ON-CHIP MULTIPROCESSOR COMMUNICATION NETWORK DESIGN AND ANALYSIS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Terry Tao Ye
December 2003

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Giovanni De Micheli
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Teresa Meng

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Martin Morf

Approved for the University Committee on Graduate Studies:

Preface

To My Wife Joanne Wang and My Parents Lu Ye and YongShang Gao

Acknowledgments

I would like to first and foremost thank my advisor Professor Giovanni De Micheli for his advice and guidance in my PhD research. Professor De Micheli is one of the most influential people in my career path. He helped me to make the right decision when I began to pursue my PhD at Stanford, and had been an inspiring mentor throughout the years.

I would like to thank my wonderful wife Joanne Wang. Without her love, encouragement and patience over the years, I would not be able to finish the PhD research. She is always the first one I would go to whenever I need support, and the first one to share my happiness on my success.

I would also like to thank my parents Lu Ye and YongShang Gao, for their love and support. They always believed in me on whatever decision I made, and are proud of me on whatever achievement I may have.

My PhD research would not be possible to proceed without the direct and indirect help from many other people. I would like to thank my PhD Committee member Professor Teresa Meng and Professor Martin Morf for their suggestions and reviews on my oral defense presentation and thesis writing. I would also like to acknowledge the support from the members of Professor De Micheli's research group - the CAD group at Stanford, and the Stanford Computer Systems Lab.

This research is supported by the *MARCO GSRC* research center.

Contents

List of Tables

List of Figures

Abstract

Future multiprocessor systems-on-chip (SoC) designs will need novel on-chip communication architectures that can provide scalable and reliable data transport – On-chip network architectures are believed to be the ideal solution to many of today's SoC interconnection problems. On-chip network architectures may adopt design concepts and methodologies from computer networks, namely from system-area-networks and parallel computer clusters. Nevertheless, silicon implementation of networks requires a different perspective, because network architectures and protocols have to deal with the advantages and limitations of the silicon fabric. These characteristics will require new methodologies for both on-chip switch designs as well as routing algorithm designs. We envision that future on-chip systems will be communication-centric, in particular, energy and performance issues in designing the communication infrastructure will become challenging.

In this thesis, we explore several critical aspects in the physical and network layers of the on-chip communication stack that include: 1) On-chip interconnect power consumption modeling and simulation. 2) On-chip routing schemes and switch architecture. 3) Packetization and its impact on system performance and power consumption. 4) Physical planning for on-chip networks. Many of the issues are new and unique for on-chip networks, thus novel techniques and design concepts have to be explored.

Using multi-processor systems-on-chip (MPSoC) networks as the experimental platform, this thesis presents both quantitative and qualitative analysis for on-chip networks. New methodologies and solutions are also proposed to achieve better performance and power balance for MPSoCs.

Chapter 1

Introduction

1.1 From Systems-on-Chip to Networks-on-Chip

Multi-processor systems-on-chips (MPSoCs) have been widely used in today's high performance embedded systems, such as *network processors* (NP) and *parallel media processors* (PMP). They combine the advantages of data processing parallelism of multi-processors and the high level integration of *systems-on-chip* (SoCs).

Driven by the advances of semiconductor technology, future SoCs will continue to accelerate in system's complexity and capacity. SoCs in the next decade are expected to integrate hundreds, or even more of, *processing elements* (PEs) and/or *storage elements* (SEs) on a single chip. SoC designs at this scale cannot be started from scratch, instead, it is common believe that SoCs will be designed using pre-existing components, such as processors, controllers and memory arrays. Future SoC design methodologies have to support component re-use in a plug-and-play fashion in order to meet time-to-market requirement.

In such a plug-and-play system integration approach, the most critical factor will be related to the communication scheme among components. The design of reliable, low-energy and high-performance on-chip communication architectures for future SoCs will pose unprecedented challenges [5][15][24]. Interconnect technology will become the limiting factor for achieving the operational goals. Therefore, we envision a communication-centric view of design methodology in the years to come [44], as

50-100nm technologies will prevail in the second part of this decade.

