM waits for a success or failure notification, upon which it either retries the request, discards it, or commences the data transfer. Any data point that is shown in the figures comes from simulation of 10 million cycles, of which the first 5% are discarded as a warm up period.

Several performance metrics are used:

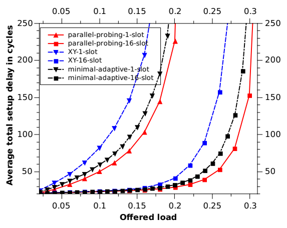
- Metrics for retry until success policy:
  total setup delay includes the *waiting time* for a request in the queue and the *setup delay* for a request, which extends from the first time sending the request until final success.

[1]Since the probe width is 17 bits, it requires that the forward data/probe path is at least 17-bit wide. However, if the data width used in data transfer is smaller than 17 bits, some wires of a link are inevitably wasted and thus regarded as additional wires.

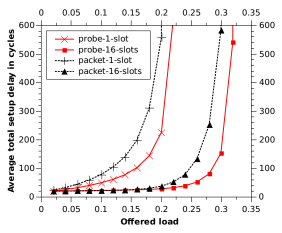
[2]The dAElite [2] [18] did not report its critical path for a 5-port router with 90 nm. However, the critical timing path of a 3 port router with 65nm is reported as 1.08 ns. Thus it is safe to deduce that with 90 nm, it is more than 1.08ns.

[3]In SDM any of the 4 lanes of an input port can be forwarded to any of the 4 lanes of an output port. But in our TDM scheme, one TDM time-slot can be forwarded only to the next time slot.

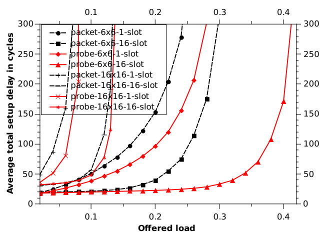
[4]From the implementation of HAGAR, we deduce that each link requires 2 bits to connect to the center node, 1 bit to distinguish the BE and GS packets, 1 bit to distinguish normal GS flits and link free signal, and 1 bit for stall/go flow control, so the additional wires are 5 bits/link in total.



(a) Comparison of path searching algorithms.



(b) Comparison with packet based setup in 8×8 mesh.



(c) Comparison with packet based setup in 6×6 and 16×16 meshes.

Fig. 5. Performance comparison under uniform traffic. Each connection delivers 100 flits.

offered load refers to the required data transfer per connection multiplied by the injection rate of setup requests. Suppose the injection rate is 1/2000 cycles, and each connection delivers 100 flits of data after setup, then the offered load is 100/2000=0.05 flits/cycle.

- Metrics for retry before deadline policy:
  request success rate denotes the ratio between established and desired paths and indicates the number of the requested paths could be established.
  master percentage denotes the percentage of nodes which can send out setup requests. These nodes are uniformly randomly distributed in the system.

B. Comparison of different path searching algorithms

Three path searching algorithms are implemented and compared, which are X-Y, minimal adaptive and parallel probing. In this experiment, the retry until success policy is applied. After a connection is established, 100 flits of data are delivered before the connection is released. E.g. when the *window size* (total slot number in the time window) is 1 (no TDM link sharing), it takes 100 cycles for data delivery; when the window size is 16 slots, it takes 1600 cycles. The actual time for data delivery equals to the required data transfer time multiplied by window size.

The results under uniform random traffic are shown in figure 5a, suggesting that parallel probing is the best path searching algorithm. E.g. at offered load 0.16 and when the window size is 1, the average setup delay of parallel probing is only 80% of minimal adaptive, and 50% of the X-Y algorithm. We also have evaluated algorithms with different window sizes, e.g. 16 slots. Their results suggest the same ranking. Consequently we choose parallel probing as the default path searching algorithm for the following experiments.

C. Comparison with distributed setup

We re-implement a packet based distributed path setup method according to Lusala's work [9] for comparison. Since with a BE packet based setup method, the message delay is unpredictable, we apply the retry until success policy.

The average total setup delay versus offered load in an 8×8 mesh under uniform random traffic is shown in figure 5b. We observe that our parallel probing method has shorter average setup delay than the packet setup method of [9]. E.g. when the window size is 16, at load 0.26, the average total delay of our probing method (refer to probe-16-slot) is 52 cycles, while the packet based method needs 134 cycles. Also, the saturation point of the network is 15-18% higher in our probe based solution, which translates into a correspondingly higher network utilization.

We also observe that increasing the number of slots in a time window can help to ameliorate both network utilization and setup delay, although the time required for data delivery and hardware costs increases.

Besides 8×8 NoC, we made comparison in different network sizes (6×6 and 16×16) and with different slot number (1 and 16), see figure 5c. We use solid, red lines to represent the setup delay of our method and dashed, black lines for the packet based method. The probe based setup has 30% to 80% lower delay and a 10% to 40% higher saturation point in this comparison.

In addition to uniform random, we use other traffic patterns for evaluation. As figure 6 shows, under shuffle traffic probe based setup has 81% delay reduction and an up to 110% higher saturation point. Under tornado traffic, we find upto 50% delay reduction and a 55% higher saturation point (not shown in the figure).

Thus, we conclude that our method offers better performance than the packet based method and without any auxiliary network. The drawbacks of the packet based method discussed in section II accounts for its inferiority.

D. Comparison with centralized setup

In this section, we will compare with two different dynamic centralized setup solutions.

1) Comparison with a centralized solution using hardware accelerator for path scheduling: We compare with HAGAR [8] by choosing the retry for deadline policy. We use *request success rate* versus *offered load*[5] as metric, since it has been reported in [8]. However, in [8], setup delay data is not reported. Actually the success rate should be related to the setup delay to make it a useful metric. Moreover, the limitation of [8] is that the supported window size is only 1.

We compare a 6×6 and a 16×16 network with 200 flits of data for each connection. The deadline is 200 cycles (just for setup not including data communication), which means the total setup delay of a successful request should be smaller than 200 cycles. Otherwise the request fails.

Figure 7 shows that the success rate of our method is better than HAGAR's. For example, in a 6×6 NoC at master percentage 50% and at offered load between 0.6 and 1.0, with 1 slot window size, our solution offers about 34% higher success rate; with 16 slots in total, this figure rises to 100%. In a 16×16 NoC, our design has even more advantages. At an offered load between 0.6 and 1.0, our solution offers 170% higher success rate with

[5]The metric *offered load* in this paper is the same as *route rate* in [8].

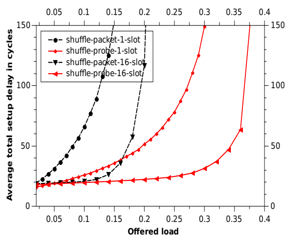


Fig. 6. Comparison with packet based setup in 8×8 mesh with shuffle traffic. Each connection delivers 100 flits.

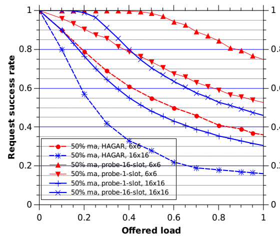


Fig. 7. Comparison with HARGAR [8] in 6×6 and 16×16 meshes. Each connection delivers 200 flits.

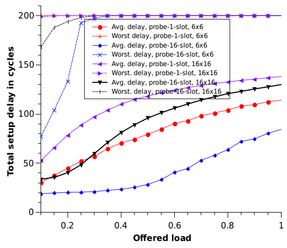


Fig. 8. Worst case and average delay of probe based setup with retry before deadline policy. Each connection delivers 200 flits.

16 slots in total. Again, increasing the window size enhances the setup performance.

The average and worst case total setup delay is reported in figure 8. The worst case delay (dashed lines) is always bounded and never exceeds 200 cycles, as required by the retry before deadline policy.

2) Comparison with a centralized solution using software based path scheduler: We compare with the software based centralized scheduler proposed by Stefan et al. [12], which can be used by dAElite [2]. Since the detailed delay analysis is not provided in that paper, we approximately calculate the expected delay figures based on the data given. The average time required for scheduling a path grows linearly with the distance between source and destination and for a distance of 4 hops it takes about 1200 cycles [12]. Thus, the sum of the request arrival rates of all the nodes together should not exceed 1/1200 per cycle, otherwise the scheduler is overloaded and requests have to be discarded.

Thus, let us ignore the path configuration and message communication delay and make an estimation. In a 6×6 network, with uniform random traffic, the average distance is 4 hops. Suppose each connection delivers 200 flits, then at offered load 0.1 the injection rate of requests per node per cycle is about 1/2000. At master percentage 50%, the total injection rate of all the nodes is $36/2000 * 0.5 = 9/1000$, which exceeds the capability of the scheduler. Even if all served requests can successfully find a path, the success rate is still less than $1/1200 \div 9/1000 \approx 9.3\%$, much less than in our work and HAGAR's (both above 90%). If path configuration and message communication delay is added, and considering the fact that not all served requests can succeed in finding a path, the success rate is even lower.

VI. Conclusion and Future Work

We have proposed a TDM circuit switching NoC with a parallel probing setup method by developing a double time-wheel technique. Our simulation results demonstrate that our solution is superior in terms of setup delay, network performance and hardware cost to all previously reported comparable solutions. The main reasons are due to

- that we use a distributed setup method that scales well with network size;
- that we propose a double time-wheel router with a parallel path search that searches effectively through all the possible shortest paths in parallel;
- that we use the available network resources also for path search, setup and configuration, thus incurring minimal extra hardware cost.

Although in our experiments each connection uses only one of the TDM slots of a time window, our design can support connections with multiple slots. This can be realized by sending out multiple requests for one connection, of which each request will build a path with one slot. However, this is inelegant and raises complications due to massive contentions. Therefore, we will tackle this problem next by developing a sophisticated technique for allocating multiple slots within a time window for a connection.

References

[1] K. Goossens and A. Hansson, "The Æthereal network on chip after ten years: Goals, evolution, lessons, and future," in *DAC*, 2010

[2] R. Stefan, A. Molnos, and K. Goossens, "dAElite: a TDM NoC supporting QoS, multicast, and fast connection set-up," *IEEE Transactions on Computers*, vol. PP, no. 99, p. 1, 2012.

[3] A. Banerjee, P. Wolkotte, R. Mullins, S. Moore, and G. J. M. Smit, "An energy and performance exploration of network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, no. 3, pp. 319–329, 2009.

[4] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *DATE*, 2004.

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

[6] A. Hansson, M. Subburaman, and K. Goossens, "Aelite: A flit-synchronous network on chip with composable and predictable services," in *DATE*, 2009.

[7] J. W. Lee, M. C. Ng, and K. Asanovi, "Globally synchronized frames for guaranteed quality-of-service in on-chip networks," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1401–1411, 2012.

[8] M. Winter and G. Fettweis, "Guaranteed service virtual channel allocation in NoCs for run-time task scheduling," in *DATE*, 2011.

[9] A. K. Lusala and J.-D. Legat, "Combining SDM-Based circuit switching with packet switching in a router for on-chip networks," *International Journal of Reconfigurable Computing*, vol. 2012, pp. 1–16, 2012.

[10] P.-H. Pham, P. Mau, J. Kim, and C. Kim, "An on-chip network fabric supporting coarse-grained processor array," *IEEE Transactions on Very Large Scale Integration Systems*, vol. PP, no. 99, pp. 1 –5, 2012.

[11] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, and H. Corporaal, "Dynamic time-slot allocation for QoS enabled networks on chip," in *3rd Workshop on Embedded Systems for Real-Time Multimedia*, 2005.

[12] R. Stefan, A. B. Nejad, and K. Goossens, "Online allocation for contention-free-routing NoCs," in *ACM Proceedings of Interconnection Network Architecture: On-Chip, Multi-Chip Workshop*, 2012.

[13] D. Wiklund and D. Liu, "SoCbus: switched network on chip for hard real time embedded systems," in *IEEE Parallel and Distributed Processing Symposium*, 2003.

[14] S. Liu, A. Jantsch, and Z. Lu, "Parallel probing: Dynamic and constant time setup procedure in circuit switching NoC," in *DATE*,2012.

[15] C. Schuck, S. Lamparth, and J. Becker, "artNoC - a novel multi-functional router architecture for organic computing," in *FPL*, 2007.

[16] N. Kavaldjiev, G. J. Smit, P. G. Jansen, and P. T. Wolkotte, "A virtual channel network-on-chip for GT and BE traffic," in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, 2006.

[17] E. Mensink, D. Schinkel, E. A. Klumperink, E. van Tuijl, and B. Nauta, "Power efficient gigabit communication over capacitively driven rc-limited on-chip interconnects," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 2, pp. 447–457, 2010.

[18] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens, "A TDM NoC supporting QoS, multicast, and fast connection set-up," in *DATE*, 2012.

Paper C

Highway in TDM NoC

Shaoteng Liu, Zhonghai Lu and Axel Jantsch

Presented at the ACM/IEEE International Symposium on Network-on-Chip (NoCS'15), Vancouver, Canada, 2015. Won the **best paper award**. Included in the proceedings, pages: 15:1-15:8.

Highway in TDM NoCs

Shaoteng Liu
liu2@kth.se
KTH Royal Institute of Technology

Zhonghai Lu
zhonghai@kth.se
KTH Royal Institute of Technology

Axel Jantsch
axel.jantsch@tuwien.ac.at
Vienna University of Technology, Austria

***Abstract*—TDM (Time Division Multiplexing) is a well-known technique to provide QoS guarantees in NoCs. However, unused time slots commonly exist in TDM NoCs. In the paper, we propose a TDM highway technique which can enhance the slot utilization of TDM NoCs. A TDM highway is an express TDM connection composed of special buffer queues, called highway channels (HWCs). It can enhance the throughput and reduce data transfer delay of the connection, while keeping the quality of service (QoS) guarantee on minimum bandwidth and in-order packet delivery. We have developed a dynamic and repetitive highway setup policy which has no dependency on particular TDM NoC techniques and no overhead on traffic flows. As a result, highways can be efficiently established and utilized in various TDM NoCs.**

According to our experiments, compared to a traditional TDM NoC, adding one HWC with two buffers to every input port of routers in an 8×8 mesh can reduce data delay by up to 80% and increase the maximum throughput by up to 310%. More improvements can be achieved by adding more HWCs per input per router, or more buffers per HWC. We also use a set of MPSoC application benchmarks to evaluate our highway technique. The experiment results suggest that with highway, we can reduce application run time up to 51%.

I. Introduction

Time Division Multiplexing (TDM) technique is frequently used for guaranteed data transfer in NoCs [1]–[5]. TDM NoC means that a physical link can be shared by different connections, with each connection allocated one or several specific time slots in a finite repeating time window. A connection can span many links from source to destination, by allocating slot(s) at each of the links in a consecutive manner. As illustrated in Fig. 1, connection $v1$ passes link 1 and link 2. If slot 0 and slot 2 of link 1 is allocated to $v1$, then slot 1 and slot 3 of link 2 must be allocated to $v1$. Once a TDM connection is established, packet delivery on the connection is free from contention. It can therefore provide hard guarantees on delay, throughput and in-order delivery. However, in TDM NoCs, quite often the TDM slot utilization is low due to *unused slots* including both unallocated and idle slots.

Firstly, *unallocated slots* commonly exist inside a TDM NoC. TDM NoC requires that reserved slots on the links of a connection must follow a consecutive sequence. Such mandatory sequence makes it difficult to utilize all the slots of links. For example, suppose there is a connection $v4$ wants to use link 3 and link 2 in Fig. 1, although both link 3 and link 2 have unallocated slots, they cannot be allocated to $v4$ since they are not consecutive. Also, when mapping an application onto a TDM NoC, it is common that some links have more bandwidth reservation requirements and some links less. Such unbalanced bandwidth requirements inside a network also cause slots of some links unallocated.

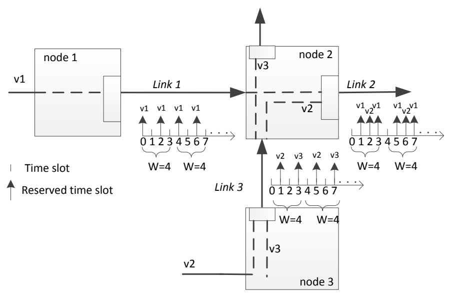


Fig. 1. Illustration of connections in TDM NoC

Secondly, *idle slots* are common for a connection with dynamic fluctuation of network traffic. Idle TDM connections withhold all the pre-reserved slots even if they have no data to deliver. For example, as shown by Fig. 1, suppose connection $v2$ becomes idle, it still occupies one slot on link 1 and link 2, respectively. Also, busy TDM connections can just use their pre-reserved slots for data transfer. For example, no matter how many data flits are waiting on connection $v1$, connection $v1$ can still just use the two reserved time slots in a time window, even if there are free slots available along link 1 and link 2.

In the paper, we develop a new technique called *highway* which can enhance the performance of TDM NoCs by utilizing free slots. With this technique, a TDM connection can dynamically acquire both idle and unallocated time slots to enhance its throughput, while keeping its QoS guarantee on minimum bandwidth and packet order. In particular, our contributions are:

- We develop the concept of highway to efficiently use time slots, which is applicable to many kinds of TDM NoCs.
- We propose a distributed, run-time highway setup and reclaim policy. Whenever the necessary resources for building a highway become available, a TDM connection can make use of it.
- We have made an efficient implementation of our pro-

NOCS '15, September 28 - 30, 2015, Vancouver, BC, Canada 2015 ACM. ISBN 978-1-4503-3396-2/15/09$15.00
DOI: http://dx.doi.org/10.1145/2786572.2786577

posed technique. By taking advantage of the contention-free property of TDM NoCs, we can set up a highway with $2D + 2$ cycles and without head flits, where D is the distance between source and destination. Besides, our HighWay Channel (HWC) allocation only needs a very simple allocator, since our highway setup method promises contention free. Thus, our hardware implementation has a relatively short critical timing path.

- We evaluated our highway technique using both synthetic traffic and application benchmarks and suggest how to efficiently use highways.

II. Related Work

Goossens et al. [9]. tried to increase the slot utilization of TDM NoC by introducing best effort (BE) traffic into a TDM NoC (Æthereal), free slots can be utilized to deliver best effort packets. However, the main issue with this mixed guaranteed and BE traffic scheme is that it may cause disorder of arrival packets, as observed in [6]. Eg., suppose a sender sends a BE packet first and sends a guaranteed packet second. At the receiver side, it is possible that the guaranteed packet arrives before the BE packet. This is due to that the transfer delay of a guaranteed packet is bounded, whereas a BE packet has no QoS guarantee. Besides, this solution is not cost-effective, as observed by Goossens et al. themselves [1]. The cost for supporting BE traffic is relatively high due to virtual channels and their allocation. But the service given to BE traffic is low. Since TDM NoC has to prioritize guaranteed traffic, the BE packets can be blocked for a very long time.

To solve the packet disorder problem as in Æthereal, Marescaux et al. [6] proposed a source routing based TDM NoC which provides a new QoS class called SuperGT to increase the slot utilization while maintaining packet order. However, this method suffers from several limitations. Firstly, it is tightly coupled to a source routing based TDM NoC. It forces the traffic flow of a connection to be divided into small packets. The packet size is limited by the number of reserved slots of the connection in a time window. Besides, each packet must have a head flit, since the head flit's information is needed for source routing, virtual channel setup, connection identification, and packet order maintenance. Secondly, it requires that each TDM connection must have at least two reserved TDM slots in a time window, and all reserved slots must be adjacent. Otherwise, the superGT technique cannot be utilized. Eg., connection $v2$ and $v3$ in Fig. 1 cannot use superGT because they only reserve one slot per time window. Connection $v1$ in Fig. 1 cannot use the superGT technique as well, because its two reserved slots are not adjacent. Because of these limitations, superGT can only be utilized in restrictive situations. Moreover, this technique always wastes bandwidth for head flit delivery, due to the limitation on packet size. As an extreme case, suppose a connection has two reserved slots in a time window, the packet size of this connection is limited to 2 flits, which means 50% of the throughput has to be wasted on the head flits.

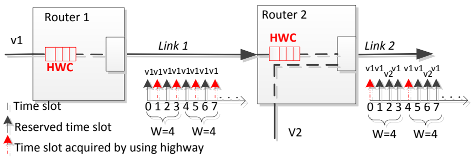


Fig. 2. The function of highway

In contrast, our highway technique can make use of unused TDM slots while keeping the packet order and minimum throughput guarantee. It needs no head flits, has no limitation on packet size, introduces no interference to normal TDM traffic flows, puts no constraint on TDM slot allocation and configuration method, and does not rely on particular routing mechanism or router architecture. As a result, our technique has no architecture dependency and thus can bring benefits to many different kinds TDM NoCs.

III. Highway Concept and Design Considerations

A. Additional Motivation

As we analysed in Section I, low utilization due to unused TDM slots is a common problem in TDM NoCs. Besides, this technique has to be leveraged in order to suit the needs of dynamic and mixed traffic scenarios. Consider the following practical situations: 1) Dynamic traffic: For streaming applications [7], [8], e.g., H.264 decoding, we just need the NoC's promise on the lower bound communication bandwidth. The upper bound of a traffic flow can be dynamically and readily adjusted by applying a flow control mechanism or adding flow regulation components. 2) Mixed traffic: As demonstrated in [6], consider a system with a guaranteed connection between a L1 cache and L2 cache. Since cache misses are not completely predictable, it is common practice to over-allocate bandwidth. Thus we might want a mechanism which can keep the guarantee on minimum throughput and on the predictable traffic flow, while offering additional non-guaranteed bandwidth to enhance the overall system performance or absorb peaks of less predictable traffic flows.

B. The Concept of Highway

Our proposed highway is an express path for a TDM connection. Based on an established TDM connection, it can speed up data transfer by using unused time slots along the links of the connection. This is made possible because a highway consists of one buffer queue at the input port per router. Arbitration is performed to use the free time slots of the output link of a router. We name these buffer queues as *highway channels (HWCs)*.

The function of a TDM highway is illustrated in Fig. 2. The two connections, $v1$ and $v2$ share link 2 in such a way that $v1$ reserves two slots (slot 1 and 3) of link2; $v2$ reserves one slot (slot 2) of link 2. Then, $v1$ also builds up a highway path by occupying one HWC in router 1 and one HWC in router 2. Thus, $v1$ can additionally use slots of link 1 and link 2 whenever they are free. For example, it can acquire slot 1 and 3 on link 1 and slot 0 on link 2. It can also acquire the slot 2 of link 2, if $v2$ does not use it. In contrast, connection $v2$

has no HWC and thus can only use the reserved one slot for data delivery.