Traditional on-chip communication structures have already encountered many limitations in today's VLSI designs. Many of these limitations will become even more problematic as the semiconductor technology advances into newer generations. These limitations are either associated with the scaling-down of the device feature size, or they are inevitable with the scaling-up of design complexity [25]. Particularly, the following issues will become the bottleneck in the future communication-centric SoC design scheme:

- **Throughput Limitation**– Traditional on-chip communication structures (i.e., the buses) cannot scale up as the number of components increases. When multiple dataflows are transmitted concurrently, they will compete for the same communication resources.

- **Energy Consumption**– As the VLSI device features are continuously shrinking down, interconnect wires have been one of the major contributors of the system energy consumption. The buses used in many of today's SoC designs are notoriously not energy-efficient, because every bit transmitted is propagated throughout the bus to every terminal.

- **Signal Integrity** – Energy considerations will impose small logic swings and power supplies, most likely below 1 Volt. Smaller device feature sizes will also produce denser wires (i.e., 7 layers or more of routing wires) connecting highly compacted transistors. Therefore, future VLSI systems will become more vulnerable to various forms of electrical noise, such as *cross-talk*, *electro-magnetic interference* (EMI) and radiation-induced charge injection (*soft errors*). An additional source of errors is contention in shared-medium networks. Contention resolution is fundamentally a non-deterministic process, because it requires synchronization of a distributed system, and for this reason it can be seen as an additional noise source. Because of these effects, the mere transmission of digital values on wires will be *inherently unreliable.*

- **Signal Latency**– The propagation delay on wires will gradually dominate the

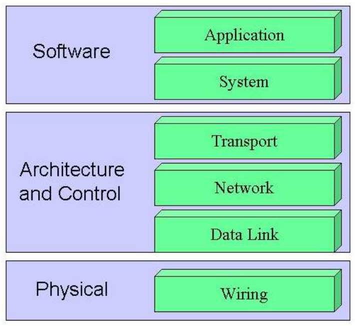

Figure 1.1: Micro-Network Stack

signal latency as the wire feature size shrinks. In fact, wire delay has already become a big challenge in today's VLSI systems, because the delay is determined by the physical distribution of the components, which is hard to predict in the early stages of the design flow. A more predictable communication scheme is of great importance in the future SoC designs.

- **Global Synchronization**– Propagation delay on global wires - spanning a significant fraction of the chip size - will pose another challenge on future SoCs. As the wire size continues to shrink, the signal propagation delay will eventually exceed the clock period. Thus signals on global wires will be pipelined. Hence the need for latency insensitive design is critical. The most likely synchronization paradigm for future chips is *globally-asynchronous locally-synchronous* (GALS), with many different clocks.

We propose to use network design technology to analyze and design future SoCs. In other words, we view a SoC as a *micro-network* of components, where the PEs and SEs are interconnected as *node components*, or simply referred to as *nodes*. We postulate that SoC interconnect design analysis and synthesis can be done by using the *micro-network stack* paradigm, which is an adaptation of the protocol stack [48] (Figure 1.1). Thus the electrical, logic, and functional properties of the interconnection scheme can be abstracted.

1.2 Micro-Networks: Architectures and Protocols

In the proposed on-chip network architecture, or *networks-on-chip* (NoC), each PE or SE is abstracted as a node, and the nodes are interconnected by the micro-network that can provide scalable and concurrent *point-to-point* (P2P) or *point-to-many* (P2M) connection. As a new SoC design paradigm, NoCs will support novel solutions to many of above mentioned SoC interconnect problems. For example, multiple dataflows can be supported concurrently by the same communication resources, data integrity can be enhanced by error correction and data restoration, and components are more modularized for IP reuse.

On-chip network architectures may adopt design concepts and methodologies from computer networks, namely from *system-area-networks* (SAN) and *parallel computer clusters* (PCC). Communication in on-chip network architecture is also regulated by protocols, which are designed in layers. The layer stack in NoCs may differ from traditional networks because of local proximity and because they exhibit much less non-determinism. Although we may borrow some concepts and approaches from computer network architectures, we do not need to follow the OSI seven-layer scheme to setup a communication transaction. Instead, on-chip networks may exploit tailor-made protocols to satisfy application specific requirements.

We analyze next specific issues related to the different layers of abstraction outlined in the micro-network stack in a bottom-up way.

1.2.1 Physical Layer

Global wires are the physical implementation of the communication channels. While the VLSI technology trends lead us to use smaller voltage swings and capacitances, the signal integrity problem will get worse. Thus the trend toward faster and lower-power communication may decrease reliability as an unfortunate side effect.