A HWC functions like an input queue: if the output link of a router is occupied, incoming flits of a connection with a highway will be buffered in the HWC. When the output link becomes free, or a reserved slot of the connection is coming, the output link can immediately serve the HWC.

C. Design Considerations

The idea of our highway technique sounds simple. However, the details are complicated, especially when we try to make it commonly used for TDM NoCs. The principles and considerations are listed as follows:

1) With or without a highway, a TDM connection is promised to use its pre-reserved slots to offer a guaranteed service. Due to this principle, if a highway is used for a TDM connection, whenever any of the pre-reserved slots arrives, the queue of the corresponding HWC will be served. For example, as Fig. 2 suggests, connection v1 has an established highway. Therefore, at slot 1 or 3, its HWC in router 2 is guaranteed to be served. This is in contrast to normal virtual channels which have no service guarantee. Moreover, this principle also requires that the dynamic highway setup/release process should incur no additional traffic to the guaranteed traffic flow.
2) We must guarantee that reordering of a traffic flow never happens in any situation. This rule sounds easy but it is tricky to follow. In Section IV-D, we will show our solution to keep flits ordered during the highway release process.
3) A highway accelerates data transfer only if one HWC per router is allocated along the path of a connection, all the way from the source to the destination. This is due to that individual HWCs cannot ameliorate performance. For example, suppose connection v1 in Fig. 2 gets a HWC in router 1 but fails to get a HWC in router 2, then link 2 will be the throughput bottleneck, since still only the two reserved slots of link 2 can be used. Thus, during the highway setup process, our HWC allocation follows a *win all or nothing* principle that, if we fail in allocating one HWC in any of the routers along a connection, all allocated HWCs are canceled as soon as possible.

IV. Highway in TDM NoCs

A. Design Overview

An overview of a TDM router with highway is depicted in Fig. 3. Each input link has an input manager, which manages all the incoming flits of that input. We can have one or several HWCs per input manager. With more than one HWC, the input manager needs additional logic for internal arbitration between the HWCs. To support our highway technique, each input /output link needs to have a $flag$ signal and a $credit$ signal coupled with the data path. The $flag$ signal is in parallel to the data path. It is used for highway setup, transfer and release. The $credit$ signal goes in the opposite direction of

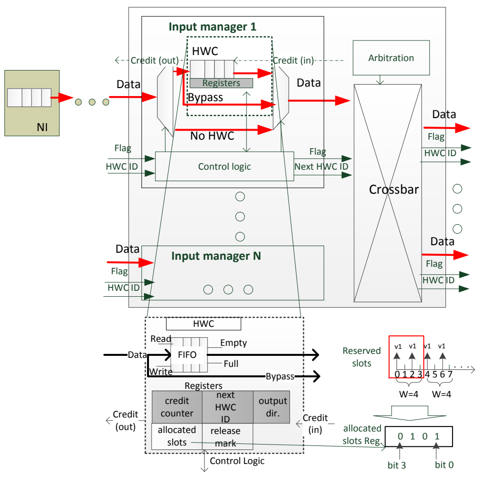


Fig. 3. The micro-architecture of a TDM router with highway

the data path. It is used for flow control in the data transfer phase and for Ack/Nack purpose in the highway setup phase. As Tables I and II suggest, we use 3 bits for the $flag$ signal and 2 bits for the $credit$ signal.

Because of the design considerations listed in Section III-C, we do not use head flits in our highway technique. Instead, we use the 3-bit flag signal for highway setup and release purpose. Unlike the information bits of a flit, a flag associated to a flit can be changed during the transfer process. It does not need to be buffered together with the flit. Instead, it is regenerated when a flit leaves a router, based on the router's current status.

Inside each HWC, there is a set of registers. The "next HWC Id" register stores the id of the HWC in the next router[1]. The "out dir." register stores the output direction to reach the the next router. The credit counter records the available queue size of the downstream HWC. All of these 3 registers are commonly used in virtual channel techniques. The "release mark" register is used in our highway release process. The "allocated slot" register uses a vector of bits to mark which are the reserved slots in a time window of a connection. The size of the bit vector equals the time window size. For example, as Fig. 3 describes, if the reserved slots for a connection are slot 0 and slot 2 inside a time window of size 4, the "allocated slot" register is configured as "0101".

In our design, HWCs are dynamically allocated and reclaimed for connections. Note that, a HWC can only be assigned to one connection at a time.

When a TDM connection has a certain amount of data buffered in the network interface, it may set up a highway for acceleration. The general operation for using a highway in a TDM NoC consists three phases, namely, *setup*, *data transfer*, and *release*.

[1]We do not need this register if each input manager only has one HWC.

TABLE I
USAGE OF THE FLAG SIGNAL

Flag	message	Usage
000	Idle	–
001	Setup	try to book a HWC
010	HWC-RS	Incoming flit has a HWC and sent/received at a reserved slot (RS)
011	HWC-NS	Incoming flit has a HWC and sent/received at a non-reserved slot (NS)
100	No HWC	Incoming flit has no HWC
101	Release	Release the allocated HWC

TABLE II
USAGE OF THE CREDIT SIGNAL

Credit	Usage during highway setup	Usage after highway is built
00	Idle	–
01	–	credit updating
10	Nack (highway setup failed)	–
11	Ack (highway setup success)	–

B. Highway Setup Phase

Fig. 4 illustrates the setup process for a 2-hop TDM connection which has 2 reserved slots inside a time window of 4. From the source node, when a reserved slot (slot 0) is coming, a data flit is sent out with $flag$ signal "setup". This flit travels by using reserved slots of links, at a constant speed of one slot per hop. If it can acquire a HWC when arriving at a router during the traversal, its flag remains "setup". When the destination node receives a flit with flag "setup", it will send back an "Ack" message by using the $credit$ signal of the allocated HWC in the destination node. Such a message will be delivered backward to the source node hop by hop, through the $credit$ signal of each allocated HWC along the forward path of the flit. When the source node receives "Ack" , it can begin to use the established highway for data transfer.

The following is worth mentioning.

1) The HWC allocation performs one hop in advance. For example, as suggested by Fig. 2, router 1 is responsible to allocate the HWCs at the input port of router 2. The allocation decision is put into "next HWC Id" reg. This technique is inherited from the virtual channel allocation technique [12].
2) If the flit with flag "setup" fails to acquire a HWC in a router, the highway setup process stops. The flit continues its traversal with $flag$ changed into "No HWC". Meanwhile, a "Nack" signal will be sent back towards the source node hop by hop through the $credit$ signals of the allocated HWCs in routers. As the "Nack" signal travels, it will release the allocated HWCs.
3) Since "setup" flits are delivered without contention by pre-reserved slots of links, arbitration is not needed in the setup phase. Thus, we can simplify the logic for the HWC allocation inside each input manager. Moreover, the highway setup time is predictable. It is $2D + 2$ cycles (assuming each slot is one cycle), where D is the distance between source and destination.
4) The whole setup process just utilizes $flag$ and $credit$ signals, it generates no additional traffic flow overhead.
5) Our highway setup method is architecture independent. We do not care about how flits of a connection are routed or how connections are established. As long as a flit is delivered without contention to the destination with a $flag$ as "setup", a highway has been established [2].

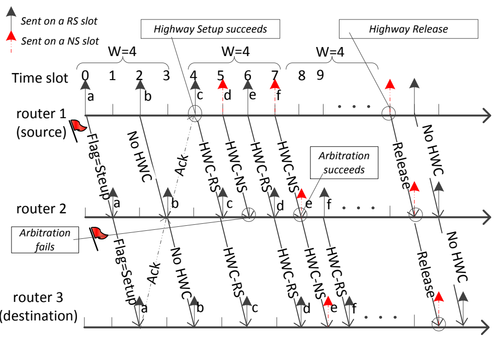


Fig. 4. Highway setup, data transfer, and release process. With highway, the RSs (reserved slots) on a link are still guaranteed to use, while a NS (non-reserved slot) can be acquired by wining an arbitration.

The highway configuration process configures the registers inside each allocated HWC. During the setup phase, the flit with flag "setup" configures the register "next HWC Id" and "out dir." The "allocated slots" register configuration takes place in data transfer phase.

In the case of failed highway setup due to HWC unavailability, we use a simple retry scheme until success.

C. Highway Data Transfer Phase

After a highway is established, flits sent at a reserved slot have the flag "HWC-RS", whereas flits sent at non-reserved slots have the flag "HWC-NS", as illustrated in Fig. 4. Arbitration for a non-reserved slot is needed, if it is requested by multiple input managers.

The "allocated slots" register of a HWC is configured by multiple "HWC-RS" flits arrived after the highway setup process, if the connection has multiple reserved slots. When a flit with flag "HWC-RS" arrives at a HWC, it will set the corresponding bit of the HWC's "allocated slots" register according to the ID of the current slot, if the bit is unset ("0" means unset and "1" set)[3]. For example, if a "HWC-RS" flit arrives at a HWC on slot 0, it will set the bit 0 of the register.

The function of a HWC in the data transfer phase is somehow similar to input queue based virtual channel. Incoming data flit of a HWC is pushed into a queue according to the flit's $HWC\ ID$. If the desired output channel is granted, one flit will be popped from the queue and sent with a valid $HWC\ ID$ pointing to the HWC in the downstream.

One specialty is that, when a reserved slot of an HWC comes, the queue of the HWC is guaranteed to be served. Besides, the transfer of "HWC" flits does not interfere with "No HWC" flits, since our arbitration mechanism prioritizes the data flit sent by using reserved slots.

[2]In source routing based TDM NoCs, we need to add a slot pointer inside each router to obtain the ID of the current slot for the "allocated slots" register configuration.

[3]Due to slots are reserved consecutively along the links of a connection, when an HWC receives a "HWC-RS" flit from its upstream node, it means its own reserved slot is met.

A FIFO is implemented in an HWC for flit buffering. However, we also add a bypass way to the FIFO. If the FIFO is empty and the desired output is ready, an incoming flit can be directly forwarded, without being buffered in the FIFO.

The credit based flow control policy used in our HWCs is similar to that in Chapter 13.3.1 of [11], except that sending data flit at reserved slots neither consumes credits nor generates credit updating signal to the upstream.

D. Highway Release Phase

If a connection no longer requires a highway, it should release all the occupied HWCs.

To release a highway, the source node sends out a "release" flag. If a HWC is empty when a "release" flag arrives, the "release" flag will reset the HWC and get forwarded to the downstream node. However, if a HWC still has buffered data, the "release" flag will set the "release mark" register and halt its forwarding until the HWC becomes empty.

As mentioned in Section III-C, it is tricky to maintain flit order of a connection during its highway release process. Let's consider the following situation: Suppose the source node of a connection has sent out a "release" flag. After that, the source node continues to send flits with the flag "No HWC". The HWC release process is relatively slow, since the "release" flag may wait inside a HWC until it becomes empty. However, the flit sent by the source node with the flag "No HWC" travels at a guaranteed speed of one slot per hop. Therefore, a "No HWC" flit of a connection may arrive at a router which still has an unreleased HWC for that connection. In this situation, this "No HWC" flit should go into the unreleased HWC to maintain the flit order. However, since all the upstream HWCs have been released, this flit arrives without a valid $HWC\ ID$. How can we find the HWC for this flit?

Unlike [6], we do not use a head flit for carrying the connection ID. Instead, we use the reserved slot information for connection identification. Let us also consider the following facts:

1) "No HWC" flits are delivered by using reserved consecutive slots of links.
2) "allocated slots" register of a HWC can be used to claim which slot is the reserved slots on a link of a connection.

Therefore, when a "No HWC" flit arrives at an input manager of a router, normally it will be directly forwarded to an output. However, if a HWC at the same input manager also claims that it meets a reserved slot and must be served, the incoming flit and this HWC must belong to the same connection and thus the incoming flit should go into the HWC. In this way, we can identify whether or not a "No HWC" flit has an unreleased HWC.

E. Implementation Cost

Our highway TDM router is built on top of a base TDM router (similar to [2]) used for mesh topology. The additional components are 1) one input manager per input port, 2) an external arbitrator for the arbitration between all the input managers. Note the buffer size per HWC should be large enough to cover the round-trip delay of credit updating (see Chapter 13.3.1 of [11]). Thus, suppose that each slot is one clock cycle, and the credit-updating signal is one cycle per hop. Depending on the implementation details, the round-trip delay is at least two cycles[4]. Thus, the minimum buffer requirement is two flits. We evaluate the additional costs in this section. The synthesis results are reported under 45 nm technology.

TABLE III
SYNTHESIS RESULTS WITH 45 nm TECHNOLOGY WITH FLIT WIDTH OF 128-BIT AND A BUFFER SIZE OF 2 FLITS

Component	Comb.	Non-comb.	Flip-flops	Total (um^2)	Total (gates)
HWC	298	1588	282	1886	2694
Other logic	457	28	5	485	693
Input manager total	755	1616	287	2371	3387

The area costs of an input manager containing one HWC with 2 buffers and with a data flit width of 128-bit and 16-slot window size are listed in table III. In our implementation, each slot represents one clock cycle. As Table III shows, the total area cost of a HWC manager is 2371 um^2, among which 1886 um^2 is spent on the HWC. Inside a HWC, FIFO constitutes about 92% of the area. In our implementation, FIFO is built by using flit-flops. If we use SRAM cells, it can be a factor of 3 or 4 less expensive. The cost of the external arbitrator is only 161 um^2. It does not scale up with the HWCs per input.

The critical timing path of a router consists of 3 components: an input manager, the external arbiter and the crossbar. The latter two components contribute a latency of 0.15 ns. The latency of a input manger varies with the number of HWCs it has. E.g., with one HWC, its latency is 0.36 ns; with 8 HWCs, its latency is 0.54 ns.

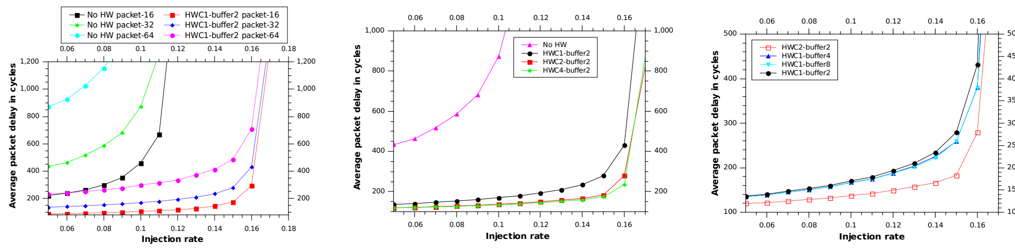
Compared to the base TDM router, our highway TDM router (adding 1 HWC with 2 buffers per inport) causes an area overhead of 12016 um^2. This is the cost of leveraging the static inefficient QoS guarantees.

There are also additional costs on the network interface (NI) when applying the highway technique. We need a state machine to control the setup/release of highways, as well as credit based flow control logic for data transfer. If a NI uses more than one highway, arbitration logic is also needed. However, since we do not need to increase the flit buffers inside a NI, the additional costs in total are still relatively small.

V. Performance experiments and results

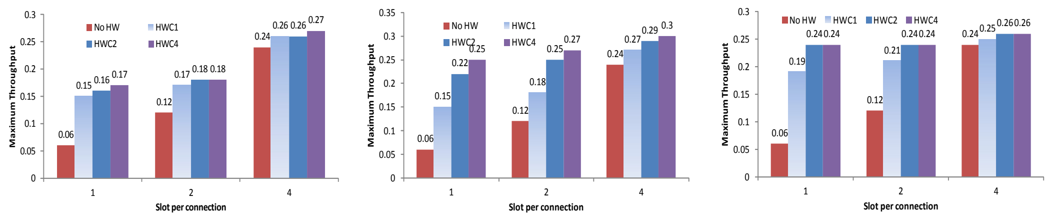
We present experiments and results concerning the benefits of using highways in different configurations, under different test scenarios, and with different TDM NoC technologies. We utilize a popular mesh topology in our evaluation. To facilitate our evaluation, we assume that each slot is one clock cycle, and each HWC should have more than 2 buffers to cover the credit round-trip delay. The performance enhancements brought about by our highway technique are evaluated with both synthetic traffic patterns and application benchmarks.

[4]In this case, the credit updating signal needs to be sent out as soon as the arbitration succeeds.



(a) Packet sizes (b) Increase HWCs (c) Increase buffers

Fig. 5. Performance evaluation under uniform traffic in an 8x8 mesh, with a window size of 16 slots. Each connection reserves 2 slots in a window. The packet size is 32 flits for (b) and (c). Injection rate is in flit/connection/cycle (fcc)



(a) Uniform random (b) Tornado (c) Shuffle

Fig. 6. Throughput evaluation under different traffic patterns and different connection width and a packet size of 32 flits

A. Performance Evaluation with Synthetic Traffic

We assume that the static TDM NoC has an architecture similar to [2]. We designed a depth first searching algorithm similar to [10] to do path searching and slot allocation for connections. After all the connections are configured, we can launch our NoC for data transfer by using these statically reserved connections.

For a uniform random traffic pattern, we randomly generated 64 connections and scheduled them in an 8×8 mesh with a window size of 16 slots. If a generated connection cannot be scheduled, we will regenerate it until it can be scheduled. We also use the Tornado and Shuffle traffic patterns for which connections are generated according to the relationship of source and destination nodes as described in the Chapter 3.2 of [11].

Packets generated for each connection obey a Poison distribution. Therefore, the traffic injection rate of a flow can be controlled by adjusting the inter arrival time between packets, while the burstiness of the traffic flow can be controlled by varying the number of flits in a packet. We vary the packet size from 16 to 64 flits, the reserved number of slots per connection per time window from 1 to 4 slots, the HWC per input from 1 to 4 and buffer per HWC from 2 to 4.

In our evaluation, we define *injection rate* as average flits/connection/cycle (fcc). The packet delay includes both the waiting delay in the NI and the transfer delay of the network. The *throughput* results are also given as flits/connection/cycle.

The results under uniform random traffic are shown in Fig. 5a. Our highway technique greatly reduces the average packet delay. Eg. with a packet size of 16 flits, applying one HWC per input with 2 buffer stages can achieve a delay reduction of 77% at injection rate 0.1 fcc. As the packet size grows, the benefits of using highways becomes more prominent. Eg. at packet size of 32 flits, the delay reduction is 80% at injection rate 0.1 fcc, while at packet size of 64 flits the reduction becomes 84%. Besides, the maximum throughput improvement also increases from 50% to 200%, when packet size increases from 16 to 64 flits. This is because when the packet size grows, the network traffic becomes more and more bursty and thus unbalanced. As a result, our highway technique gains more chances to utilize free slots for busy connections.

We also observe that, increasing the number of HWCs per input can further improve performance. As Fig. 5b shows, increasing HWC from 1 to 2 can have a further 35% delay reduction at injection rate 0.15. However, more HWCs do not lead to apparent performance improvements, showing a performance saturation phenomenon.

Compared with adding the buffers per HWC, increasing the number of HWCs per input is more effective on performance. As Fig. 5c suggests, two HWCs per input with 2 buffers in each is far better than one HWC with 8 buffers.

Furthermore, we studied the effect of different number of reserved slots per connection and different traffic patterns. All of these results suggest that using our highway technique can greatly reduce packet delay, which is similar to the delay curves in Fig. 5a. For example, when each connection reserves 4 slots in a time window, we can still have 47% delay reduction at injection rate 0.1 fcc and 70% delay reduction at 0.23 fcc with uniform random traffic. We skip the delay results here due to space limitation.

The effects on maximum throughput are shown in Fig. 6. We find that the same HWC configuration under different traffic pattern and with different reserved slots per connection generates different throughput improvements, ranging from

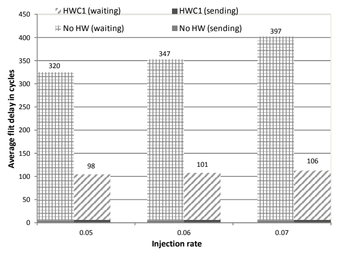


Fig. 7. Delay decomposition under uniform random traffic and a packet size of 32 flits.

417% to 8%. We also observe that, as the reserved slots per connection grows, the throughput improvements decreases. This is due to that given a fixed number of connections, as the reserved slot per connection increases, the unallocated slots decrease.

Finally, in our experiments we separate the total delay into: waiting delay and transfer delay. As shown in Fig. 7, compared with the transfer delay, the waiting delay of a flit is much bigger. For example, without a highway, the average flit waiting delay is more than 300 cycles, which will also increase dramatically when the injection rate increases, while the transfer delay is always 6 cycles. The large waiting delay of a flit is due to two reasons: (1) when a burst of flits arrives, the FIFO order requires the flits waiting in a queue, if the service rate of the output is not enough; (2)Without a highway, a TDM connection mandates a flit to wait for one of the reserved slots. The highway technique can reduce the waiting delay in both scenarios, by increasing the service rate of a connection as well as reducing the waiting time for a slot. As suggested by Fig. 7, the waiting delay is reduced from above 300 cycles to around 100 cycles, when applying 1 HWC with 2 buffers per input. With HWC, the transfer delay of a flit may slightly increased due to buffering delay in the HWCs, for example, at injection rate 0.7 fcc, the average transfer delay with a highway is 7 cycles, which is 1 cycle more than without highway. However, when compared with the waiting delay reduction, such increase can be neglected.

From these results, we find that adding 1 HWC per input with two buffers can have up to 80% packet delay reduction and up to 310% throughput enhancement. Increasing the buffers per HWC does not have significant improvements, whereas using more HWCs can further reduce the packet delay from 10% to 40% and increase the throughput by 4% to 75%. However, considering the doubled, even tripled area cost, as well as the more than the 15% increase on the critical timing path, we think it is not cost-effective to use more HWCs or more buffers per HWC. Applying one or two HWC for each input link, and 2 buffers in each HWC to cover the round-trip delay seems to be a reasonable compromise.

B. Performance Evaluation with MPSoC Benchmarks

We experimented with the NoC benchmarks designed by Pekkarinen et al [14] [15], to confirm the benefits of highways. The NoC benchmark utilizes *task communication graphs* (TCGs) to model MPSoC applications. It contains a set of processor and DSP models. Tasks can be mapped and running on these processor and DSP models. [15] has already given the task mappings. It mapped all the applications to either a 2×2 or a 4×4 mesh based MPSoC platform depending on the size of the TCGs. Users of the NoC bench are required to use their own NoC to connect all the processor/DSP cores to run these applications for evaluation.