Current design styles consider wiring-related effects as undesirable parasitics, and try to reduce or cancel them by specific and detailed physical design techniques. It is important to realize that a well-balanced design should not over-design wires so that their behavior approaches an ideal one, because the corresponding cost in performance, energy-efficiency and modularity may be too high. Physical layer design should find a compromise between competing quality metrics and provide a clean and complete abstraction of channel characteristics to micro-network layers above.

1.2.2 Datalink, Network and Transport Layers

The *data-link layer* abstracts the physical layer as an unreliable digital link, where the probability of bit upsets is not negligible (and increasing as technology scales down). An effective way to deal with errors in communication is to *packetize* data. If data is sent on an unreliable channel in packets, error containment and recovery is easier, because the effect of errors is contained by packet boundaries, and error recovery can be carried out on a packet-by-packet basis. At the data link layer, error correction can be achieved by using standard *error correcting codes* (ECC) that add redundancy to the transferred information.

At the *network layer*, packetized data transmission can be customized by the choice of switching and routing algorithms. The former establishes the type of connection while the latter determines the path followed by a message through the network to its final destination.

At the *transport layer*, algorithms deal with the decomposition of messages into packets at the source and their assembly at destination. Packetization granularity is a critical design decision, because the behavior of most network control algorithms is very sensitive to packet size. Packet size can be application-specific in SoCs, as opposed to general networks.

1.2.3 Software and Application Layer

Software layers comprise system and application software. The system software provides us with an abstraction of the underlying hardware platform, which can be leveraged by the application developer to safely and effectively exploit the hardware's capabilities.

From a high level application viewpoint, multiprocessor SoC platforms can be viewed as networks of computing nodes equipped with local storage. Software layers are critical for the NoC paradigm shift, especially when energy efficiency is a requirement. Software programming abstractions, development tools and system software need to help programmers understanding communication-related costs and coping with them

1.3 Research Challenges in Networks-on-Chip

The above analysis shows that on-chip networks differ from traditional computer networks, many assumptions and solutions have to be adapted to the on-chip implementation. NoC architectures and protocols have to deal with the advantages and limitations of the silicon fabric. In particular, chip-level communication is localized between nodes (PEs and SEs). On-chip networks do not need to follow the standard schemes for communication since they can use lighter and faster protocol layers. NoCs will require novel methodologies for both on-chip switch designs as well as routing algorithm designs.

1.3.1 Current Research in NoCs

NoC research has been an active topic in recent years. Many researchers around the world are contributing to this field from different aspects. The NoC related research has been primarily carried out in the following three areas: 1) NoC architectures, 2) NoC protocols and 3) NoC design automation. In reality, these three areas are closely interacted to each other, any design exploration on NoC has to consider all issues as well. Next, we will summarize the research activities in different areas briefly.

NoC Architectures

A torus network architecture was proposed by [15] for the feasibility of on-chip communication. In the architecture proposed, each PE is placed as a tile and connected by the torus network (Fig. 1.2a). The tiles represent the processing elements or storage elements. Although the tiles may have different functionalities, they have homogeneous physical dimension and are regularly floorplanned and placed. Packet switching technique is applied for inter-tile communication. The dataflows are segmented into packets, and the packets are routed from sources to destinations. Each tile can perform packet routing and arbitration independently. The network interfaces are located on the peripherals of each tile. Since in a torus network, each tile is abutted to four neighboring tiles, the packets are exchanged between adjacent neighbors.

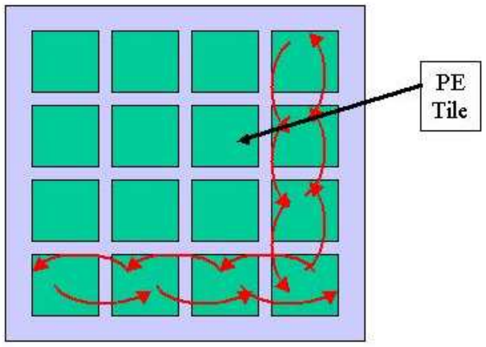

Figure 1.2: The Two-Dimensional Torus Networks Proposed by Dally, et. al

The *Nostrum* network was proposed by [31]. It also adopts the tile-based floorplan structure and utilizes a two-dimensional mesh topology (Fig. 1.3). The dataflows are segmented into packets and packets are routed independently. Different from the architecture proposed in [15], a dedicated switch network is used to perform the routing function and acts as a network interface for each node. The dedicated network consists of switches and interconnect wires. The switches are located at the intersection of the mesh grids. Each switch has its own buffers and arbitration circuits to deliver the packets.