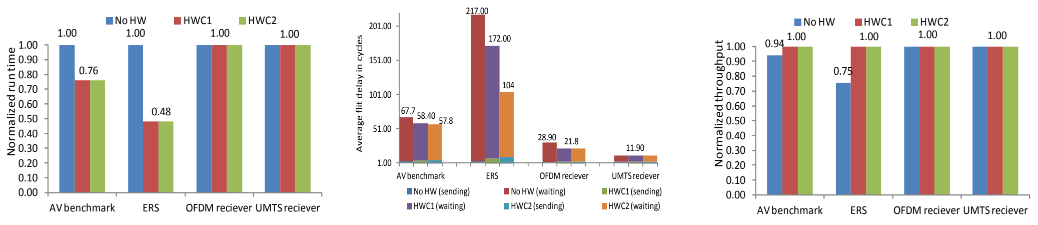
TABLE IV
Communication Properties of the applications

	AV bench	ERS	UMTS	OFDM
Connections	57	26	11	12
Avg. Burst size (flits)	531.60	12797.0	14.1	1.43
Total Request (MB/s)	6772	4488	94	196
Max Request (MB/s)	1168	512	246	80
Min Request (MB/s)	0.25	64	2	52

The NoC bench contains four applications which have throughput requirements annotated on the edges of TCGs. The details of the four applications have been described in [14]. Their TCGs can also be found in [15]. Besides, we list the communication properties of each application in Table IV, in which *Total Request* means the total throughput requirement of an application. *Min/Max Request* refers to the minimum/maximum throughput requirement among all the connections' requirements of an application. Note that the AV benchmark and Ericsson Radio System (ERS) are communication intensive, their *Total Request* are 6772 MB/s and 4488 MB/s respectively, whereas the traffic flows generated by the UMTS receiver and OFDM receiver are a magnitude less, which are 96 MB/s and 196 MB/s respectively.

Before running an application on the statically scheduled TDM NoC, we first establish TDM connections between processor/DSP cores to satisfy all the communication needs and throughput requirements described by the TCG of the application. We use a depth-fist search algorithm to search paths and allocate slots to connections. We can statically optimize a TDM NoC for an application by tuning the NoC clock frequency and the TDM window size, in order to avoid too much bandwidth waste. For this reason, we gradually increase the NoC clock frequency and the time window size until reaching a condition where our search algorithm can schedule all the connections of an application. The clock frequency of the NoC ranges from 100 to 1000 MHz. The time window size is between 2 to 32 slots.

As Fig. 8a shows, the highways significantly improve the performance of AV benchmark and ERS. The average application run time is reduced by 24% and 52%, respectively. Meanwhile, as suggested by Fig. 8c, the throughput is enhanced by 6% in AV benchmark and 25% in ERS. Such performance improvement is due to the highways in the TDM NoC. For example, we see 14% (with one HWC per input link) and 15% (with two HWCs per input link) average flit delay reduction with AV bench. We also find about 20% (one HWC per input link) and 52% (with two HWCs per input link) delay reduction in the ERS application. For these two applications, improvements on communication can also help to enhance the overall system performance, which is reflected



(a) Application performance (b) Average flit delay (c) Normalized throughput

Fig. 8. Benchmark evaluation of our highway technique

by the shortening of application run time and the increase of system throughput.

Applications like the OFDM receiver and UMTS receiver have less communication needs and their performance mainly depends on the processor speed. Therefore, from Fig. 8b, we find that for the OFDM receiver, although our highway technique can bring about 25% flit delay reduction, such improvement on communication system does not help to increase the application performance, as suggested by Fig. 8a and Fig. 8c.

We further studied the network performance, as described in Fig. 8b. We find when using highways, applications with larger burst size tend to have more delay improvements. For example, the biggest delay improvement happens with ERS, since its traffic flows has an average burst size of 12797 flits. In comparison, there is no improvement for the UMTS receiver, since it has a low traffic generation rate and the average burst size is only 1.43. In this situation, the highway setup process is seldom initiated because of not enough flits waiting in the network interface. Furthermore, we find that the transfer delay of a flit is very small, ranging from 3 to 7 cycles on average. However, the average waiting delay is relatively large, eg. it can reach up to 213 cycles. Our highway technique can greatly reduces the total delay by shortening the waiting delay, eg. from 213 cycles to 104 cycles.

VI. Conclusions

We have proposed a TDM highway technique to utilize free time slots in TDM NoCs. With the highway technique, the upper bound throughput of a connection is adaptive to link sharing situations, while it still offer QoS guarantees on the lower bound throughput and flit order. The delay of packets are greatly reduced. We can dynamically setup highways on TDM connections. One prominent aspect of our highways is that the dynamic setup method has no interference with the normal TDM data transfer and no dependency on the TDM NoC architecture. Thus, it can be utilized by TDM NoCs with either a distributed or a centralized TDM connection setup method, with either source routing or distributed routing by using distributed slot tables.

We use both synthetic traffic pattern and application benchmarks to evaluate our highway technique. With synthetic traffic pattern we find a delay reduction up to 80% and a throughput enhancement upto 417% in a statically scheduled TDM NoC, as well as up to 80% delay reduction and 17% throughput enhancement in a dynamically allocated TDM NoC. Generally speaking, the more unused slots, the more benefits; the larger the burst size, the more improvements. Also, using more HWCs for a link and more buffers per HWC can provide more performance enhancement. However, the cost-performance study suggests that using one or two HWCs per link and 2 buffers per HWC is most cost-effective in our extensive experiments. With application benchmarks, we confirm that highways can enhance the performance of a TDM NoC. However, the enhancement on NoC can reduce the run time of an application only if it is communication intensive.

References

[1] K. Goossens and A. Hansson, "The Æthereal network on chip after ten years: Goals, evolution, lessons, and future," in *DAC*, 2010

[2] R. Stefan, A. Molnos, and K. Goossens, "dAElite: a TDM NoC supporting QoS, multicast, and fast connection set-up," *IEEE Transactions on Computers*, vol. PP, no. 99, p. 1, 2012.

[3] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens, "A TDM NoC supporting QoS, multicast, and fast connection set-up," in *DATE*, 2012.

[4] A. Hansson, M. Subburaman, and K. Goossens, "Aelite: A flit-synchronous network on chip with composable and predictable services," in *DATE*, 2009.

[5] Moreira, Orlando and Mol, Jacob Jan-David and Bekooij, Marco," Online resource management in a multiprocessor with a network-on-chip" in *ACM* symposium on Applied computing, 2007.

[6] T. Marescaux, H. and Corporaal, "Introducing the SuperGT Network-on-Chip; SuperGT QoS: more than just GT " in *DAC*, 2007.

[7] Mirza, Usman Mazhar and Gruian, Flavius and Kuchcinski, Krzysztof, "Design Space Exploration for Streaming Applications on Multiprocessors with Guaranteed Service NoC" in *NocArc*, 2013.

[8] Ma, Ning and Lu, Zhonghai and Zheng, Lirong, "System design of full HD MVC decoding on mesh-based multicore NoCs" in *Hournal of Microprocessors and Microsystems*, vol. 35, no. 2, pp.217–229, 2013.

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

[10] Zhonghai Lu and Axel Jantsch, "TDM virtual-circuit configuration for network-on-chip" in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* vol. 16, no. 28, pp. 1021–1034, 2008

[11] Dally, William James and Towles, Brian Patrick, "Principles and practices of interconnection networks" in *Published by Elsevier* pp. 245–246, 2004.

[12] D. Becker and W. Dally, "Allocator implementations for network-on-chip routers," in *ACM Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. pp.52:1–52:12, 2009.

[13] S. Liu, A. Jantsch, and Z. Lu, "Analysis and evaluation of circuit switched NoC and packet switched NoC" in Euromicro Conference on Digital System Design (DSD), 2013.

[14] E. Pekkarinen, L. Lehtonen, E . Salminen, et al. "A set of traffic models for Network-on-Chip benchmarking " in *IEEE International Symposium on System on Chip (SoC)*, 2011

[15] http://www.tkt.cs.tut.fi/research/nocbench/download.html

Paper D

A Fair and Maximal Allocator for Single-Cycle On-Chip Homogeneous Resource Allocation

Shaoteng Liu, Axel Jantsch and Zhonghai Lu

Included in IEEE Transactions on Very Large Scaled Integration systems (TVLSI), vol 22, no. 10, pages: 2229–2233, 2014.

A Fair and Maximal Allocator for Single-Cycle On-Chip Homogeneous Resource Allocation

Shaoteng Liu, Axel Jantsch, and Zhonghai Lu

***Abstract*— Traditional allocators for network-on-chip (NoC) routers suffer from either poor-matching quality or limited fairness. We propose a waterfall (WTF) allocator targeting homogeneous resource allocation, which provides single-cycle maximal matching while guaranteeing strong fairness based on the round-robin principle. It can be implemented with a loop-free structure. In 90 nm technology, the allocator operates at about 1 GHz clock frequency. We compare WTF with wave-front, separable-input-first, and separable-output-first allocators and find that it is at least 10% smaller, has 50% less delay under high load, and uses 3% less power than any of these alternatives. Also, WTF is at least as fair or clearly fairer. We also find that in a 4 x 4 circuit switched NoC the use of WTF gives up to 20% higher network performance.**

***Index Terms*— Allocator, fairness, maximal matching, network-on-chip (NoC), round-robin.**

I. Introduction

An allocator performs a matching between multiple resources and multiple requesters. A matching is an assignment of resources to requesters satisfying the following three constraints [1]: 1) a resource is granted to a requester only if the corresponding request exists; 2) each resource is granted to at most one requester; 3) a requester is at most granted once. A matching in which no additional requests can be served without removing one of the existing grants is called a maximal matching [2] and the one containing the maximum number of grants is called a maximum matching. Maximum matching is often too costly to be realized in hardware. However, maximal matching is achievable. In addition to matching quality, fairness is an important property for an allocator, and we can distinguish between strong fairness and weak fairness [2]. Intuitively, strong fairness guarantees that all requesters are served in proportion to their relative request rates. In practice, this means that persistently active requesters are served in a periodic sequence equally often within each reasonably short period. In contrast, weak fairness only requires that every request is eventually granted, without any guarantee at which rate or in which relative proportion different requesters are served.

Allocators used in network-on-chip (NoC) routers have limitations. Compared to large scale networks, the performance of NoCs is more sensitive to the latency of each router. This mandates the use of single cycle allocators in router design [2]. Consequently, conventional allocators in NoCs do not take into account the maximal matching quality and strong fairness at the same time. Strong fairness is usually provided by using a variation of the round-robin principle, which states that a request that was just served should have the lowest priority in the next round [1]. On one hand, allocators which adopt round-robin cannot ensure maximal matching. Separable-input-first (SIF) and separable-output-first (SOF) allocators [2], [3] are classified in this category. On the other hand, maximal matching allocators,

Manuscript received January 1, 2013; revised June 26, 2013; accepted September 9, 2013. Date of publication October 22, 2013; date of current version September 23, 2014.

The authors are with the Department of Electronic Systems, KTH Royal Institute of Technology, Stockholm 16440, Sweden (e-mail: liu2@kth.se; axel@kth.se; zhonghai@kth.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2284563

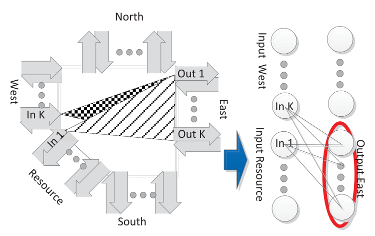


Fig. 1. Illustration of homogeneous allocation in NoC.

such as wave-front (WVF) [4]–[6] or rectilinear-propagation-arbiter and diagonal-propagation-arbiter [7], do not use the round-robin principle and only provide week fairness. For general matching problems, no ideal solution is known to design a NoC allocator overcoming all these shortcomings.

However, in NoC design practice, we frequently encounter a kind of special matching problem called homogeneous resources allocation (HRA). Here, homogeneous resources refer to a class of resources that have the same functionality and can be used interchangeably. This kind of matching problem obeys two more constraints besides the three constraints introduced above: 4) for each requester, all resources it desires belong to the same class; and 5) any resource of a class can be granted to the requester who have requests on this class.

We use a router model to illustrate the HRA problem. The router in Fig. 1 has five directions and each direction contains k-duplexed channels. Channels in this model can either be regarded as virtual channels (VCs) that share one physical channel in a time division multiplexing way, or subphysical channels by splitting the wires of a link in an spacial division multiplexing (SDM) way. The output channels of each direction form one resource class. The routing algorithm is deterministic, e.g., dimension-order-routing, which assigns each arrival packet one and only one desired output direction. For example, in Fig. 1, both packets from input channel k of the west and input channel 1 of the resource desire an output channel to the east. The right part of Fig. 1 is the bipartite graph of this case with each line representing a request. Since all requests of a packet are confined to output channels of the east direction, constraint 4) holds, and because each packet asserts requests on every output channel of the east direction, constraint 5) can be satisfied. Therefore, channel allocation inside such a router is an HRA problem.

For HRA, we propose a single cycle allocator which guarantees both maximal matching and strong fairness. We call it "water-fall" (WTF) because it finds the matching in several consecutive steps. For an n-requester allocator, it requires n steps. WTF is implemented entirely as combinational logic, which means the allocation takes one cycle. It can be implemented free of combinational loops that are common in traditional maximal allocators, e.g., the wave-front allocator in [4]. We develop a fairness policy which inherits the principle of round-robin and name it as massive round-robin (MRR) for HRA.

HRA is an abstraction of a class of allocation problems which are frequently encountered in NoC designs. For example, either VC allocation in wormhole-based packet switched NoC [3] or subchannel allocation in circuited switched NoC using SDM [8] is an HRA problem. However, aside from our examples, we believe that our allocator can be further utilized by other kinds of on-chip applications which have HRA problems, even beyond the scope of NoC usage.

II. Design of the Allocator

A. Basic Structure of the Allocator

Intuitively WTF reduces the request matrix (rows in this matrix represent requesters and columns represent resources) by taking advantage of homogeneous resources. As the matrix in Fig. 2 suggests, input channels 0, 1, 3 each request a channel from output direction 1. Input channel 2 requests a channel from output direction 2. Each input packet just has one desirable output direction, thus constraints 4) and 5) hold and it is an HRA problem. For an HRA problem, first, the big matrix can be split into submatrices according to resource classes (output directions). Secondly, we can merge the requests of a submatrix. As described in Fig. 2, the 4 × 4 matrix is split up into two separate 4 × 1 matrices representing requests for output direction 1 and 2, respectively.

Continuing the example, we apply two separate homogeneous resource allocators to solve the two reduced matrices. We use a ripple carry arbitration scheme to design such an allocator. Taking the allocation in output direction 1 as an example, as described in Fig. 3(a), since it has two resources and four requesters (marked as $2 \rightarrow 4$), the allocator is made up of two columns and four rows of arbitration cells accordingly. The three active requesters are indicated by "1" in the reduced matrix. Two tokens are used to denote the availability and grant decisions of the two output channels (resources), respectively. In the current round, the arbitration for tokens starts from the row 2, moves counter-clock wise, and ends at row 1. Thus, the 4 requesters are served in this order: (r_2, r_3, r_0, r_1). r_3 will be the first to catch a token and r_0 the second. This means channel 1 is granted to r_3 and channel 2 is granted to r_0.

Considering fairness, we need to roll the start row. Our MRR fairness policy is illustrated in Fig. 3(a) and (b). The principle is that the end row in the next round is the last successful requester of the previous round. And the start row is acquired by incrementing the end row by 1, then modulo n. If no requester is granted, the start row remains the same as in the previous round. Applying this policy, active requesters r_0, r_1, r_3 in Fig. 3 (assuming they are persistently active) are granted in the periodic sequence: $\{(r_3, r_0)_{\text{round}0}, (r_1, r_3)_{\text{round}1}, (r_0, r_1)_{\text{round}2}, (r_3, r_0)\ldots\}$. The start row i is selected by asserting $p_i = 1, 0 \leq i \leq n-1$, where n is the number of requesters.

In general, we derive the function of the MRR policy as follows. Given current grants g_i, $0 \leq i \leq n-1$ for each requester ($g_i = 1$ means granted), and suppose G is the set of granted requesters of the current round with $G = \{x | 0 \leq x \leq n-1,\ g_x = 1\}$ and b denotes any granted requester that $b \in G$, and the current start row is $k_{(t)}$ and the end row is f. Compute the start row of the next round $k_{(t+1)}$ as follows:

$$\text{(if } \exists f \text{ that)}$$
$$\begin{cases} (f + n - k_{(t)})_{\text{mod}n} = \max\{(b + n - k_{(t)})_{\text{mod}n}\} \\ f \in G \end{cases}$$
$$\text{(then } k_{(t+1)} = (f+1)_{\text{mod}n},\ \text{otherwise } k_{(t+1)} = k_{(t)}\text{)}.$$

Although in WTF there is no actual logic feedback, we also need to avoid combinational circuits with loops which are undesirable due to issues in verification and testing [7]. In Fig. 3(c), we propose a loop-free structure by adding redundant logic. For an n-requester allocator, our loop-free structure contains $2n - 1$ rows. The bottom $n - 1$ rows replicate the top $n - 1$ rows. In this way, rolling of the start row is equivalent to selecting an active area. Fig. 3(c) shows how to convert a loop structure into a loop-free structure. For example, suppose all requesters are served in the order (r_1, r_2, r_3, r_0) in the current round. Mapping into the loop-free structure, the area from row 1 to row 5 is activated, as the rectangular box in Fig. 3(c) suggests. Since row 5 replicates row 0, the allocation is also in the order (r_1, r_2, r_3, r_0). In this way, the functions of the loop and loop-free structures are equivalent.

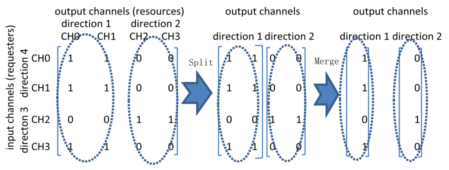


Fig. 2. Reduction of request matrix.

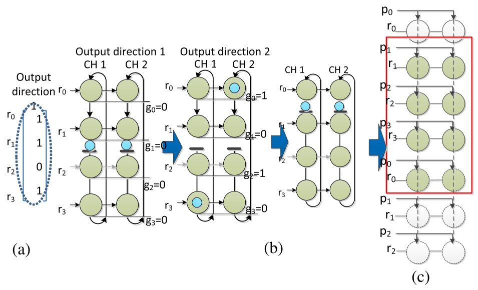


Fig. 3. Allocation mechanism, fairness policy and loop-free structure (circles represent arbitration cells, dots represent tokens. (a) Allocation in output direction 1. (b) Start row rotation. (c) Loop-free structure).

B. Implementation

A WTF allocator consists of two parts: allocation logic and priority updating logic. The allocation logic is used to generate grants of the current round. The priority updating logic is used to guarantee the fairness.

1) Implementation of the Allocation Logic: A loop-free design is described in Fig. 4(a). An n-requester m-resource allocator ($m \rightarrow n$) has $2n - 1$ rows and m columns. The top n rows are made up of white cells, whereas the bottom $n - 1$ rows are composed of dark cells. The right part of Fig. 4(a) depicts the internal logic of the two arbitration cells. A white cell plays a role as "token" starter when its priority p is asserted. It directly accepts the channel status as tokens. When p is de-asserted, its role is a "token" passer and it can receive a token passed by the upper cell from its north input. A request is injected from the W(west) input of a cell. When an arbitration cell receives a token and if it has one request asserted, it consumes the token and grants the request with $g_{ij} = 1$. Otherwise, it passes the token on from its south output. The role of a black cell is a token passer when p is de-asserted. When p is asserted, the black cell's function is that of a token terminator. In this situation, all its outputs are set to "0." Token passing ends at this cell.

Suppose p_i, $0 \leq i \leq n-1$, is set as "1," then the effective tokens are injected from row i and passed downward. Meanwhile, row $i+n$ in the dark region also gets its priority line p_i asserted and thus stops tokens passing. In such a way, the active region starts from row i and ends at row $i + n - 1$.

2) Implementation of the Priority Updating Logic: The key point of the priority updating algorithm is to find the end row for the new round. This is a $1 \rightarrow n$ allocation problem. The n-bit grant vector generated by the allocation logic denotes requesters. The requester which gets the token will turn into the end row, as shown in Fig. 4(b). This time the token passing is clock-wise, which is

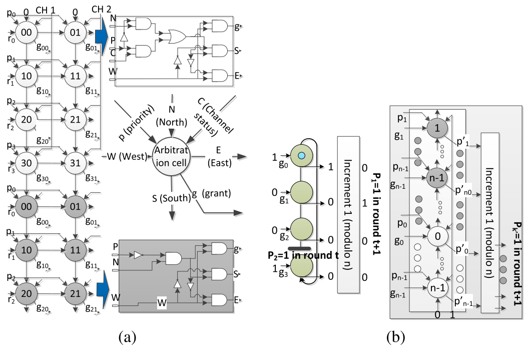


Fig. 4. Hardware implementation of the WTF allocator. (a) Allocation logic. (b) Priority updating logic.

opposite to the allocation logic. The start row is the same start row used by allocation logic for the current round. For example, in a (2 → 4) allocator, suppose p_2 is asserted and thus the grant vector generated is 1001. For the priority updating logic, the token is also injected from row 2 but passed in clock-wise order. As a result, row 0 (g_0) catches the token. Then row 1 will be the start row in the next round (incrementing the end row by 1, then modulo n). The whole priority updating logic is implemented as illustrated in the right part of Fig. 4(b).

III. Evaluation and Analysis

It is known that fairness and performance are two key metrics for an allocator. We evaluate and analyze both aspects. The router model described in Fig. 1 is simulated. The arrival packets at each input channel are queued up. When an output channel is granted to an input channel, one packet leaves the queue. Each packet just needs 1 cycle to be delivered. The arrival time of packets obey a Poison process or an on-off process. The arrival packets are uniformly and randomly distributed among all input channels.

In our setting, each direction of the router contains four duplex-channels. Thus the channel allocation inside this router forms a 20 → 20 allocation problem. However, considering HRA, the 20 → 20 matrix can be split into five 4 → 16 submatrices, each of which represents the channel allocation of one output direction. Accordingly, we can assign each output direction an allocator. Since our purpose is to evaluate the fairness and performance of an allocator, in order to avoid influences such as head of line blocking, we assign every arrival packet the same target output direction.

In addition to the WTF allocator, we modeled several other allocators for comparison, including SIF allocator, SOF allocator, and WVF allocator. They are reported as representative allocators in NoC design in [2]. Both the two separable allocators adopt two stages of round-robin arbitration. The wave-front allocator adopts rotating policy [1], [9] by incrementing the priority each round. All allocators are tested under the same packet injection test bench for each utilization. Here utilization equals the duration of an output channel occupied by a packet (1 cycle in this case) multiplied by the sum of injection rates of all inputs and divided by the number of resources (4 in this case). We simulated 40 000 cycles at each utilization for each allocator.

A. Performance

Performance is affected by matching quality of an allocator. Fig. 5(a) shows the average delay in terms of cycles versus the

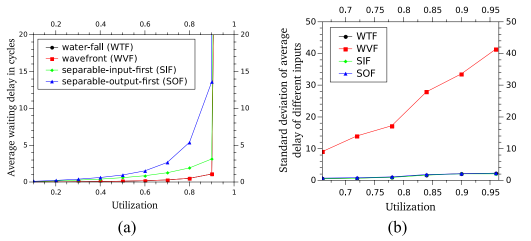


Fig. 5. Performance comparison between allocators [the waiting delay of a packet is the time that the packet waits in the input queue. The total delay (response time) of a packet equals the waiting delay plus one cycle]. (a) Average waiting delay in cycles (The curves of WVF and WTF are almost identical.) (b) Random case fairness. (The curves of WTF, SIF, and SOF are almost identical.)

TABLE I
Synthesis Results of Different Allocators

Structure	WTF	SIF	SOF	WVF
Area (um^2)	29990	33549	34196	120466
Delay (ns)	1.04	1.0	1.0	2.0
Power (mW)	9.2	9.5	11.2	9.6

utilization. It suggests that WTF exhibits the same average waiting delay in cycles as WVF. This is because both WVF and WTF are maximal allocators, they generate the same number of grants every cycle, thus the average packet delay is the same. The performances of the two separable allocators are worse than WTF and WVF because no maximal allocation guarantee is provided. At utilization 0.9, the waiting delay of WTF and WVF are both 1.2 cycles. And it is 3.1 cycles for SIF and 13.3 cycles for SOF. For separable allocators, in some cases, resources are left unassigned even in the presence of requests waiting for resources, and thus performances are degraded. Although the delay in cycles of WTF and WVF are very close, when the differences in clock frequency are considered, WTF has 50% of the actual delay in ns of WVF (WTF is about as twice fast as WVF, as Table I suggests.)

B. Fairness

In addition to lower performance, WVF is also less fair.

1) Example Study: Continuing our previous example shown in Fig. 3(a), for an 2 → 4 allocator, suppose r_0, r_1, r_3 is continuously asserted. For WTF, three active requesters are served in a periodic sequence (r_3, r_0), (r_1, r_3), (r_0, r_1), $(r_3, r_0)\ldots$. Each period has three allocation rounds. Inside a period, every requester is served twice. The normalized throughputs are (0.66, 0.66, 0.66), respectively. However, for WVF with priority rotating policy, the periodic sequence is (r_0, r_3), (r_1, r_0), (r_3, r_1), (r_3, r_0), $(r_0, r_3)\ldots$. Each period has four allocation rounds. Inside a period, r_0 and r_3 are served three times, but r_1 is only served two times. The throughputs are (0.75, 0.75, 0.5), respectively.

2) Worst Case Analysis: In general, we assume an allocator has n inputs and m resources ($m \le n$). Suppose during a time interval (t_1, t_2), p requesters ($m \le p \le n$), numbering $r_1, r_2, \ldots, r_p$, are constantly active. Each requester will occupy the resource for L cycles when it is granted. Let $V_i(t_1, t_2)$ denote the amount of service received by requester i in (t_1, t_2).

Applying our MRR policy, we can ensure that between any two service opportunities given to requester r_i, requester r_j must have had an opportunity. Thus, suppose during (t_1, t_2), requester i gets

z service opportunities and requester j gets z', then $\mid z - z' \mid \leq 1$. Therefore

$$|V_i(t_1, t_2) - V_j(t_1, t_2)| = |zL - z'L| \leq |z - z'|L = L. \quad (1)$$

As interval $(t_1, t_2) \to \infty$, the difference in amount of services of any two requesters is always bounded by L.

For the WVF, the p requesters are served in a periodic sequence and each period contains n allocation rounds. Thus, each period is nL cycles long. During every period, in the worst case, the most favorite requester r_i can have $m+n-p$ service opportunities, while the least served requester r_j just has m service opportunities. Suppose there are q periods during interval (t_1, t_2) that $q = \lfloor t_2 - t_1/nL \rfloor$, then

$$(m+n-p)(q-1)L \leq V_i(t_1, t_2) \leq (m+n-p)(q+1)L \quad (2)$$

$$m(q-1)L \leq V_j(t_1, t_2) \leq m(q+1)L \quad (3)$$

$$|V_i(t_1, t_2) - V_j(t_1, t_2)| \leq (n-p)qL + (2m+n-p)L. \quad (4)$$

As interval $(t_1, t_2) \to \infty$ and $q \to \infty$, there is no worst case bound for WVF.

3) Random Case Study: The simulation model used for performance is used to study the stochastic fairness behavior.

Under certain traffic patterns, an unfair allocator may generate significant biased allocations. For example, as long as frequently active requesters are not evenly spaced among all requesters, the allocation may be unfair with WVF. We will take one of such traffic pattern for a study. The traffic pattern is set as follows: the packets are injected from only 12 of the total 16 inputs: from input 0, 1, ... to input 12. The packets injection process is an on-off process. This two-state Markov modulated process has probability α to switch from off to on, and a probability to β switch from on to off. In the on state, each input has the same probability r_1 of injecting a packet. While in the off state, no packets can be injected. Thus, the average injection rate of each input is $\alpha r_1/(\alpha + \beta)$.

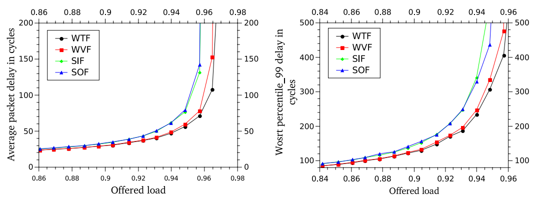
Under a certain utilization, the average packet waiting time of input queue i is marked as $D_i, 0 \leq i \leq 12$. We use standard deviation of $D_i, 0 \leq i \leq 12$ [denoted as $\sigma(D)$] as a metric of fairness. As we can imagine, unfair allocators will result in significant variance of these D_i values. Therefore, its $\sigma(D)$ should be higher than that of fair allocators.

Fig. 5(b) suggests the $\sigma(D)$ values of several allocators under different utilizations. Generally speaking, under every utilization, the value of WVF is much higher than the others. For example, at utilization 0.78, the $\sigma(D)$ of WVF is 17.2, while that of WTF is only 1.0, SIF is 0.90 and SOF is 1.1. This means that WVF does not treat every input queue fairly. In other words, certain inputs are served more often than the others.

C. Synthesis Results

We synthesize the allocators used in the router model with TSMC 90 nm technology with Synopsys Design Compiler. The results are listed in Table I. The power numbers are obtained by assuming 50% switching activity of each input signal. Note that the WVF used for synthesis also has a loop-free implementation by replicating the array for each possible priority diagonal and selecting the grant matrix generated by the currently active one. For details of this WVF allocator implementation, refer to [2].

We find that our WTF allocator is slightly slower than SIF and SOF, but it consumes less area and power. It is much faster than WVF. This is because WVF has to be square. In this case, it is more efficient to be implemented as one $20 \to 20$ allocator rather than five $4 \to 16$ allocators. As a result, its critical path is longer than WTF. Besides, since the WVF avoids combinational loops by replicating the entire 20×20 array for each priority, it consumes much more area than other allocators.



(a) (b)

Fig. 6. Performance comparison in circuit switched NoC in cycles. (a) Average delay in cycles. (b) Percentile delay in cycles.

IV. Application Study

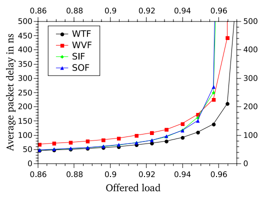
Our allocator is used in a circuit switched NoC design. The circuit switched NoC consists of 4×4 routers and adopts a mesh topology. Inside each router, every direction has four-duplexed SDM subchannels. A parallel probing method [10] is used for path set-up. Probes are used to set-up a minimal path for data transfer. At the beginning, one probe carrying a set-up request is sent out by the source node. At each hop, when a probe enters into a router, it can split into multiple probes if it has multiple preferable output directions. As probes travel, they will reserve the output channels which they have been allocated inside each router for future data transfer. Whenever two probes carrying the same request meet, one of them is regarded as redundant and is cancelled and all channels used only by the cancelled probe are released. Each probe takes two cycles for a hop. For detailed description of this router architecture and set-up method, refer to [10].

Since each probe is assigned only one target direction, allocating output channels to probes is an HRA problem. For evaluation, uniform random traffic is applied. When a path is established, a packet with eight flits is delivered. After data transfer the path is released.

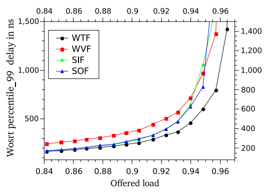
The results are shown in Fig. 6. Fig. 6(a) gives the average packet delay in cycles versus offered load, by assuming that routers with different allocators are working at the same clock frequency. Offered load refers to the time used for data transfer of a path (8 cycles in this case) multiplied by per node set-up request injection rate. In this case, WTF has the best performance, e.g., at offered load 0.95, the average packet delay by using WTF is 56 cycles. By using WVF, it is 59 cycles. For SIF and SOF, they are 76 and 79 cycles, respectively. Although WTF and WVF are both maximal allocators, the unfairness of WVF might cause unbalanced congestion of channels. In this situation, some channels become saturated earlier than others. Thus, we observe that the performance of WVF is slightly inferior to WTF. This difference will be enlarged at high load, e.g., at load 0.96, the average delay of WTF is 107 cycles while it is 152 cycles for WVF.

We also compared their worst packet delay by using a statistical method. We use percentile as a measurement. For example, Percentile(n) is the minimum delay of the $(1-n)$% packets that experience longest delays in our simulation.[1] In Fig. 6(b), the percentile(99) delay of WTF is always smaller than WVF. For example,

[1]We choose this metric because it is less susceptible to statistical abnormalities [6]. Each percentile value is taken from a large amount of samples that come from 2 million simulation cycles. Thus, the upper and lower bound of the confidence interval of a percentile value are believed to be very close. For example, at load 0.95, for WTF, we ran several simulations with different random seeds and total simulation cycles, ranging from 250 thousand to 5 million cycles. The percentile(99) values of each simulation were all around 307, with less than 1% difference.



(a)



(b)

Fig. 7. Performance comparison in circuit switched NoC in ns. (a) Average delay in ns. (b) Percentile delay in ns.

at load 0.95, percentile(99) delay of WTF is 307 cycles, and it is 335 cycles for WVF. At load 0.96, the WTF is 406 cycles, and it is 476 cycles for WVF. This result also support that WTF is fairer than WVF. We also notice that the percentile(99) delay curves of SIF and SOF are worse than WVF and WTF, because their performances cannot catch up with maximal allocators such as WVF and WTF.

In Fig. 7, we evaluate the influences of using different allocators on the critical timing path, which mainly consists of one allocator latency plus one crossbar latency. In this case, a router with a WTF allocator can work at 510 MHz clock frequency. A router with SOF or SIF can work at 526 MHz. And a router with WVF operates at 345 MHz. As a result, we can measure average packet delay in ns. In Fig. 7(a) we find that the WTF has the best performance and WVF the worst. At load 0.94, the average packet delay is 92 ns for WTF. And it is 117 and 116 ns for SIF and SOF, respectively. For WVF, it is 141 ns. We also measure percentile(99) of the worst case delay in ns, as shown in Fig. 7(b). For example, at load 0.93, for WTF, it is 366 ns. For SIF and SOF, it is 469 and 473, respectively. It is 569 ns for WVF. Hence, WTF is superior to the three alternatives.

V. Conclusion

Matching quality and fairness are two important concerns for designing an allocator. While achieving one or the other, traditional allocators for NoCs fail to succeed in both aspects. In this brief, we propose an allocator called WTF for homogeneous resource allocation. This allocator guarantees both maximal matching quality and strong fairness. Furthermore, our allocator can be implemented in hardware without combinational loops. The abilities in performance and fairness of our allocator are analyzed and demonstrated in simulation. We also use WTF in a 4 × 4 circuit switched NoC design. Experiment results suggest that WTF offers better performance and lower area than traditional allocators while achieving strong fairness.

References

[1] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks* (The Morgan Kaufmann Series in Computer Architecture and Design). San Mateo, CA, USA: Morgan Kaufmann, Dec. 2003, pp. 351–375.

[2] D. Becker and W. Dally, "Allocator implementations for network-on-chip routers," in *Proc. Conf. High Perform. Comput. Netw., Storage Anal.*, 2009, pp. 1–12.

[3] S. Park, T. Krishna, C. Chen, B. Daya, A. Chandrakasan, and L. Peh, "Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45 nm SOI," in *Proc. 49th Annu. DAC*, 2012, pp. 398–405.

[4] J. Delgado-Frias and G. Ratanpal, "A VLSI crossbar switch with wrapped wave front arbitration," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 50, no. 1, pp. 135–141, Jan. 2003.

[5] W. Olesinski, H. Eberle, and N. Gura, "PWWFA: The parallel wrapped wave front arbiter for large switches," in *Proc. Workshop HPSR*, Jun. 2007, pp. 1–6.

[6] Y. Tamir and H.-C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 1, pp. 13–27, Jan. 1993.

[7] J. Hurt, A. May, X. Zhu, and B. Lin, "Design and implementation of high-speed symmetric crossbar schedulers," in *Proc. IEEE ICC.* vol. 3. Jun. 1999, pp. 1478–1483.

[8] A. Lusala and J.-D. Legat, "Combining sdm-based circuit switching with packet switching in a NoC for real-time applications," in *Proc. IEEE ISCAS*, May 2011, pp. 2505–2508.

[9] H. Chi and Y. Tamir, "Decomposed arbiters for large crossbars with multi-queue input buffers," in *Proc. IEEE ICCD VLSI Comput. Processors*, Oct. 1991, pp. 233–238.

[10] S. Liu, A. Jantsch, and Z. Lu, "Parallel probing: Dynamic and constant time setup procedure in circuit switching NoC," in *Proc. IEEE DATE*, Mar. 2012, pp. 1289–1294.

High-Speed Dynamic Asynchronous Pipeline: Self-Precharging Style

C.K. Midhun, Jephy Joy, and R.K. Kavitha

***Abstract*—This brief proposes a new type of high throughput asynchronous pipeline structure called self-precharging (SP) pipeline. The proposed SP pipeline targets dynamic linear datapaths for fine grain and gate level pipelining. The novel SP protocol and modified structure enables SP pipeline to deliver multi gigahertz throughput without degrading the per-stage forward latency of the pipeline. An asymmetric C (aC) element is used to combine the two control signals, which are used to evaluate and precharge a stage. Because of the aC element, the pipeline is able to remove one of the important timing constraints present in lookahead pipelines (LP). Since the SP signal is coming from the same stage without altering the functionality of the signal, the wiring load of handshaking signals between two stages is maintained minimum. The pipeline is implemented in 90 nm UMC technology and it offers (2.227 giga data items/s) more than twice the throughput of Williams' PS0 and more than 20% improvement over the best lookahead dual rail pipeline (LP2/1). The area and power consumption of the proposed pipeline are comparable with the state-of-the-art asynchronous pipeline topologies.**

***Index Terms*—Asynchronous pipeline, dual-rail logic, dynamic logic, fine-grain pipelining.**

I. Introduction

The speed of VLSI circuits increases as the feature size of transistor decreases. To satisfy the ever growing industrial requirements, pipelining can be further employed to enhance the speed. Pipelining can be either synchronous or asynchronous. In synchronous circuit design, a globally distributed clock controls and updates all the memory elements such as flip-flops [1] and latches at the same time. The period of the clock is designed by considering the combinational logic delay, setup time of the memory elements, and the process, voltage, and temperature variations.

Synchronous circuit designs are less complex when compared with asynchronous circuit designs [2]. But the distribution of global clock with controlled clock skew and latency is an

Manuscript received January 25, 2013; revised July 10, 2013; accepted August 28, 2013. Date of publication October 22, 2013; date of current version September 23, 2014.

The authors are with the Department of Electronics and Communication Engineering, National Institute of Technology, Tiruchirappalli 620015, India (e-mail: midhu.nambiar@gmail.com; jephyjoy@gmail.com; rkkavitha@nitt.edu).

Digital Object Identifier 10.1109/TVLSI.2013.2282834

Paper E

MultiCS: Circuit Switched NoC with Multiple Sub-Networks and Sub-Channels

Shaoteng Liu, Axel Jantsch and Zhonghai Lu

Included in Journal of System Architecture (JSA), vol 61, issue. 9, pages: 423–434, 2015.

Journal of Systems Architecture 61 (2015) 423–434

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

MultiCS: Circuit switched NoC with multiple sub-networks and sub-channels

Shaoteng Liu [a,*], Axel Jantsch [b], Zhonghai Lu [a]

[a] KTH Royal Institute of Technology, Sweden
[b] TU Wien, Vienna, Austria

A R T I C L E I N F O

Article history:
Received 26 August 2014
Received in revised form 9 June 2015
Accepted 27 July 2015
Available online 13 August 2015

Keywords:
NoC
Circuit switched
Multi-channel
SDM

A B S T R A C T

We propose a multi-channel and multi-network circuit switched NoC (MultiCS) with a probe searching setup method to explore different channel partitioning and configuration policies. Our design has a variable number of channels which can be configured either as sub-channels (spatial division multiplexing channels) or sub-networks. Packets can be delivered on an established connection with one or multiple channels. An adaptive channel allocation scheme, which determines a connection width according to the dynamic use of channels, can greatly reduce the delay, compared to a deterministic allocation scheme. However, the latter can offer exact connection width as requested. The benefits and burden of using different number of channels and configurations are studied by analysis and experiments. Our experimental results show that sub-network configurations are superior to sub-channel configurations in delay and throughput, when working at the highest clock frequency of each configuration. Under reasonable channel partitioning, sub-networks with narrow channels can generally achieve higher throughput than the network using single wide channels.

1. Introduction

Compared to packet switched (PS) NoCs with TDM channels, circuit switched (CS) NoCs are preferable under certain requirements. For example, the advantages of using CS NoC with spatial division multiplexing (SDM) channels on streaming applications like 3D graphics have been demonstrated in [1].

With the increasing number of wires available on-chip, the number of possibilities to organize, use and allocate them grows combinatorially. We have more freedom to choose the number of channels and allocate wires for them, instead of giving all wires to only one channel. Thus, how to allocate wire resources and organize multiple channels becomes an interesting research question.

We propose a CS network with multiple channels and multiple networks (MultiCS) to explore the effects of different channel partitioning and configuration polices. We offer the following contributions:

- The proposed CS NoC combines spatial division multiplexing [1], which we call sub-channels, with sub-networks. Sub-channels divide the wires between two switches which then can be allocated separately and independently. Sub-networks are independent networks that connect to the same nodes. A connection between two nodes can utilize one or several sub-channels and one or several sub-networks (Section 3).
- We extend the parallel probing setup method described in [2] to allow a single communication to utilize several sub-networks or sub-channels to meet its bandwidth requirements. This setup method uses a minimal number of extra wires. If a connection, possibly spanning across multiple sub-channels and sub-networks, with sufficient bandwidth is available, it will be found in $3 * D + 4$ cycles, where D is the distance between source and destination (Section 3).
- We propose two schemes for building connections with multiple channels. One is called deterministic channel allocation (DCA), which imposes an exact width requirement on a connection. The other is called adaptive channel allocation (ACA), with which the width of a connection is allocated according to run-time channel usage of the network. ACA avoids the channel fragments and superfluous connections problems associated with DCA. However, it guarantees only a minimum connection width (Section 4).

* Corresponding author. Tel.: +46 722935320.
E-mail addresses: liu2@kth.se (S. Liu), axel.jantsch@tuwien.ac.at (A. Jantsch), zhonghai@kth.se (Z. Lu).

http://dx.doi.org/10.1016/j.sysarc.2015.07.013
1383-7621/© 2015 Elsevier B.V. All rights reserved.

- We discuss the implementation cost of MultiCS. We also develop an analytical performance model to explain how the maximum throughput is related to packet size and the number of channels (Section 5).
- We evaluate five configurations of MultiCS under the two schemes. The five configurations vary the number of sub-networks and sub-channels. The experiment results comply with our previous model and analysis (Section 6).

2. Motivation

CS NoC has a fundamental limitation. When a channel between two switches is allocated to one connection, it cannot be used by any other connection. This inherent inflexibility limits the usefulness of CS. A solution is to partition the channel into multiple narrow channels and allocate only one or a few narrow channels to a given connection. An obvious question is, what would be the trade-off of number of narrow channels: as many as possible, or is there any limitation?

The second question is: what are pros and cons of different ways of organizing multiple channels? Generally speaking, there are two methods to configure channels of a node: by sub-channels or by sub-networks, as illustrated in Fig. 1. The term sub-network refers to several separate circuit switched networks working in parallel. A network interface has access to all available sub-networks, but once data has entered one particular sub-network, it cannot change to another sub-network. In contrast, sub-channels are parallel links between switches. Those sub-channels are organized as SDM inside a switch. Data can use different sub-channels for different hops in the network.

The third question is how to establish a connection with multiple channels. Traditionally, even if there are multiple channels, still each packet flow is delivered by building a connection with only one channel [3,4] per hop. However, multiple channels have offered us the possibility of building a wide connection by combining several channels together for data transfer. Thus, we need new schemes and guidelines for connection set-up.

In the following we answer these three questions by comparing alternatives with the same total wire resources and with the same path search and setup algorithm. We construct our platform by combining features and merits from earlier work [2,5,6], and modify them to support multiple sub-channels and multiple sub-networks. We use a mesh topology but arguable many of our main conclusions are also valid for other topologies.

3. Architecture of MultiCS using parallel probing setup

Our MultiCS NoC can have any number of sub-channels, sub-networks, or a combination of them. The term “sub” denotes the number of sub-networks used, and the term “ch” denotes the number of sub-channels used. Suppose k is the number of sub-channels of a switch and m is number of sub-networks (*subm_chk*), and given that the total number of wires of a switch is a constant.