The SPIN network [20] was proposed as a communication architecture for on-chip multiprocessors. The network utilizes a fat-tree topology where each processor is located at the leaf node of the fat-tree, as shown in Fig. 1.4. Messages (segmented into packets) can be exchanged between processing elements by traveling up and down the fat-tree network. The packets are defined as sequences of 36-bit words. The packet header fits in the first word, where a byte in the header identifies the destination address, and other bits are used for packet tagging and routing information. The

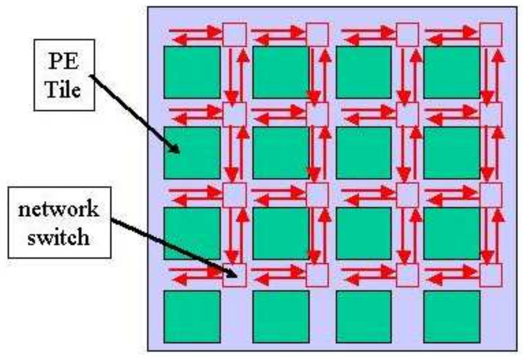

Figure 1.3: The Two-Dimensional Indirect Mesh Proposed by Kumar, et. al.

packet payload can be of variable sizes. Each packet is terminated by a trailer, which contains no data, but a checksum for error detection.

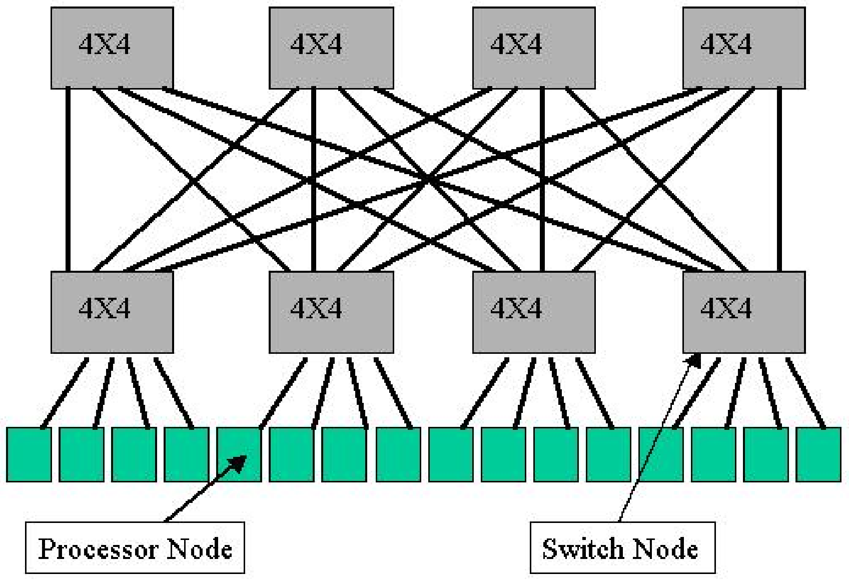

Figure 1.4: The SPIN Networks

The Octagon network (Fig. 1.5) was proposed by [29] as an on-chip communication architecture for network processors. In this architecture, eight processors are connected by an octagonal ring and three diameters. The delays between any two node processors are no more than two stages (through one intermediate node) within the local ring. The Octagon network is scalable. If one node processor is used as the

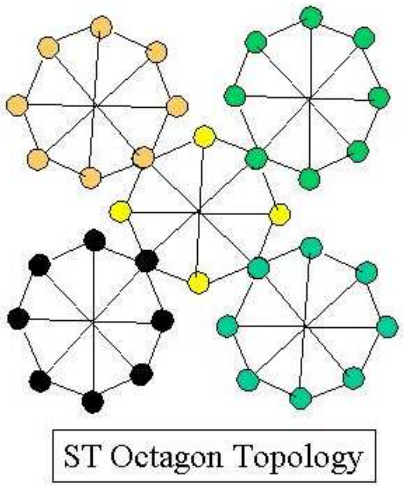

Figure 1.5: The Octagon Networks

bridge node, more Octagons can be cascaded together, as shown in Fig. 1.5.