3.1. Overview of a switch

As shown in Fig. 2, in a mesh topology every switch has five directions which are used for connecting to four neighbors and one local resource. Each direction may have multiple duplex channels. Each channel contains a data path, which is used for carrying the probe during the setup phase and for transmitting data when a connection has been established. Every data path is also associated with 3 bits control signals: an answer (ANS) signal consisting of 2 bits, which goes in the opposite direction to the data channel, and 1

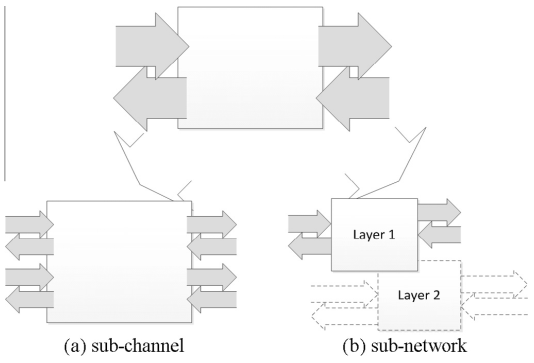


Fig. 1. Splitting wide channel into sub-channels or sub-networks.

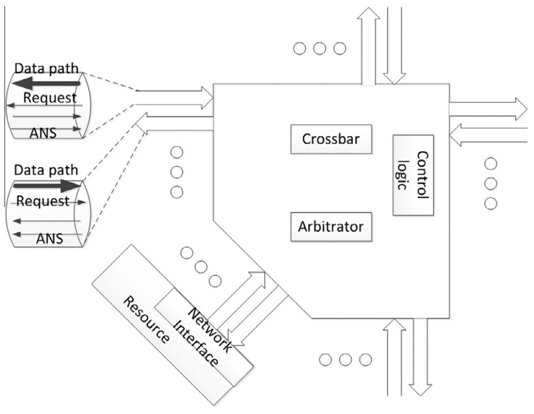


Fig. 2. Overview of a switch.

bit for a request signal, which travels in the same direction as the data channel. When the request signal is ‘1’, a probe search is running or data transfer is active. When request signal is ‘0’, it denotes the idle state, and an established connection will be released. The usage of the ANS signal is listed in Table 1.

The messages required for connection configuration are simplified by using these control signals: tear-down message is carried by the request signal, and Ack/Nack messages are delivered by ANS signal. They are free of contention.

In this switch architecture, the overhead of each channel is just the 3-bit control signals. We believed this is the minimum requirement for supporting probing based setup. Although it may be argued that the 1 bit request signal can be omitted, this would increase the logic inside a switch.

The probe format is also compact. It contains source address, destination address and the channel id of the network interface of the source (Table 2). In an 8×8 mesh with 4 sub-channels/sub-networks, the minimum width of a probe is 14 bits. Each probe is one flit in length. Thus the width of a data path is ranged from 14 bits to any bits.

3.2. Path searching algorithm

Generally speaking, our platform is not bound to a particular path searching algorithm, meaning that any algorithm can be

Table 1
Usage of ANS signal.

ANS	Usage
00	Idle
01	Probe search continue
10	Probe failed
11	Path established

applied. The performance comparison results of different path searching algorithms are shown in Section 6.2.1.

The parallel probing algorithm [2] is chosen as our default path searching algorithm in this paper because of its high performance.

Parallel probing is an adaptive path searching algorithm. The fundamental idea was proposed by us in [2]. We illustrate this idea in Fig. 3. Suppose node 1 want to set-up a path to the destination node 9. In the beginning, a setup probe carrying information such as source address, destinations address and channel id is generated by the network interface and enters into node 1. Then, node 1 sends out two probes to the neighboring nodes 2 and 4. In the second hop each probe splits into two, and all the probes continue to move towards the destination along all the possible minimum paths.

Whenever two probes containing the same setup request meet, one of them is regarded as redundant and is canceled, and all channels used only by the canceled probe are released. For example, when node 5 receives two probes which are the same, one of them is canceled and all the channels it has booked before are released, (we have marked the released channel with a cross marker) as shown in Fig. 3b). Note that, the channel between node 1 and node 2 is not released, because it is still needed for the probe that has traveled further to node 3.

The release process proceeds backwards hop by hop. The switch does a release based on the registered connectivity information that connects an input port to an output port. When a release signal (ANS = 10, "probe failed") arrives at an output port from a downstream switch, the corresponding input port is looked up, the connection is canceled, and the release signal is forwarded upstream to the input port. Applying this mechanism, if several possible paths exist, one and only one of them can be finally booked, just as desired.

In this way, a wavefront of probes travel through the network and reach the destination on a minimal path. The time is exactly 2D, where *D* is the distance in terms of hops, and it takes 2 cycles to traverse each hop. When a probe successfully reaches the destination, an acknowledgment is sent back to the source node, which takes 1 cycle per hop.

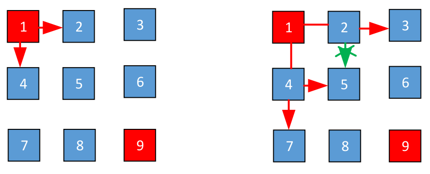
However, in our previous work [2], this algorithm is designed for circuit switched NoC with one wide link per direction. It applied a complicated priority based arbitration mechanism which is not applicable for multiple sub-channel usage. It requires a sorting component when applied in a multi-sub-channel switch, which is too costly to implement in hardware. Instead we use a round-robin based allocation. The set-up clock frequency is increased from 570 MHz [2] to 1.1 GHz (using SMIC 90 nm Tech).

3.3. Operation flow

Our circuit switched network has six operations which are explained in Fig. 4. In stage 1, a probe carrying setup information is sent out from source node and moving towards the destination

Table 2
Probe format.

6-Bit src. addr	6-Bit dest. addr	2-Bit channel id



(a) In each node a probe may double. (b) When two probes meet, one is cancelled.

Fig. 3. The overview of the parallel probing algorithm.

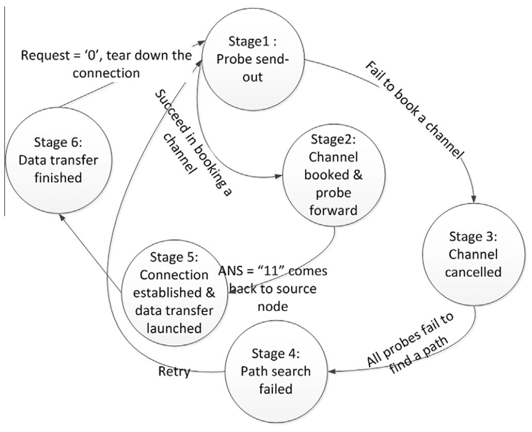

Stage1 : Probe send-out
Request = '0', tear down the connection
Fail to book a channel
Succeed in booking a channel
Stage2: Channel booked & probe forward
Stage 6: Data transfer finished
Stage 3: Channel cancelled
Stage 5: Connection established & data transfer launched
ANS = "11" comes back to source node
All probes fail to find a path
Stage 4: Path search failed
Retry

Fig. 4. Operation flow of parallel probing method.

according to the following algorithm. At each hop, if there are free channels, the probe will book one channel and move forward (stage 2). Otherwise, the probe is failed and it will use ANS signal (Nack) to clear all the channels it has already been booked (stage 3). When the source node gets notice that all the sent out probes have failed (stage 4), it will retry again. If a probe successfully reaches its destination, the Ack signal will be sent back, which means the connection has been established and data transfer can be launched (stage 5). After data transfer finished (stage 6), connection will be torn-down. It will then go back to stage 1 to wait and serve new setup request.

3.4. Detailed switch architecture

The internal structure of a switch is shown in Fig. 5. It is divided into two parts: control path and data path. The data path transfers data through the configured data crossbar. The control path is used to set up or tear-down a data path. The control path and data path share the same input and output wires.

We designed an allocator to do channel allocation for the probes arriving simultaneously at a switch. The principle of our single cycle maximal allocator with round-robin fairness is similar to a wave-front allocator [7], but it is smaller, fairer, faster and free of combinational loops. Detailed description of this allocator is described in our work [8].

It should be noted that the area of this allocator increase with $O(n^2)$, the critical path length scales $O(n)$, with n being the number of channels per direction. For example, in a 1-channel-per-direction switch, the allocator in charge of each output direction consists of only 4 tiles, while in a

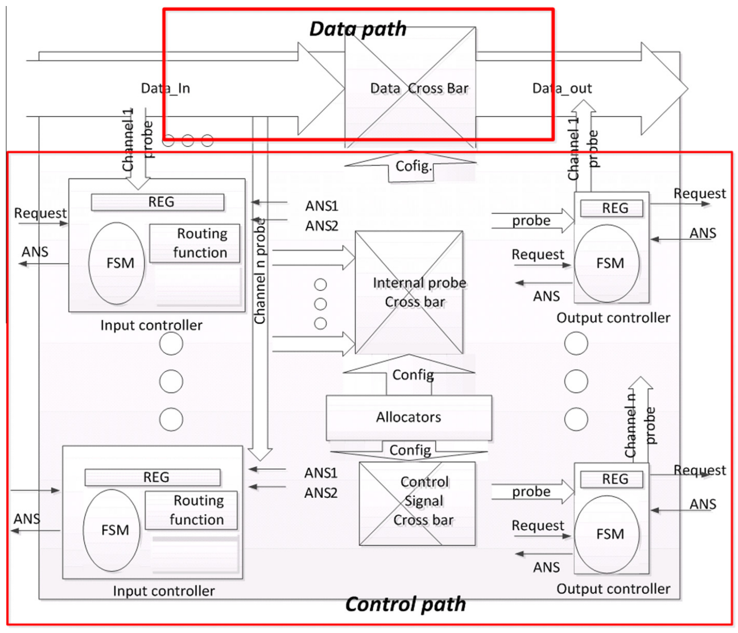


Fig. 5. Internal structure of a switch.

4-channel-per-direction switch, the allocator per direction contains 64 tiles.

3.4.1. Source synchronized data transfer

The control path and data path can work at different clock frequencies and share the same wires without interference. This clock scheme takes advantage of the property of circuit switched NoC that the setup phase never overlaps with the data transfer phase. During the connection setup phase, the data path should have no active clock signal, thus it is idle. And the cross-bar of the data path can be configured under the control path clock (probe clock). During the data transfer phase, the control path ignores data variations on the shared wire links. It just listens to the request and ANS signals.

Therefore, we can utilize either source synchronous data transfer [9–11] or clock gating to realize this separation of data and control path clock schemes, so that the data transfer can benefit from a higher clock frequency. In this paper, we chose the former source synchronous data transfer. The usage of this technique on CS NoC has been justified by Pham et al. [11,6].

3.4.2. Predictable delay

One of the benefits of the probing set-up approach is predictable latency. In our design, each probe takes 2 clock cycles per hop, and the ANS signal takes 1 cycle per hop. So, it takes at most $3 * D + 4$ cycles for a probe to travel from source to destination and back the ANS signal (D is the hop distance between source and destination). 4 cycles is the overhead consumed in the source and destination nodes. Therefore, in an $n * n$ mesh the worst case for a single search takes $3 * (2 * n - 2) + 6$ cycles, no matter if the result is a success or a failure. There is no such bound for the packet configuration approach such as [12]. It is reported that it takes 76 cycles on average for 6 hop distances [12], while it is fixed to 22 cycles by using our probing approach.

For data transfer, the head flit takes two cycles per hop, and the following flits are pipelined.

3.4.3. Configurable sub-channels and sub-networks

Even though the total number of wires between switches is the same in different configurations, their costs and performances are different. Fig. 6 depicts configurations sub1_ch2 and sub2_ch1. Their intra-switch connection relationship of channels is unveiled by using a switch block diagram, in which a line denotes a bi-directional connection between two duplex channels. For example, in the multi-sub-channel sub1_ch2 case, an output channel can be connected to all input channels except to the ones of the same direction. Since each output channel needs to select from 8 input channels, which means an 8-to-1 multiplexer is required by each output channel, and thus the switching logic in total has ten 8-to-1 multiplexers. However, in the multi-sub-network sub2_ch1, a channel is restricted to connect to the channels of the same sub-network. Thus, each output channel just needs a 4-to-1 multiplexer and the entire switching logic is just made up of ten 4-to-1 multiplexer. As a result, for a given number of wires, although sub-channel configuration offers more switching flexibility [13] than sub-network, its switching logic is much more complicated.

4. Connection building schemes in MultiCS

Since a network interface has access to all available sub-networks and sub-channels (Fig. 6), we have the flexibility to build a wide connection with one or more channels per hop to deliver a packet, rather than only one-channel-per-hop. For example, in a sub2_ch2 network, each resource node is connected to 4 input–output channel pairs, so at most it can use all its 4 output channels for one connection. In order to set up a connection with 4-channel width, the resource node has to send 4 setup probes

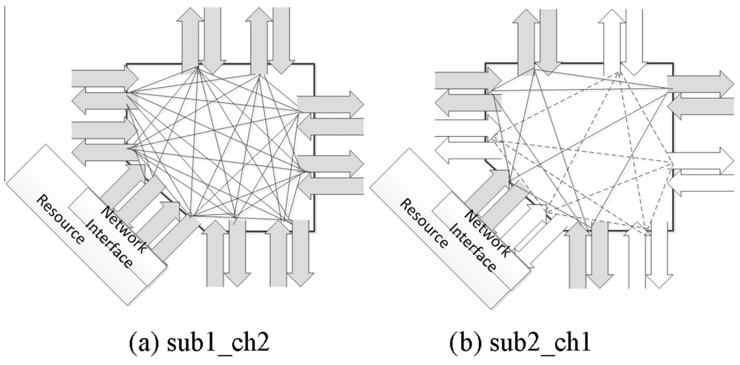


Fig. 6. Switch block diagram of sub1_ch2 and sub2_ch1.

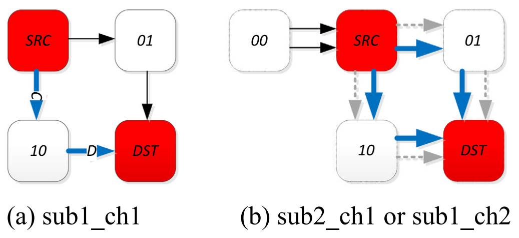


Fig. 7. Illustration for channel fragments.

out through its 4 output channels. Each probe will try to set up a narrow connection of 1-channel width to the destination. After the success of all 4 probes, a 4-channel connection is established.

We propose two schemes to explore the opportunities and challenges of building connections with multiple channels. We name the two schemes as deterministic channel allocation (DCA) and adaptive channel allocation (ACA), respectively.

- *DCA:* DCA imposes mandatory requirement on the connection width. For example, if a packet has a connection width requirement of 4 bytes, it is restricted to use two 2-bytes channels per hop, or four 1-byte channels per hop to build up a connection; any allocation below or above this figure is unacceptable.
- *ACA:* ACA scheme has no hard connection width requirement. During the setup phase, a setup request tries to utilize as many channels as possible to build a connection. However, the final connection width is determined by the number of successful setup probes, which depends on the run-time congestion situation of the network.

The DCA scheme is intended to provide desired and predefined throughput and flit width for data transfer. It imposes strict requirement on the setup phase. DCA scheme is useful in the circumstances when an end-to-end transfer has exact predefined throughput or flit width requirement.

The ACA scheme is designed to achieve better performance, at the expense of only minimum throughput and flit width (1-channel width) guarantee. Depending on channel use, ACA adaptively sets up a connection, of which the width of a connection can vary from 1 to *k* channel-width, where *k* is the total number of output channels per direction. As a result, at low load connections are likely to be wide and thus data transfer time can be shortened; at high load connection width tends to be narrower because of contention. Thus, more requests can be served and high throughput can be reached.

The traditional one-channel-per-connection (OCPC) scheme is a special case of DCA scheme (DCA with 1-channel width requirement). In OCPC scheme, the width of every connection is restricted to 1-channel width.

The implementation of DCA may introduce additional steps during setup. For example, if a packet in a sub4_ch1 or sub1_ch4

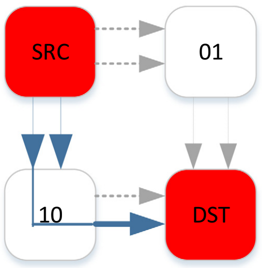


Fig. 8. Illustration for a superfluous connection.

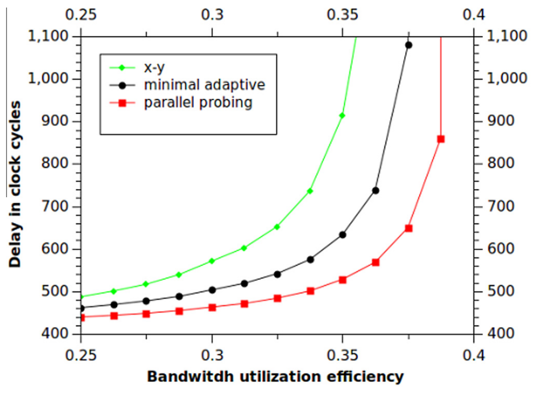


Fig. 9. Evaluation of path searching algorithm by using sub1_ch4 in ACA-FPS (packet size 5120 bytes).

configuration has a DCA requirement of 8-byte width, and each channel is 2-byte wide, the resource node has to wait until four 2-byte output channels of the network interface are available, and then four probes are sent out simultaneously. Only if all of the probes succeed, the data transfer can commence. Otherwise, the one-channel connections (*superfluous connections*) established will be released and all four probes will be re-sent again until all four succeed. The release of superfluous connections is adopted to avoid deadlock.

The implementation of ACA takes advantage of the predictable set-up delay of MultiCS. For each setup request, a resource sends out probes through all free output channels of the network interface, and each probe tries to build up a one-channel connection. After one probe succeeds, the sender will wait a short period ($<(3 * D + 4)$ cycles) for all outstanding probes to return their results, then combine all the established one-channel connections together to form a wide connection for transfer.

5. Cost and performance analysis

5.1. Implementation cost

Suppose k is the number of sub-channels of a switch and m is number of sub-networks (*subm_chk*), and given that the total number of wires of a switch is a constant.

The critical latency of the data path of a switch is chiefly decided by the crossbar latency, which scales with $O(\log k)$, and it is independent of m.

The area of a data cross-bar scales $O(k^2)$ with sub-channels, and it is again independent of m. The registers inside the data path take a large part of area, but their number is independent of k and m.

To the latency of the control path, the allocator contributes $O(k)$ latency and the cross-bar contributes $O(\log k)$. Combining both we see that latency scales $O(k)$ with sub-channels. Again, the latency is independent of the number of sub-networks m.

The area of control path scales $O(m)$ with sub-networks and $O(k^2)$ with sub-channels. This is because using sub-networks causes a linear increase of certain components, e.g. FSMs. However, using sub-channels will cause certain components, e.g. allocator, have a k^2 increase.

The synthesis results of a few configurations are listed in Table 3 with SMIC 90 nm library. The number of wires per-direction of each configuration is the same, i.e. 8 bytes. The total power and area per node is reported by Design Compiler and calculated at each one's maximum clock frequency.

Generally speaking, synthesis results are in accordance with our expectation. For example, sub1_ch1 has the smallest area and power consumption, and can work at the fastest clock frequency. Sub4_ch1 has the same frequency as sub1_ch1, while sub1_ch4 consumes the largest area because it has an $O(k^2)$ increase in area, and works at the slowest clock frequency due to its $O(k)$ latency scale factor.

5.2. An analytical performance model

We propose a model for per-node maximum throughput analysis. We assume that every node inside a network has the same behavior and the network achieves the maximum throughput when a node is always busy in requesting connection setup or transferring data. This means that there is no idle time.

Suppose t_0 is the time used for data transfer when a connection has been established, t_1 is the time consumed for a single search (it has a bounded value in our approach), α is the failure rate ($1-\alpha$ is the success rate). Suppose further that the intensity (average number of certain events per time unit) of successful transfers of a node is $\lambda(A)$, the intensity of a single search is $\lambda(B)$. Based on the conclusions of Palm Calculus [14], we have

$$\frac{\lambda(A)}{\lambda(B)} = 1-\alpha \tag{1}$$

The average time between two single searches is $1/\lambda(B)$, which is equal to

$$\frac{1}{\lambda(B)} = (1-\alpha)(t_0+t_1)+\alpha t_1 \tag{2}$$

As there is always either a search or a data transfer going on, $\lambda(A)t_0+\lambda(B)t_1 = 1$.

The average time between two successful searches is $1/\lambda(A)$, and, combining (1) and (2), we obtain

$$\frac{1}{\lambda(A)} = t_0+\frac{t_1}{1-\alpha} \tag{3}$$

According to (3), maximum normalized throughput (bandwidth utilization rate) is

$$THN = \frac{t_0}{t_0+\frac{t_1}{1-\alpha}} \tag{4}$$

Suppose B is the bandwidth of a resource node, then the maximum throughput of each resource is

$$TH = THN * B = \frac{Bt_0}{t_0+\frac{t_1}{1-\alpha}} \tag{5}$$

This simple model can explain the following intuitions:

I. As the packet size increases, t_0 goes up and thus TH goes up, i.e. CS NoC becomes more efficient with larger packets.
II. Assume a fixed packet size of M (bytes) and a total bandwidth B of a resource node. If each node just has one channel, then the required time for data transfer is $t_0 = M/B$, and from (4) we obtain the normalized throughput THN. However, if we allocate the total bandwidth into two channels, with each one $B/2$ the bandwidth, then the data transfer time for a packet become $t'_0 = 2t_0$, and the normalized throughput becomes

$$THN' = \lambda(A)^{*}t'_0 = \frac{2t_0}{2t_0+\frac{t_1}{1-\alpha}} \tag{6}$$

Thus, splitting a wide channel into narrow channels increases the maximum throughput. This conclusion is less expected but can intuitively be explained by the fact, that if the channel bandwidth is smaller, the penalty of not using this bandwidth during setup is also smaller.

This model is a simplification. For example, in reality, α in formula (4) is not a constant. It depends on a number of factors such as t_0, topology, sub-network/sub-channel configuration, connection build-up schemes and so forth. Although the model is fairly simple we will see in the subsequent simulation results, that the main conclusions are confirmed.

5.3. Analysis of connection building schemes

The behavior of ACA in general follows our previous intuitions, as we will see in Section 6, e.g. Fig. 10.

For DCA, however, contrary to our intuition II, under certain circumstances, the throughput of multiple narrower channels is inferior to a single wide channel. Two phenomena degrade the performance of DCA.

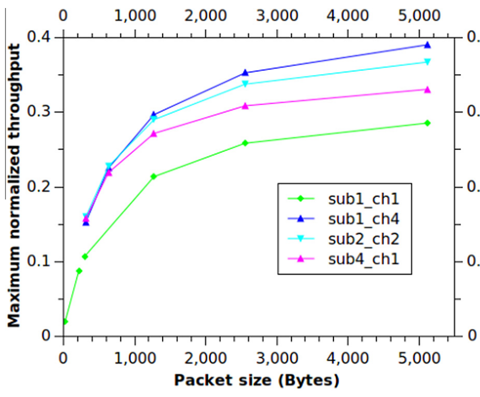


Fig. 10. Influences of packet size on maximum throughputs.

5.3.1. Channel fragments

As Fig. 7 illustrates, in multiple channel configurations such as sub2_ch1 or sub1_ch2, a DCA requirement demands a 2-channel connection and such a connection is possibly built by using channels from two distinct routes[1] (Fig. 7b)). This kind of channel allocation will generate channel fragments. Although there is a free channel between nodes SRC and 01, it cannot be utilized by node 00 to build a connection with 2-channel requirement to node 01. In comparison, single channel configuration sub1_ch1 has the double channel width of sub2_ch1 or sub1_ch2. Thus, it can build a one-channel connection to satisfy the same DCA requirement. There is no fragment in sub1_ch1 under such a DCA requirement. Channel fragments are the result of increased flexibility when given a higher number of narrower channels. But for certain traffic scenarios this increased flexibility is doing more harm than good, as we will see in Fig. 15.

5.3.2. Superfluous connections

Connections that are setup but cannot be used, are superfluous. Suppose we have a CS NoC as sub2_ch1, and each channel is 2-byte wide. For example, as Fig. 8 suggests, a connection with 4 bytes width requirement needs to send out 2 probes and set up two one-channel connections simultaneously with DCA scheme. However, since some of the channels have already been occupied, only one one-channel connection can be established. As a result, since the DCA requirement cannot be satisfied, data transfer cannot be launched and thus the established one-channel connection becomes superfluous. The superfluous connection will be released and then a new setup is attempted. However, the reserve and release of these superfluous connections inside a CS NoC puts a heavy burden on the network and degrades performance.

6. Experiments and evaluations

In this section, we will check whether experiment results are in accordance with our design goals, intuitions and analysis. All experiments are based on 8×8 mesh topology. Uniform random traffic with Poisson arrival time distribution is used for evaluation purpose.

In addition to the four configurations in Table. 3, configuration sub2_ch1 is also used in our experiments. This configuration has two sub-networks, each of which has a channel width of 4 bytes, so that the total channel width per-direction is also 8 bytes. The clock frequencies of sub2_ch1 are the same as sub4_ch1.

We use two scenarios which use ACA and DCA, respectively, to compare the performance of different number of sub-channels and sub-networks, as well as the path searching algorithms. In both scenarios we include several test cases, such as fixed packet size case (FPS) (all packets have the same size), variable packet size case (VPS) (all packets have random packet sizes). However, since FPS and VPS have similar results, we just show the results of FPS.

When nothing else is specified, we use the parallel probing algorithm by default.

6.1. Simulation method and metrics

Inside each resource node a generator generates setup requests according to a probability and pushes them into a queue. An FSM pops a request from the queue and sends it out when sufficient output channels are available. Then the FSM waits for a success or failure notification. Then it either retries the request or commences the data transfer.

We have implemented an HDL model for synthesis and for area and power evaluation. We have also built a cycle accurate SystemC simulator which can run 10–30 times faster than the HDL model. Any data point that is shown in the figures comes from simulation of 250 million cycles, of which the first 250 thousand cycles are discarded as warm up period.

Suppose β is the packet generation probability and M is the packet size (in bytes), and clk_{freq} is the data path clock frequency of a configuration, B is the bandwidth of a resource node, then the injection rate per node (IR) is defined as

$$IR = \beta * M * clk_{freq}$$

Besides throughput and delay, we use *bandwidth utilization efficiency* (Eb), also called *normalized throughput*, as one of the metrics. It is defined as

$$Eb = \frac{Throughput\ (per\ node)}{Bandwidth\ (per\ node)}$$

In our simulations each configuration operates at its maximum frequency.

6.2. ACA (adaptive channel allocation) evaluation

6.2.1. Evaluation of path searching algorithms

Three path searching algorithms are compared, which are *x–y* [15], minimal adaptive [15] and parallel probing [2]. The results in Fig. 9 suggest that parallel probing is the best path searching algorithm for ACA scheme. E.g. at offered load 0.35, the average packet delay of parallel probing is only 83% of minimal adaptive, and 57% of *x–y* algorithm. We also have evaluated algorithms in different channel number and configurations. Their results suggest the same ranking of algorithms. Consequently, we choose parallel probing as our default path searching algorithm.

6.2.2. Influences of packet size on maximum throughputs

The influence of packet size on maximum normalized throughput is shown in Fig. 10, which suggests that as packet size increases, the maximum normalized throughput for each configuration also goes up. This result complies with intuition 1 from our model. Thus, we may safely conclude that CS NoC is suitable for delivering large packets. This is the reason why throughput this paper we prefer large packets for evaluations. This conclusion implies that applications that generate large bulk of data for communication, like task allocation and migration on MPSoC, or page

[1] In this situation, each flit will be split into two phits at the source, with each route simultaneously delivering only one phit. At the receiver side, it will restore a flit by combining the two phit together.

Table 3
Per-node synthesis results of different CS NoCs with 8 bytes of wires per-direction.

Configuration	Sub1_ch4	Sub2_ch2	Sub4_ch1	Sub1_ch1
Channel width	2 Bytes	2 Bytes	2 Bytes	8 Bytes
Num. sub-network	1 (Sub1)	2 (Sub2)	4 (Sub4)	1 (Sub1)
Num. sub-channel	4 (Ch4)	2 (Ch2)	1 (Ch1)	1 (Ch1)
Max. probe freq. (MHz)	556	740	1111	1111
Max. data freq. (GHz)	1.116	1.397	1.786	1.786
Total area (um^2)	150214.5	86777.5	57599.5	30874.9
Probe path area	85791.8	53161.3	35632.8	8908.2
Data path area	64422.7	33616.2	21966.7	20133.7
Power@max. freqs. (mW)	50.1	45.0	49.5	26.8

based virtual memory management, benefit from CS NoCs, while applications with mostly short messages may prefer PS NoCs, as concluded in [16].

6.2.3. Evaluation of different number of channels

The experiment results of splitting a wide channel into narrow sub-channels is shown in Figs. 11 and 12. The packet size is fixed to 5120 bytes in this evaluation.

As shown in Fig. 12, sub4_ch1 provides higher throughput than sub2_ch1, which in turn is better than sub1_ch1. E.g. the maximum throughput of sub4_ch1 is about 17% higher than sub1_ch1. The increase in maximum throughput complies with our intuition II. We can imagine that some packets in the sub4_ch1 configuration use only 1, 2, or 3 of the subnetworks. However, e.g. using 1 sub-network with ¼ the channel width compared to the sub1_ch1 configuration means that the packet consists of 4 $\times$ the number of flits. As we studied in the last section, larger packet sizes lead to higher maximum throughput in CS NoC. Thus, using narrow sub-links will achieve higher maximum throughput because the average packet size counted in flits is also larger.

Regarding delay, as Fig. 11 suggests, sub1_ch1 has better packet delay results only when the injection rate is low. This is due to the connection setup delay contributes little to the total packet delay because of low contention probability. In this situation, data transfer delay dominates the total packet delay. Sub1_ch1 has shorter data transfer delay due to its wider channel. However, at high injection rate, sub4_ch1 outperforms sub1_ch1. For example, at injection rate 3500 MB/s, the average packet delay of sub4_ch1 is 20% less than sub1_ch1.

If throughput is our main concern, the number of channels should be maximized. However, in our design, the minimum channel width is decided by probe format, which is about 14 bits. Narrower than this value complicate the probe delivering process.

6.2.4. Evaluation of different configurations (Fig. 13)

Although sub1_ch4 has more switching flexibility than sub4_ch1, this advantage is compensated by its slower clock frequency. As a result, sub-network configuration (sub4_ch1) outperforms sub-channel configurations (sub2_ch2 and sub1_ch4) in delay and throughput. Sub1_ch1 has lower maximum throughput than the other multi-channel configurations. However, it presents better latency at low load for the same reasons explained above.

Bandwidth utilization efficiency discounts the difference in frequency and gives a performance comparison under the assumption that the networks operate at the same frequency. Sub1_ch4 has the best bandwidth utilization efficiency under ACA scheme. For example, we observed 30% higher efficiency than sub1_ch1 in ACA scheme. Sub2_ch2 and sub4_ch1 fall in between.

Bandwidth utilization efficiency may also be useful because in certain situations the maximum clock frequency differences of configurations are not sharp. For example, as reported in [25], when implemented in FPGA, a CS NoC with SDM channels roughly has the same maximum clock frequency no mater if 1 or 4

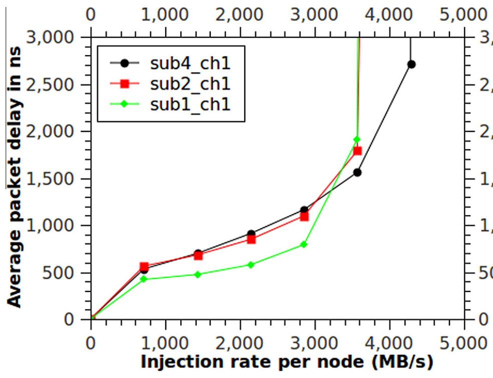


Fig. 11. Delay influence of dividing a wide channel into narrow sub-channels (packet size 5120 bytes).

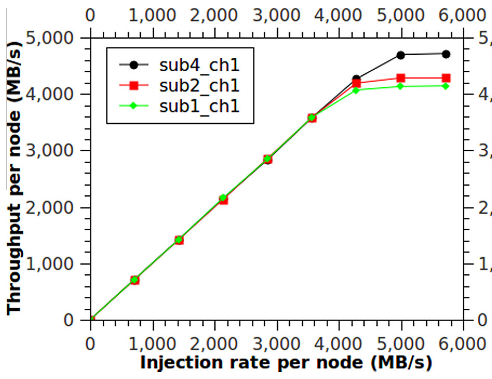


Fig. 12. Throughout influence of dividing a wide channel into narrow sub-channels (packet size 5120 bytes).

sub-channels are used. In situations like this, sub1_ch4 could offer better performance than other configurations.

We also tested under ACA scenarios with variable size of packets. The comparison among different configurations basically shows consistent results and is thus omitted here.

6.2.5. Comparison between ACA scheme and one-channel-per-connection (OCPC) scheme

OCPC is compared with ACA by using configuration sub4_ch1, as Fig. 14 suggests, at low load ACA offers much better average packet delay. E.g. at load 0.02, average packet delay with ACA is 170 probe clock cycles, while it is 490 cycles with OCPC. At very high load, OCPC represents slightly higher bandwidth utilization efficiency, and its maximum bandwidth utilization efficiency is 0.283, while for ACA it is 0.271.

The comparison result obeys our design goals of ACA in Section 4. At low load when data transfer delay dominates, ACA can significantly shorten the delay since the majority of packets are delivered by wide connections. At high load, due to contention, the probability of building a connection containing multiple channels is low and the majority of connections contain one channel only. Thus, the average packet length in flits by using ACA at high load is just slightly smaller than using one-channel-per-connection

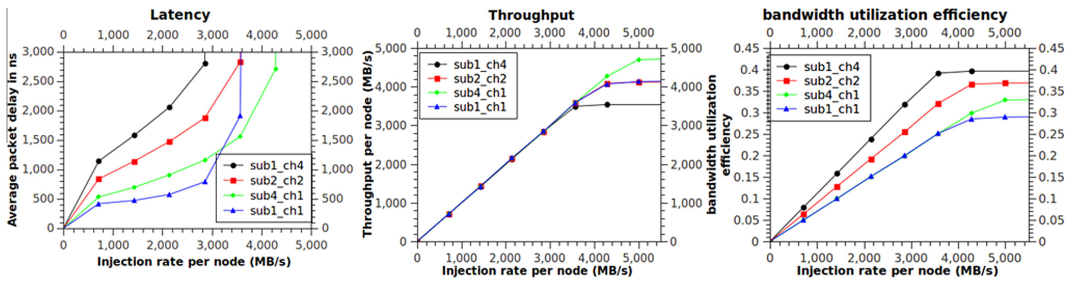


Fig. 13. Performance results of scenario ACA–FPS (packet size 5120 bytes).

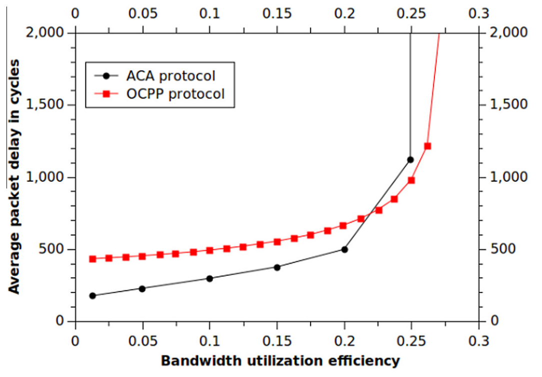


Fig. 14. Comparison between ACA scheme with traditional OCPC scheme (packet size 1280 bytes).

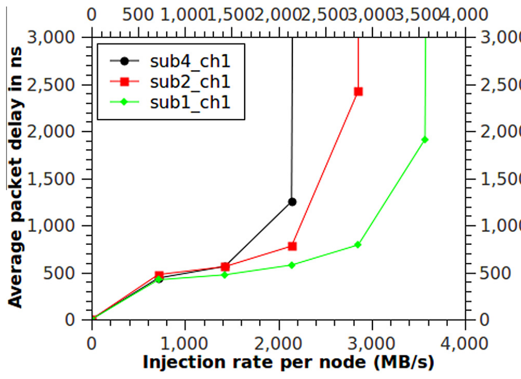


Fig. 15. Influence of dividing a wide channel into narrow sub-links for DCA transfer (packet size 5120 bytes).

scheme. As suggested by intuition II, the maximum bandwidth utilization of ACA drops slightly (smaller than 5% in this experiment).

6.3. DCA (deterministic channel allocation) evaluation

As mentioned before, ACA puts more emphasis on performance, while DCA focuses on desired throughput and flit width.

6.3.1. Evaluation of different number of channels

Previously, we showed that ACA benefits from the usage of multiple channels. However, the following example with DCA tells a different story.

Fig. 15 shows the result of a simulation with the exact connection width requirement of 8 bytes and a fixed packet size of 5120 bytes. In this case, as the latency curves suggest, more channels lead to higher delay.

This observation complies with our analysis in Section 5.3. Sub4_ch1 and sub2_ch1 perform worse than sub1_ch1, since sub1_ch1 generates neither channel fragments nor superfluous connections. Sub4_ch1 performs worse than sub2_ch1 because it generates more superfluous connections.

In addition, it is worth noting that, if less than half of the bandwidth can be utilized, splitting the wide channel into sub-links seems to be beneficial, even without special care on channel fragments and superfluous channels. In Fig. 16, the exact throughput requirement is 4 bytes/cycle. Because only half of the channel width in sub1_ch1 can be utilized for data transfer, sub1_ch1 is inferior to the other multi-channel configurations. This result also suggests that, according to the connection width requirement, proper channel partitioning could still be beneficial.

6.3.2. Evaluation of different configurations

For delay and throughput, both Figs. 16 and 17 demonstrate that sub-network configuration sub4_ch1 outperforms sub-channel configurations sub2_ch2 and sub1_ch4. These results are similar to those under ACA. However, in Fig. 17, the channel utilization efficiency of sub1_ch4 is worse than sub2_ch2, which is worse than sub4_ch1. This observation opposes the result with ACA and is not quite expected. It seems that switching flexibility becomes a handicap in this DCA case.

The reason for this phenomenon is that sub1_ch4 and sub2_ch2 are more likely to generate superfluous connections. Due to the increased switching flexibility, sub1_ch4 and sub2_ch2 have higher chances to set up a one-channel connection, which leads to a higher burden on the network due to set up and release of superfluous connections.

Generally speaking, as the comparison of Figs. 16 and 17 with Fig. 13 suggests, we may conclude that ACA offers better performance than DCA scheme. However, as mentioned before, DCA offers exactly predefined connection width and throughput.

We also tested DCA scenarios with variable size of packets. The comparison among different configurations basically shows consistent results and is thus omitted here.

7. Related work

The usage of sub-network and sub-channel in PS NoC has been studied during the past. For example, the cost and effect of intro-

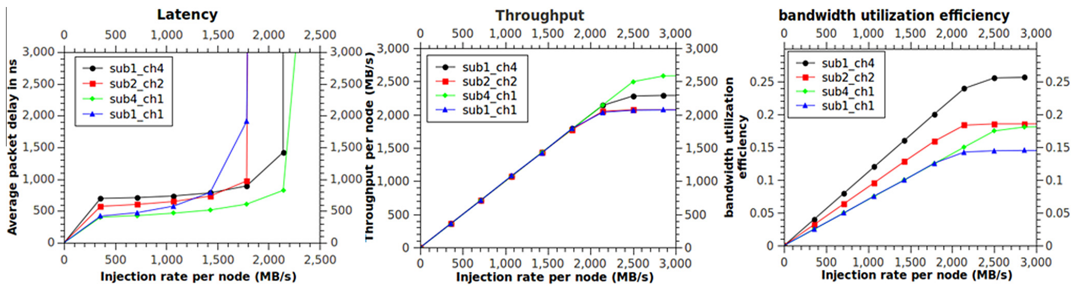


Fig. 16. Performance results of scenario DCA-FPS (packet size 2560 bytes, connection width requirement is 4 bytes).

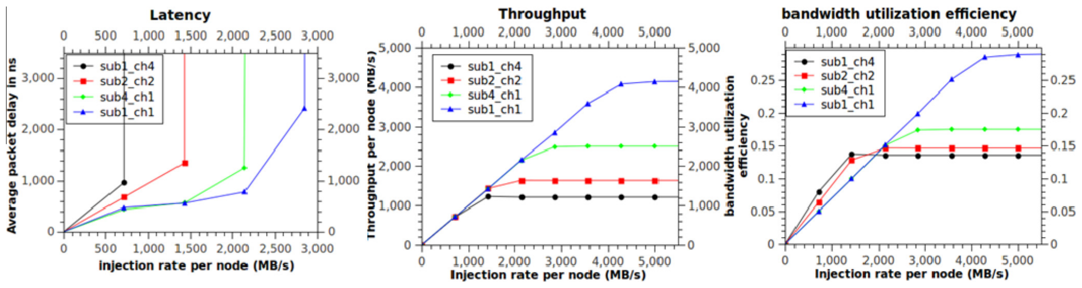


Fig. 17. Performance results of scenario DCA-FPS (packet size 5120 bytes, connection width requirement is 8 bytes).

ducing sub-networks into PS NoC has been studied by Yoon et al. in [17,18]. The pros and cons of using sub-channels in PS NoC has also been investigated in [3]. Besides, work [19,20] intend to increase the switching flexibility between virtual channels and the output ports of PS NoCs. In [19], several separate cross-bars are used inside one router, so that each virtual channel (VC) can choose between multiple crossbars to reach an output. In [20], a new switching layer is introduced at each input port, so that multiple VCs of an input port can be connected to different outputs at the same time.

However, compared with PS NoC, the usage of sub-network and sub-channel in CS NoC is not fully exploited and evaluated. Actually, in the past, the CS NoC architecture assumed in many papers (e.g. [2,11,21]) has just one duplex-channel between every two neighboring nodes. They did not consider the situation when a CS NoC can have multiple physical channels between two nodes.

Although some works [4,22] design CS NoCs with multiple channels and organize them in a sub-channel (SDM) way, the consequences of applying multiple channels in CS NoCs are still not well studied. For example, although [4,22] have multiple channels, packets are still delivered by following connections with only 1-channel width.

Another import aspect about CS NoC is connection setup, since a CS NoC requires a connection should be established before data transfer begins. According to the connection search and setup method, CS NoCs can be classified into two categories: dynamic setup or static setup. Static setup methods schedule connections at compilation time. As a result, they [23,24] may not well support applications like H.264 [25] with requirements for dynamic communication setups. Therefore, in this paper, we only focus on dynamic methods which search and setup connections at run time.

Dynamic methods can be further classified into centralized or distributed methods. Generally speaking, centralized set-up like [21,26] has two disadvantages. Firstly, the central schedule node needs to receive setup/release requests and distribute allocation decisions from/to the entire network. Such multiple-to-one and one-to-multiple traffic pattern is likely to become the system bottleneck which the number of nodes inside a NoC grows [27]. Secondly, since retrying of failed requests causes the blockage of the following requests, failed setup requests are usually dropped in centralized setup methods. Thus, we focus on decentralized setup.

Distributed setup can be implemented by sending configuration packets [4,28,29] or by a probing search approach [1,11,6,5].

Sending configuration packets requires a separate PS (packet switched) NoC to deliver configuration messages like set-up, tear-down and Ack/Nack during a connection setup procedure. In our view, this approach suffers from four major drawbacks. Firstly, using an additional PS NoC for connection set-up is an unnecessary overhead. Secondly, set-up, tear down and Ack/Nack packets of a connection must be routed by pre-determined routing algorithm to ensure them on the same connection. For example, [4,29] use deterministic routing algorithm, and in [28], source based routing information has to be carried by each configuration packet. However, such pre-determined routing algorithm is a sub-optimal choice among routing algorithms. Thirdly, compared with probing search, tear-down and Ack/Nack signals have to be sent in the form of packets. These packets will contend with set-up packets inside the PS NoC. There is typically no delay guarantee for configuration packets in the PS network, rendering the connection set-up procedure unpredictable. Fourthly, this approach does not scale well. The auxiliary PS NoC has fixed throughput, since each output port of a switch just allows to deliver one setup packet at a time. However, if there are many sub-channels in a CS NoC, and since each sub-channel requires a separate setup packet for connection configuration, this will significantly increase the number of setup packets as observed in [4].

Compared with above mentioned shortcomings of a packet configuration approach, probing search is the superior choice because of its efficiency in wire usage and connection setup procedure. The concept of the probing search was first proposed in [5]. Pham et al. [11,6] developed a backtracking path searching algorithm, which reportedly has better performance than [5]. Another contribution of [11,6] is that a source synchronized data transfer mechanism is introduced into CS NoCs, so that separate clocks can be applied to connection set-up and data transfer. [2] developed a parallel probing method for CS NoC. It can complete a search over all possible paths within $O(n)$ time complexity where n is the geometric distance between source and destination. They demonstrated superior performance of this parallel probing algorithm compared to Pham's backtracking algorithm [11] by experiments. But their channel allocation mechanism [2] is too complicated for multi-sub-channel usage.

The probing search approaches in all aforementioned works [1,11,6,5] are only implemented on CS NoC with a single channel between two neighboring nodes.

In this paper, we extend the parallel probing search method [2] to multiple sub-channels and sub-networks and study cost and performance of several configurations with sub-channels and sub-networks among 1 and 4.

8. Conclusion and future work

We have implemented MultiCS, a CS with multiple sub-channels and sub-networks with a parallel probing setup algorithm to study the consequences of splitting a wide channel into narrow channels. The design space of multi-channel CS NoC is explored from two angles: the channel number, and channel configurations. We have reached the following main conclusions:

A. Given a number of wire resources for each node inside a CS NoC, with ACA scheme, the thinner the channel width with more channels, the higher the throughput. However, the latency for data transfer also increases by using thinner channels. Sub-channels (SDM channels) consume much more resources than sub-networks. When splitting a wide channel into n narrow channels, organizing those channels into sub-networks gives an $O(n)$ increase in area, and the critical latency is unchanged. However, organizing those channels in sub-channels increases the area by $O(n^2)$, and the delay by $O(n)$. Furthermore, our experiments suggest that sub-networks offer better performance than sub-channels. Although sub-channels can achieve better channel efficiency due to higher switching flexibility, this is only useful in special situations. Thus, in general sub-networks are more efficient than sub-channels.

B. We can build a connection consisting of multiple channels with different schemes. The DCA offers desired and predefined throughput and flit width, but channel fragments and superfluous connections are two obstacles for DCA. Because of this, under certain width requirements, the performance of using multiple channels is even worse than using one single wide channel. ACA generally offers better performance than DCA. However, although ACA provides minimum connection width guarantee (one channel width), the actual width of a connection by applying ACA cannot be known beforehand. The connection width is decided by the success probability of setup probes, which depends on the dynamic channel use.

Our future work will study techniques to avoid channel fragments and superfluous connections, in depth evaluation of multi-channel CS NoC, and implementation and evaluation of mixed packet and circuit switched NoCs.

References

[1] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, D. Verkest, Spatial division multiplexing: a novel approach for guaranteed throughput on NoCs, in: Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, 2005, pp. 81–86.

[2] S. Liu, A. Jantsch, Z. Lu, Parallel probing: dynamic and constant time setup procedure in circuit switching NoC, in: proceedings of Design, Automation Test in Europe Conference Exhibition (DATE'12), 2012, pp. 1289–1294.

[3] C. Gomez, M. E. Gomez, P. Lopez, J. Duato, Exploiting wiring resources on interconnection network: increasing path diversity, in: Proceedings of Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'08), 2008, pp. 20–29.

[4] A.K. Lusala, J.-D. Legat, Combining SDM-based circuit switching with packet switching in a router for on-chip networks, Int. J. Reconfigurable Comput. 2012 (2012) 1–16.

[5] D. Wiklund, D. Liu, SoCBUS: switched network on chip for hard real time embedded systems, in: Proceedings of Parallel and Distributed Processing Symposium, 2003, p. 8.

[6] P.-H. Pham, P. Mau, J. Kim, C. Kim, An on-chip network fabric supporting coarse-grained processor array, IEEE Trans. Very Large Scale Integr. VLSI Syst. 21 (99) (2013) 178–182.

[7] D.U. Becker, W.J. Dally, Allocator implementations for network-on-chip routers, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009, pp. 52:1–52:12.

[8] S. Liu, A. Jantsch, Z. Lu, A fair and maximal allocator for single-cycle on-chip homogeneous resource allocation, IEEE Trans. Very Large Scale Integr. VLSI Syst. 22 (10) (2014) 2229–2233.

[9] D. Walter, S. Hoppner, H. Eisenreich, G. Ellguth, S. Henker, S. Hanzsche, R. Schuffny, M. Winter, G. Fettweis, A source-synchronous 90 Gb/s capacitively driven serial on-chip link over 6 mm in 65 nm CMOS, in: proceedings of Solid-State Circuits Conference Digest of Technical Papers (ISSCC'12), 2012, pp. 180–182.

[10] D. Schinkel, E. Mensink, E. Klumperink, E. van Tuijl, B. Nauta, Low-power, high-speed transceivers for network-on-chip communication, IEEE Trans. Very Large Scale Integr. VLSI Syst. 17 (1) (2009) 12–21.

[11] P.-H. Pham, J. Park, P. Mau, C. Kim, Design and implementation of backtracking wave-pipeline switch to support guaranteed throughput in network-on-chip, IEEE Trans. Very Large Scale Integr. VLSI Syst. 20 (2) (2012) 270–283.

[12] A.K. Lusala, J.-D. Legat, A SDM-TDM-based circuit-switched router for on-chip networks, ACM Trans. Reconfigurable Technol. Syst. 5 (3) (2012) 15:1–15:22.

[13] J. Rose, S. Brown, Flexibility of interconnection structures for field-programmable gate arrays, IEEE J. Solid-State Circuits 26 (3) (1991) 277–282.

[14] J.Y. Le Boudec, Performance Evaluation of Computer and Communication Systems, Epfl Press, 2011.

[15] W.J. Dally, B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2003.

[16] S. Liu, A. Jantsch, Z. Lu, Analysis and evaluation of circuit switched NoC and packet switched NoC, in: Proceedings of Euromicro Conference on Digital System Design (DSD'13), 2013, pp. 21–28.

[17] Y. J. Yoon, N. Concer, M. Petracca, L. Carloni, Virtual channels vs. multiple physical networks: a comparative analysis, in: Proceedings of IEEE Design Automation Conference (DAC'10), 2010, pp. 162–165.

[18] Y.J. Yoon, N. Concer, M. Petracca, L.P. Carloni, Virtual channels and multiple physical networks: two alternatives to improve NoC performance, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 32 (12) (2013) 1906–1919.

[19] S. Noh, V.-D. Ngo, H. Jao, H.-W. Choi, Multiplane virtual channel router for network-on-chip design, in: Proceedings of First International Conference on Communications and Electronics (ICCE'06), 2006, pp. 348–351.

[20] F. Gilabert, M.E. Gómez, S. Medardoni, D. Bertozzi, Improved utilization of NoC channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip, in: Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS'10), 2010, pp. 165–172.

[21] M. Winter, G.P. Fettweis, Guaranteed service virtual channel allocation in NoCs for run-time task scheduling, in: Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE'11), 2011, pp. 1–6.

[22] A. Leroy, D. Milojevic, D. Verkest, F. Robert, F. Catthoor, Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip, IEEE Trans. Comput. 57 (9) (2008) 1182–1195.

[23] R. Stefan, A. Molnos, K. Goossens, dAElite: a TDM NoC supporting QoS, multicast, and fast connection set-up, IEEE Trans. Comput. PP (99) (2012) 1.

[24] K. Goossens, J. Dielissen, A. Radulescu, AEthereal network on chip: concepts, architectures, and implementations, IEEE Des. Test Comput. 22 (5) (2005) 414–421.

[25] N. Ma, Z. Lu, L. Zheng, System design of full HD MVC decoding on mesh-based multicore NoCs, Microprocess. Microsyst. 35 (2) (2011) 217–229.

[26] M. Winter, G.P. Fettweis, A network-on-chip channel allocator for run-time task scheduling in multi-processor system-on-chips, in: Proceedings of EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD'08), 2008, pp. 133–140.

[27] S. Liu, A. Jantsch, Z. Lu, Parallel probe based dynamic connection setup in TDM NoCs, in: Proceedings of the Conference on Design, Automation & Test in Europe (DATE'14), 2014, pp. 239:1–239:6.
[28] J. Lim, E. Hunt Siow, Y. Ha, P.K. Meher, Providing both guaranteed and best effort services using spatial division multiplexing NoC with dynamic channel allocation and runtime reconfiguration, in: Proceedings of International Conference on Microelectronics (ICM'2008), 2008, pp. 329–332.
[29] A.K. Lusala, J.-D. Legat, Combining sdm-based circuit switching with packet switching in a NoC for real-time applications, in: Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'11), 2011, pp. 2505–2508.

Shaoteng Liu received the B.Sc. degree from Fudan University, Shanghai, China, in 2006. He received his M.Sc. degree from Royal Institute of Technology (KTH), Stockholm in 2010. He is currently a PHD student at KTH. His current research interests include system modeling, performance analysis, embedded operating system, reconfigurable computing, network-on-chip and software defined network.

Axel Jantsch received the Dipl.Ing. and Dr.Tech. degrees from the Technical University of Vienna, Vienna, Austria, in 1988 and 1992, respectively. He was a professor of electronic system design with the Royal Institute of Technology, Stockholm, Sweden, from December 2002 to September 2014. He is currently a professor in system on chip with TU Wien, Vienna, Austria. His current research interests include VLSI design and synthesis, system-level specification, modeling and validation, HW/SW co-design and co-syntheses, reconfigurable computing, and networks-on-chip.

Zhonghai Lu received the B.Sc. degree from Beijing Normal University, Beijing, China, in 1989, and the M.Sc. and Ph.D. degrees from KTH Royal Institute of Technology, Stockholm, Sweden, in 2002 and 2007, respectively. He is currently an Associate Professor with KTH. His research interests include Network-on-Chip, Embedded Systems, Computer Architecture, and Internet-of-Things. He has published over 130 peer-reviewed papers in transactions, journals and international conferences in these areas.

Paper F

Analysis and evaluation of circuit switched NoC and Packet Switched NoC

Shaoteng Liu, Axel Jantsch and Zhonghai Lu

Presented at IEEE Euromicro Conference on Digital System Design (DSD'13), Santander, Spain, 2013. Included in the proceedings, pages: 21-28.

2013 16th Euromicro Conference on Digital System Design

Analysis and evaluation of circuit switched NoC and packet switched NoC

Shaoteng Liu
Royal institute of technology
liu2@kth.se

Axel Jantsch
Royal institute of technology
axel@kth.se

Zhonghai Lu
Royal institute of technology
zhonghai@kth.se

***Abstract*-Circuit switched NoC has, compared to packet switching, a longer setup time, guaranteed throughput and latency, higher clock frequency, lower HW complexity, and higher energy efficiency. Depending on packet size and throughput requirements they exhibit better or worse performance. In this paper we designed a circuit switched NoC and compared that with packet switched NoC. By speculation and analysis, we propose that, as packet size increases, performance decreases for packet switched NoC, while it increases for circuit switched NoC. By close examination on the router architecture, we suggest that circuit switched NoC can operate at a higher clock frequency than packet switched NoC, and thus at zero load above a certain packet size circuit switched NoC could be better than packet switched NoC in packet delay. Experiment results support our intuitions and analysis. We find the cross-over point, above which circuit switching has lower latency, is around 30 flits/packet under low load and 60-70 flits/packet under high network load**.

I. INTRODUCTION

Packet switched (PS) NoCs (Network on Chip) have been studied more extensively and thoroughly. However, circuit switched NoC (CS) NoC could be preferable under certain traffic patterns and requirements[1]. Thus, it requires an analysis and comparison on PS and CS NoC to reveal their properties and limitations, and offer intuitions for people to make design decisions.

In order to commence such a study, on one hand, for PS NoC, we use an input-buffered virtual channel (VC) wormhole routed PS NoC for comparison. This kind of PS NoC is widely utilized in practice, eg. TILE64[2]. On the other hand, since no classical design exists for CS NoC, we build our own platform by inheriting and integrating the merits from state-of-art works. In addition, mesh topology is used for both NoCs because it is the most popular NoC topology, and it is scalable, easy to layout, and offers path diversity.

In this paper we will show the respective strengths and weaknesses of circuit and packet switched NoCs. Particularly, our work offers following contributions:

- We reveal the detailed mechanisms which make PS NoC not fit for large packets (Section III).
- We suggest that CS NoC with proper design should work at a higher clock frequency than PS NoC (Section VI).
- By analysis on zero load packet delay, we find that above a certain packet size, CS NoC delivers faster than PS NoC (Section VI).
- By experiments and evaluations, we represent the respective favorite working areas of PS NoC and CS NoC. We reveal that if packet size is very large, even with more VCs and buffers, PS NoC still suffers a performance loss (section VIII)

II. RELATED WORK

There are only a few papers on the analysis and comparison of CS NoCs and PS NoCs. In [3][4] some comparisons on area and power consumption are presented. Although [5][6] concern about the performance of PS and CS NoC in a ring topology, the influence of packet size on delay and throughput is not evaluated and compared. The pros and cons of a certain kind of NoCs are still poorly understood. In addition, even though some other works have been done on combining PS and CS NoC [7][8][9][10], but none of them provides a serious analysis and evaluation on their respective characteristics.

PS NoC has been intensively studied by researchers. There are several kinds of PS NoC [11][12][13][14]. Generally speaking, input-buffered virtual channel wormhole routed NoC is well accepted and utilized[2][15]. Dally's book [16] has covered almost every aspect of such a PS NoC. Moreover, many works have focused on optimizations on this kind of PS NoC [12]. Therefore, we will also choose input-buffered virtual channel wormhole routed PS NoC in our paper.

However, there is no well-accepted CS NoC architecture. According to the path search and setup methods, CS NoCs can be classified into two categories: dynamic setup methods [10][9][19][20][21] or static setup methods [22][23]. Static setup methods schedule paths at compilation time. As a result, they may not well support applications like H.264 [24] with requirements for dynamic communication setups. Therefore, we only focus on dynamic methods which search and setup paths at run time.

Dynamic methods can be further classified into centralized [10][9] or distributed methods [1][25][19][20][21][7][9][8][26][27]. Generally speaking, centralized setup has two disadvantages: One is scalability and the other is dropping of failed setup requests [17][18]. Since retrying of failed requests causes the blockage of the following requests, failed setup requests are usually dropped in centralized setup methods[17], [18]. We focus on distributed setup.

`/13 $26.00 © 2013 IEEE
DOI 10.1109/DSD.2013.13

Distributed setup can be implemented by packet configuration [21][1][9][8][27] or by a probing search approach [25][19][20][26].

Packet configuration requires an additional separate PS (packet switched) NoC to deliver configuration messages like set-up, tear-down and Ack/Nack during a path setup procedure. In our view, this approach suffers from four major drawbacks. Firstly, the adding of an additional PS NoC is a unnecessary overhead. Secondly, since set-up, tear down and Ack/Nack packets of a path must be routed by deterministic routing algorithm to ensure them on the same path. However, deterministic routing algorithm is a sub-optimal choice among routing algorithms. Thirdly, compared with probing search, tear-down and Ack/Nack signals have to be sent in the form of packets. These packets will contend with set-up packets inside the PS NoC. Besides, there is typically no delay guarantee for packets in the PS network, rendering the path set-up procedure unpredictable. Fourthly, this approach does not scale well. In PS NoC, each output port of a switch just allows to deliver one packet every time slot. However, if there are several sub-channels in the CS NoC each sub-channel requires a separate setup packet for path configuration, thus significantly increasing the number of setup packets [21].

Compared with above mentioned shortcomings of a packet configuration approach, probing search is the superior choice because of its efficiency in wire resource utilization and path setup procedure. The concept of the probing search was firstly proposed by [26]. In [19][20], Pham et al. developed a backtracking routing algorithm, which reportedly has better performance than [26]. Another contribution of [19][20] is that a source synchronized data transfer mechanism is introduced into CS NoCs, so that separate clocks can be applied to path setup and data transfer. [28] developed a parallel probing method for CS NoC. It can complete a search over all possible paths within O(n) time complexity where n is the geometric distance between source and destination. They demonstrated superior performance of this parallel probing algorithm compared to Pham's backtracking algorithm [19] by experiments.

III. Intuitions and Conjectures

According to our considerations, large packets are detrimental to PS NoC in two ways:

Firstly, large packets will cause burst injection of flits. Bursty traffic may cause massive contentions in a short period, and thus prolongs the average flits delivering delay.

Secondly, large packets will incur unbalanced usage of channels. As illustrated in Fig. 1, we consider the traffics inside an input buffered virtual channel wormhole router by assuming that there is only two virtual channel queues (FIFO queues) and two output directions. Each packet desires either output A or output B for delivering. We suppose that packets with different desires are uniform randomly distributed. However, flits belong to one packet must have the same desired output direction and queued by the same VC. Round-robin allocator is used for allocation. Since it is possible that at a certain cycle both flits from VC1 and VC2 want the same output channel, leaving the other output channel an idle cycle.

As Fig. 1 suggests, when each packet just contains one flit, the span of idle periods are just made up of a few cycles and equally distributed between output flow A and B. However, when packet size increases, the average span of idle periods grows wider and wider.

Although from statistical view, in a very long term, output flow A and B should have the same number of idle cycles; no matter what the packet size is, the total idle cycles should be equal. However, within a short period, we observe that with large packets, output flow A is quite busy, while output flow B is almost idle. Such unbalanced usage of physical channels is destructive to throughput.

In PS NoC, we deal with the burstiness by increasing the buffer depth of a VC, and balance the traffics by adding more VCs. Long packets are detrimental to PS NoC since they increase the needs on both sides. The longer the packets, the more VCs and buffers are required to compensate the performance loss. Unfortunately, both VC and buffer are expensive, especially that adding virtual channels will lower down the clock frequency. Thus, we can imagine that, if packets are long enough, they are almost impossible to be well handled by PS NoC with acceptable cost.

Nevertheless, CS NoCs favor large packets. In CS NoC, after a path has been established, data transfer will be launched. So that the channel utilization ratio for CS NoC is $= \frac{t_{trans}}{t_{setup}+t_{trans}}$.

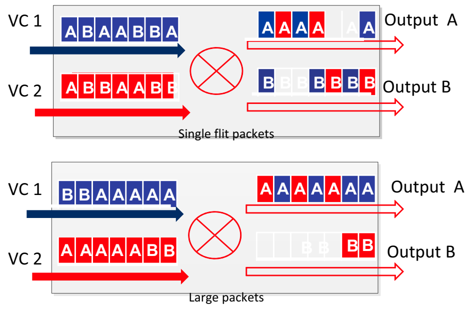


Fig. 1 Unbalance traffic in PS NoC caused by large packets

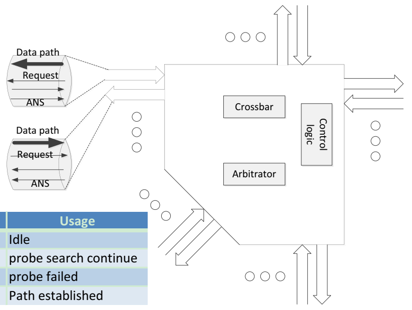
As packet size increases, t_{trans} goes up, so the channel utilization ratio U increase. The per-node throughput, which equals to channel utilization ratio multiplies bandwidth, increases as well.

We may conjecture that, as the packet size increases, the performance curve of CS NoC and performance curve of PS NoC might have a cross point, since one rises and the other falls. However, we need practical parameters such as clock frequency, as well as experiment results, to justify the existence of such a point.

IV. Architecture of our CS NoC

A. Overview of a switch

In this paper, our CS NoC design adopted probing approach with parallel probing routing algorithm [28].



ANS	Usage
00	Idle
01	probe search continue
10	probe failed
11	Path established

Fig. 2 Overview of a switch

As shown in Fig. 2, in a mesh topology every switch has five directions which are used for connecting to four neighbors and one local resource. Each direction may has one duplex channel. The data path of a channel is used for carrying the probe during setup and for transmitting data when a connection has been established. Each probe is one flit in length. Every data path is associated with an answer (ANS) signal consisting of 2 bits, which goes in the opposite direction to the data channel, and 1 bit for a Request signal, which travels in the same direction as the data channel. When the request signal is '1', a probe search is running or data transfer is active. When request signal is '0', it denotes the idle state, and an established path will be released. The usage of the ANS signal is listed in the table of Fig. 2.

B. Detailed switch architecture

The internal structure of a switch is shown in Fig. 3. It is divided into two parts: control path and data path. The data path transfers data through the configured data crossbar. The control path is used to set up or tear-down a data path. The control path and data path share the same input and output wires.

There are five allocators, each of which is responsible for the channel allocation of one output direction. The principle of our single cycle maximal strong fairness allocator is similar to a wave-front allocator [13][14]. Detailed discussion of our allocator is beyond the scope of this paper.

Besides, every input or output channel has a controller that controls the value of the backward ANS signal and the forward probe. The FSMs of the input and output controllers are shown in Fig. 4.

C. Properties of our CS NoC

1) Source synchronized data transfer

Similar to Pham's work [19] and as in [28], the control path and data path can work at different clock frequencies but share the same wires without interference. This clock scheme takes advantage of the property of CS NoC that the setup phase never overlaps with the data transfer phase. During the path setup phase, the data path should have no active clock signal, thus it is idle. And the cross bar of the data path can be configured under the control path clock (probe clock). During the data transfer phase, the control path ignores data variations on the shared wire links. It just listens to the request and ANS signals.

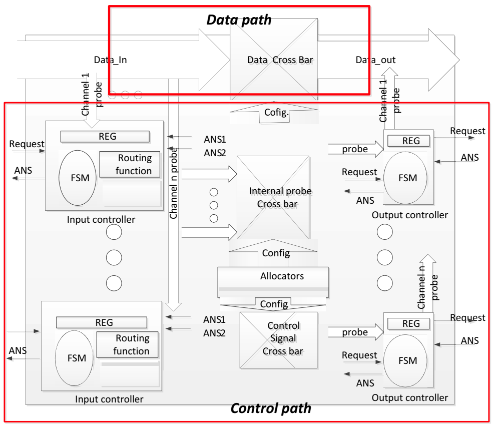
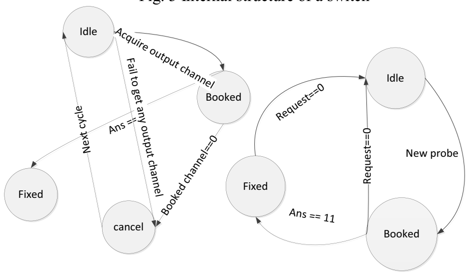


Fig. 3 Internal structure of a switch



Input controller FSM.   b) Output controller FSM

Fig. 4 FSM of input controller and output controller

Therefore, we can utilize either source synchronous data transfer [29][30][19] or clock gating to realize this separation of data and control path clock schemes, so that the data transfer can be benefit from a higher clock frequency. In this paper, we chose the former source synchronous data transfer. The usage of this technique on CS NoC has been justified by Pham et al. [19][20].

We want to emphasize that source synchronized data transfer is a unique property of CS NoC. PS NoC cannot take advantage of such technique because its TDM channel sharing nature.

2) Predictable latency

One of the benefits of the probing set-up approach is predictable latency. In our design, each setup probe takes 2 clock cycles per hop, and the ANS signal takes 1 cycle per hop. So, it takes at most 3*D+6 cycles for a probe to travel from source to destination and back the ANS signal (D is the hop distance between source and destination). 4 cycles is the overhead consumed in the source and destination nodes. Therefore, in an n*n mesh the worst case for a single search takes 3*(2*n-2)+6 cycles, no matter if the result is a success or a failure.

For data transfer, the head flit takes 2 cycles per hop, and the following flits are pipelined with 2cycles per hop.

V.PS NoC USED FOR COMPARISON

The PS NoC for comparison adopts a classical input-buffering virtual channel (VC) wormhole router architecture with the dimension order routing algorithm. The router's architecture is shown in Fig. 5. A router has 3 pipeline stages: (1) VC Allocation and speculative Switch Allocation Switch Traversal (ST) and Next Route Computation (NRC). (3) Link traversal[1].

The PS NoC adopts some advance techniques like speculative allocation and look-ahead route computation [16][31]. Speculative allocation enables head flits to bypass the VC allocation stage in the router pipeline by allowing them to bid for crossbar access at the same time they request an output VC. Look-ahead route computation, or next route computation, computes the route for the next router before a head flit is actually delivered to the next router. These two techniques can reduce the pipeline stages of a PS router.

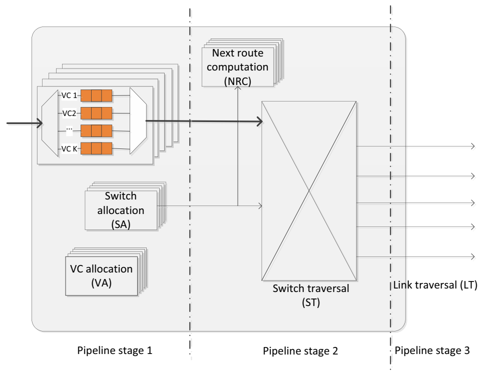


Fig. 5 Architecture of the PS NoC for comparison

[1] The link traversal stage is used to compensate wire delay.

cycle	1	2	3	4	5
head flit	VA SA	NRC ST	LT		
body flit 1		SA	ST	LT	
body flit 2			SA	ST	LT

Fig. 6 Pipeline stages of the PS NoC

As suggested by Fig. 6, head flits and body flits experience 3 pipelines. Both virtual channel allocations and switch allocations are realized by simple separable-input-first round-robin allocator to reduce the critical path latency.

The PS NoC in this paper can have different number of virtual channels and buffer depths.

VI.ANALYSIS ON PS NOC AND CS NOC

A. Critical path latency analysis

Let us consider a CS NoC. For data path, the critical path latency is basically a 4-by-4 cross-bar latency. Using standard library, the critical path of the cross-bar is just a 4-to-1 multiplexer, which consists of 4 level and/or gates latency. Full custom design can further reduce such latency to one gate level. For control path, the critical path consists of a 4-channel allocator latency plus a 4-by-4 crossbar latency.

For PS NoC, the critical path latency comes from VC-allocator. A separable allocator with round-robin fairness consists of two consecutive steps of round-robin arbitration. Even each input direction just has one VC, still the latency of such 4-channel allocator is much larger than the 4-by-4 cross-bar latency inside CS NoC. With the most advanced design [32], the latency of a round-robin allocator scales up with $O(logq)$, where q is the number of virtual channels. Empirically, each input directions needs 4 or more VCs to reduce head-of-line blockings, so the VC allocator in practice is at least a 16-channel allocator.

Comparing CS NoC with PS NoC which has 4 VCs per input, we may deduct that the critical data path latency td of CS NoC should be much smaller than the critical path latency tp of PS NoC. The critical control path latency tc of CS NoC should be close to, or smaller than tp. This is because tc consists of one 4-channel allocator latency plus one 4-by-4 crossbar latency, while tp is basically the latency of a 16-channel allocator, which may double the latency of a 4-channel allocator.

B. Zero load analysis of CS vs. PS

Zero load analysis is useful, since in practice, compared to actual requirements, a NoC is usually over-designed. This means that in a lot of chances, an on-chip network is operating under very low traffic load.

Suppose a packet contains k flits, the clock period is tp for PS NoC, and tc for the probe path of CS NoC and td for data path. The average hops of a packet is D. Suppose further that it takes *hp* cycles-per-hop for the head flits and 1 cycle per hop for the following flits in the PS NoC, and in the CS NoC it takes *hc* for the setup probes (including the travelling back of acknowledgement signal) and *hd* for the data. For simplicity, overheads in sender, receiver and network interface are neglected. Then, at zero load, the approximate average time of packet delivery is, PS NoC and CS NoC, respectively:

$$Tps \ = [D * hp + k] * tp \qquad (1)$$

$$Tcs = D * hc * tc + D * hd * td + (k-1) * td \ (2)$$

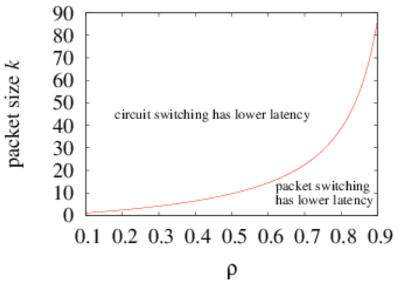


Fig. 7 Packet size k for break even points between PS and CS NoC

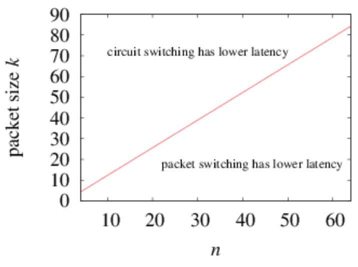


Fig. 8 Mesh size n for break even points between PS and CS NoC

To make a simple comparison, we assume that tp=tc, hd=hc=hp=2 cycles per hop. Then the breakeven point is defined by

$$Tps - Tcs = k * tp - (k-1) * td - \mathrm{D} * 2 * td \ (3)$$

$\rho = td/tp$ expresses how much faster the circuit switching logic can be compared to packet switching. $\rho = 1$ means both run at the same frequency. However, according to our previous analysis, $\rho < 1$. When $\rho < 1$, circuit switching exhibits lower packet delay for packets above a certain size *k*. This value of *k* is plotted in Fig. 7 as a function of ρ and for (D=6).

Does the critical packet size *k* depend on the size of the network? We consider this by using uniform random traffic in an nxn mesh, so that $\mathrm{D} = \frac{2}{3}n$. For $\rho = 0.5$, we vary *n* between 4...64, and find the break even k grows from 5 to 85, as shown in Fig. 8. Thus, with larger networks the packet size has to be larger as well for CS NoC to become faster than PS NoC.

Mathematical analysis of network contention under load is complex and has to be based on many assumptions. Therefore, we study the networks under various loads by simulation where we also have more realistic values for *tc*, *td*, *tp*, *hc*, *hd*, and *hp*.

VII. Experiment Settings

In section 3, we have made our conjecture on the performance curves of CS NoC and PS NoC. Now, we will check whether experiment results are in accordance with our conjecture. All experiments are based on PS NoCs and CS NoCs with 8x8 mesh topology. Uniform random traffic with Poisson arrival time distribution is used in our experiments for evaluation purpose.

A. Simulation method

As in Fig. 9, inside each resource node a request generator generates set-up requests according to a certain probability distribution and pushes them into a queue. An FSM (Finite State Machine) pops a request out of the queue and sends it out when the output channel is available. Then the FSM waits for the ANS signals to decide what to do next.

We have implemented an HDL model for synthesis and for evaluation. Any data point that is shown in the figures comes from a simulation of 250 million cycles, of which the first 250000 cycles are discarded as warm up period.

Since PS NoC and CS NoC are operating at different clock frequencies, performances in the unit of clock cycles are not persuasive. Thus we evaluate the delay in the unit of nano-second, and the throughput and injection rate in MB/s.

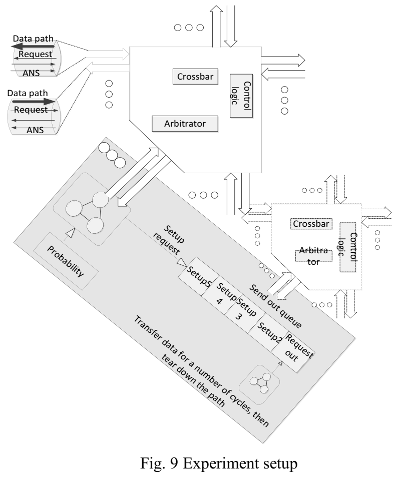


Fig. 9 Experiment setup

VIII. Evaluation and Comparison

A. Evaluations on baseline candidates

The baseline candidates of the PS and CS NoC are listed in Tab. 1, both of which are synthesized by Synopsys

Design Compiler (DC) with SMIC 90 nm library. Both candidates have the same channel width (8 bytes).The power and area per switch reported by DC is calculated at each one's maximum clock frequency.

The synthesize results obey our previous analysis. The data path of CS NoC is more than twice faster than PS NoC. The control path of CS NoC is also about 1.3 times faster than PS NoC.

The header flits in PS NoC take 3 cycles per hop. And for CS NoC, the probes take 2 cycles per hop and 1 cycle for backward ANS signal, under the control of probe clock. The data transfer takes 2 cycles per hop, under the control of the data clock.

Therefore, in our experiments, according to Tab. 1, *tp*=1.2 ns, *tc*=0.9 ns, *td*=0.56 ns, *hp*=3, *hc*=2, *hd*=2, D = $\frac{2}{3}n = 5.3$.

Tab. 1 Parameters and synthesis results of PS NoC and CS NoC (Per switch)

Packet Swiched NoC	Circuit Switched NoC
Virtual channels: 4 (v4)	Channel per direction: 1
Buffer size: 4 (b4)	
Flit width : 8 bytes	Channel width : 8 bytes
Max. Freq. : 833 MHz	Max. Data Freq.: 1.786GHz
	Max. Probe Freq.: 1.11GHz
Area: 144572 um^2	Area: 30874. um^2
Power: 21.7 mW @ 833MHz	Power: 26.8 mW @1.786/1.11GHZ

The channel bandwidth and the clock frequency of PS and CS NoC can be found in Tab. 1. The influence of delivering packets of variable size (from 4 flits to 160 flits) in both PS NoC and CS NoC is evaluated and shown in Fig. 10 and Fig. 11. The trends in these two plots are opposite to each other. PS NoC (Fig. 10) decreases in performance under load as the packet size gets larger, while CS NoC (Fig. 11) improves in performance with larger packets. To better illustrate this phenomenon we take snapshots at injection rates 2000 MB/s and 1142 MB/s, and plot packet size against average packet delay (Fig. 12.).

For a given injection rate CS NoC improves performance as packets get larger, while the performance of PS NoC deteriorates. Consequently, there are cross-points between the circuit-switching and the PS NoC curves in Fig. 12. At injection rate 2000 MB/s, the cross-point is 62 (496 bytes) flits and at injection rate of 1142 MB/s the cross point is 40 flits. In general, the cross point shifts towards larger packets as the injection rate increases.

This observation can be explained by three phenomena:

- For medium packets (20-60 flits in this case) and large packets (above 60 flits in this case) at low load, CS NoC performs better than PS NoC. This is in agreement with our zero load analysis.
- PS NoC handles small (below 20 in uniform traffic case) and medium packets at high load better than CS NoC. A given number of VCs and buffers can handle burstiness and unbalance below a certain level well. If in a PS NoC small packets compete for a resource, the only cost is the waiting time of one packet for a few cycles. If contention occurs in a CS NoC during the setup phase, one probe has to go back, tear down all allocated resources and start over again. Thus, the penalty is more sever because the delay due to contention is higher and many resources are allocated unnecessarily.
- As packet size increases, the contention overhead increases for PS and it is constant (decreases in relative terms) for CS NoC. In PS the delay incurred by contention is proportional to packet size. Thus, the larger the packets the higher the penalty of contention. For CS the cost of contention is relatively independent of packet size and is thus better amortized over large packets at high load.

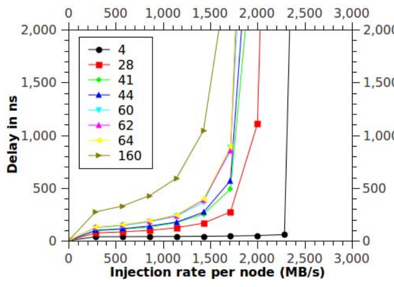


Fig. 10 Delay for PS NoC with different packet size in flits

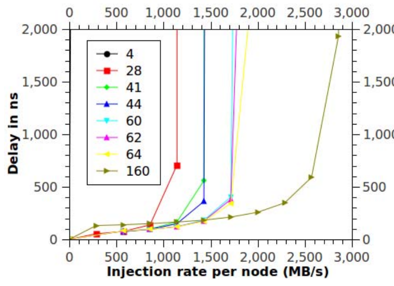


Fig. 11 Delay for CS NoC with different packet size in flits

We also studied the influence of packet size on maximum throughput, as suggested by Fig. 13. As the packet size increases, the maximum throughput decreases in PS NoC, while CS NoC has a contrary trend. For example, for PS NoC with packets of 4 flits (32 bytes), the maximum throughput reaches 2614 MB/s. However, when packet size grows up to 640 flits (5120 bytes), the maximum throughput is 1950 MB/s. For CS NoC, the maximum throughput is 279 MB/s with 4-flit packets. However, as the packet size grows to 640 flits, the throughput goes up to 3479 MB/s. Again, there is a cross over point which lies around 62 flits/packet.

Thus, when packets are large enough, CS NoC is superior to PS NoC in both latency and throughput.

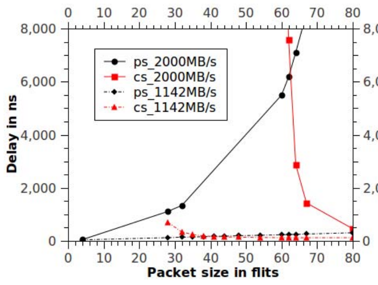


Fig. 12 Latency snapshot of injection rate 2000 MB/s and 1142 MB/s

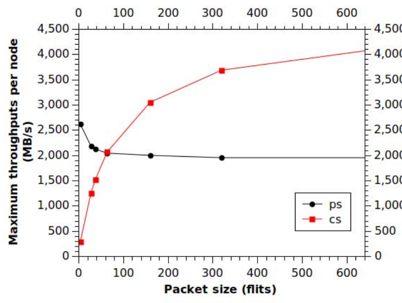


Fig. 13 Maximum throughput comparison.

B. Comparisons between more candidates

In this section, we compared more configurations of PS NoC. Throughput comparison results are shown in Fig. 14. The packet size is in the unit of byte.

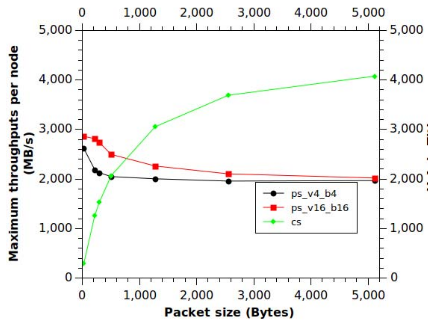


Fig. 14 Maximum throughput comparison between PS NoC and different configurations of CS NoC

For PS NoC, as we expected, we can see that ps_v16_b16 (16 virtual channels and each contains 16 buffers) can enhance the maximum throughput when packets are not very large.

However, for large packet size, eg. each packet contains 640 flits (5120 bytes), 16 VCs with 16 stages of buffers are still not enough. The maximum throughput of ps_v16_b16 at such a packet size is reduced to that of ps_v4_b4. Since large packet weakens the effects of VCs and buffers, we can conjecture that if packets are large enough, the maximum throughput of configurations which contain more VCs and buffers will shrink to that of the baseline candidate ps_v4_b4.

IX. Conclusions

Our general conclusions are shown in Tab. 2, detailed conclusions are listed as followings:

Tab. 2 The favorite working areas of CS and PS NoC

	small packets	medium packets	large packets
Low load	PS is better	CS is better	CS is better
High load	PS is better	PS is better	CS is better

- At very low injection rate, CS NoC is better than PS NoC above a certain packet size (see equation (3)). The critical packet size, above which CS NoC outperforms PS NoC, grows linear with the size of the network.
- For small packet size, PS NoC handles congestion better than CS NoC. Hence, when increasing injection rate, PS NoC shows better performance. This explains why for smaller size of packets (below 62 flits), PS NoC outperforms CS NoC at high injection rate.
- Growing packet size increases the congestion penalty for PS NoC, while it is relatively packet size independent for CS NoC. This means for a given injection rate, CS NoC performs better and better with growing packet size, while PS NoC is getting worse.
- Increasing virtual channels and buffers of PS NoC can enhance throughput for small packets. But this has little influence on very large packets.

References

[1] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, "Concepts and Implementation of Spatial Division Multiplexing for Guaranteed Throughput in Networks-on-Chip," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1182 –1195, Sep. 2008.

[2] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect," in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, 2008, pp. 88–598.

[3] K.-C. Chang, J.-S. Shen, and T.-F. Chen, "Evaluation and design trade-offs between circuit-switched and packet-switched NOCs for application-specific SOCs," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 143 –148.

[4] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit, "An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005, p. 155a.

[5] N. Chin-Ee and N. Soin, "Qualitative and quantitative evaluation of a proposed circuit switched network-on-chip," in *2010 IEEE*

International Conference on Semiconductor Electronics (ICSE), 2010, pp. 108–113.

[6] N. Chin-Ee and N. Soin, “A study on circuit switching merits in the design of network-on-chip,” in *2010 International Conference on Computer and Communication Engineering (ICCCE)*, 2010, pp. 1–5.

[7] J. Lim, E. Hunt Siow, Y. Ha, and P. K. Meher, “Providing both guaranteed and best effort services using Spatial Division Multiplexing NoC with dynamic channel allocation and runtime reconfiguration,” in *Microelectronics, 2008. ICM 2008. International Conference on*, 2008, pp. 329 –332.

[8] M. Modarressi, H. Sarbazi-Azad, and M. Arjomand, “A hybrid packet-circuit switched on-chip network based on SDM,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, 3001 Leuven, Belgium, Belgium, 2009, pp. 566–569.

[9] A. K. Lusala and J.-D. Legat, “Combining sdm-based circuit switching with packet switching in a NoC for real-time applications,” in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, 2011, pp. 2505 –2508.

[10] A. K. Lusala and J.-D. Legat, “Combining SDM-Based Circuit Switching with Packet Switching in a Router for On-Chip Networks,” *International Journal of Reconfigurable Computing*, vol. 2012, pp. 1–16, 2012.

[11] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, “The Nostrum backbone-a communication protocol stack for Networks on Chip,” in *17th International Conference on VLSI Design, 2004. Proceedings*, 2004, pp. 693–696.

[12] S. Park, T. Krishna, C.-H. Chen, B. Daya, A. Chandrakasan, and L. Peh, “Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45nm SOI,” in *2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2012, pp. 398 –405.

[13] P. Guerrier and A. Greiner, “A generic architecture for on-chip packet-switched interconnections,” in *Proceedings of the conference on Design, automation and test in Europe* , 2000, pp. 250–256.

[14] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, “HERMES: an infrastructure for low area overhead packet-switching networks on chip,” *the VLSI Journal*, vol. 38, no. 1, pp. 69–93, Oct. 2004.

[15] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, “GARNET: A detailed on-chip network model inside a full-system simulator,” in *IEEE International Symposium on Performance Analysis of Systems and Software, 2009. ISPASS 2009*, 2009, pp. 33–42.

[16] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[17] M. Winter and G. P. Fettweis, “A Network-on-Chip Channel Allocator for Run-Time Task Scheduling in Multi-Processor System-on-Chips,” in *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, 2008. DSD '08*, 2008, pp. 133 –140.

[18] M. Winter and G. P. Fettweis, “Guaranteed service virtual channel allocation in NoCs for run-time task scheduling,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, 2011, pp. 1 –6.

[19] P.-H. Pham, J. Park, P. Mau, and C. Kim, “Design and Implementation of Backtracking Wave-Pipeline Switch to Support Guaranteed Throughput in Network-on-Chip,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 2, pp. 270 –283, Feb. 2012.

[20] P.-H. Pham, P. Mau, J. Kim, and C. Kim, “An On-Chip Network Fabric Supporting Coarse-Grained Processor Array,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1 –5, 2012.

[21] A. K. Lusala and J.-D. Legat, “Combining SDM-Based Circuit Switching with Packet Switching in a Router for On-Chip Networks,” *International Journal of Reconfigurable Computing*, vol. 2012, pp. 1–16, 2012.

[22] R. Stefan, A. Molnos, and K. Goossens, “dAElite: A TDM NoC Supporting QoS, Multicast, and Fast Connection Set-up,” *IEEE Transactions on Computers*, vol. PP, no. 99, p. 1, 2012.

[23] K. Goossens, J. Dielissen, and A. Radulescu, “AEthereal network on chip: concepts, architectures, and implementations,” *IEEE Design Test of Computers*, vol. 22, no. 5, pp. 414 – 421, Oct. 2005.

[24] N. Ma, Z. Lu, and L. Zheng, “System design of full HD MVC decoding on mesh-based multicore NoCs,” *Microprocess. Microsyst.*, vol. 35, no. 2, pp. 217–229, Mar. 2011.

[25] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest, “Spatial division multiplexing: a novel approach for guaranteed throughput on NoCs,” in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, New York, NY, USA, 2005, pp. 81–86.

[26] D. Wiklund and D. Liu, “SoCBUS: switched network on chip for hard real time embedded systems,” in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003, p. 8–pp.

[27] C. Hilton and B. Nelson, “PNoC: a flexible circuit-switched NoC for FPGA-based systems,” *IEE Proceedings of Computers and Digital Techniques,* vol. 153, no. 3, pp. 181–188, 2006.

[28] S. Liu, A. Jantsch, and Z. Lu, “Parallel probing: Dynamic and constant time setup procedure in circuit switching NoC,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, 2012, pp. 1289 –1294.

[29] D. Walter, S. Hoppner, H. Eisenreich, G. Ellguth, S. Henker, S. Hanzsche, R. Schuffny, M. Winter, and G. Fettweis, “A source-synchronous 90Gb/s capacitively driven serial on-chip link over 6mm in 65nm CMOS,” in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, 2012, pp. 180 –182.

[30] D. Schinkel, E. Mensink, E. Klumperink, E. van Tuijl, and B. Nauta, “Low-Power, High-Speed Transceivers for Network-on-Chip Communication,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 12 –21, Jan. 2009.

[31] D. U. Becker and W. J. Dally, “Allocator implementations for network-on-chip routers,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009, pp. 52:1–52:12.

[32] H. J. Chao, C. H. Lam, and X. Guo, “Fast ping-pong arbitration for input–output queued packet switches,” *International Journal of Communication Systems*, vol. 14, no. 7, pp. 663–678, 2001.