NoC Protocols

Communication in a NoC architecture is regulated by protocols. A protocol not only determines how dataflows are distributed among networks nodes (known as routing protocol), but also defines how senders and receivers execute the communication transactions (known as transmission protocols). For NoC applications, the reliability and power consumption issues are greatly affected by different protocol options. Next, we will describe some recent works related to the NoC transmission protocol and routing protocol implementation.

A low-swing signaling, error detection coding and a retransmission scheme is proposed by [50]. It minimizes the interconnect voltage swing and frequency subject to workload requirement and S/N conditions. Simulation results show that tangible saving in energy can be attained while achieving at the same time more robustness to large variations in actual workload, noise and technology quality. It shows that traditional worse-case correct-be design paradigm will be less and less applicable in future NoC designs.

Two alternative reliability-enhancement communication protocols: *error-correcting* *codes* and *error-detecting codes with retransmission*, are analyzed and compared by [6]. A set of experiments that apply error correcting and detecting codes to an AMBA bus are compared in terms of the energy consumption. The results show that retransmission strategies are more effective than correction ones from an energy viewpoint. Because larger detection capability allows to use smaller voltage swing on the interconnect, thus helps to reduce the energy consumption. Moreover, error-detection requires minor decoding complexity than the error-correction approach.

Guaranteed service protocols are essential to provide predictable interconnects performance. However, guaranteed communication protocol implementation typically utilize resources inefficiently. In comparison, *best-effort service* protocols overcome this problem but provide no guarantees. In [41], an integration of guaranteed and best-effort services is proposed, which provides efficient resource utilization, and still provides guarantees for critical traffic.

Another NoC traffic mapping and routing algorithm is proposed by [27]. The proposed scheme can automatically map the cores onto a mesh-based NoC architecture and constructs a deadlock-free deterministic routing function such that the total communication energy is minimized. The performance of the resulting communication system is guaranteed to satisfy the specified constraints through bandwidth reservation. An efficient branch-and-bound algorithm is used to solve this mapping and allocation problem. The proposed algorithm is fast and achieves significant energy savings compared to an ad-hoc implementation.

Design Automation for NoC

Xpipes [12] was designed in University of Bologna. It is a scalable and high-performance NoC architecture for multi-processor SoCs. It consists of soft macros that can be turned into instance-specific network components at instantiation time. The flexibility of the Xpipes components allows the NoC to support both homogeneous and heterogeneous architectures. The interface with IP cores at the periphery of the network is standardized. Links can be pipelined with a flexible number of stages to decouple data introduction speed from worst-case link delay. Switches are lightweight and support reliable communication for arbitrary link pipeline depths (latency insensitive operation). Xpipes components have been described in synthesizable SystemC, at the cycle-accurate and signal-accurate level. Xpipes architecture is highly configurable to different network topologies and technology parameters.

A framework for early exploration of the on-chip communication architecture was proposed by [30] at Aachen University of Technology. The framework is also written in SystemC. It is able to capture the performance and cost requirements for different on-chip networks, such as dedicated point-to-point, shared bus, and crossbar topologies. The mapping of the inter-module traffic to an efficient communication architecture is driven by monitoring the performance parameters, i.e., utilization, latency and throughput, etc. The effectiveness of this approach is demonstrated by the exemplary design of a high performance *Network Processing Unit* (NPU), which is compared against a commercial NPU device.

1.3.2 My Contribution in NOC Research

The idea of on-chip network architecture will create many new research opportunities within the *Electronic Design Automation* (EDA) field. In particular, this dissertation has explored the on-chip network and communication design spaces in the following directions:

1. **On-chip interconnect power consumption modeling and simulation**

   Future systems-on-chip have very stringent limitation on power dissipation. Therefore, system performance needs to balance with the power consumption in on-chip network designs. I have proposed a bit-level power model that can be used for on-chip network power consumption analysis. As a case study, I applied this model to analyze the power consumption of switch fabrics used in many different on-chip network topologies.

2. **Novel on-chip communication routing protocol**

   The characteristics of silicon fabrics require new methodologies for on-chip network packet routing. I have proposed a new routing scheme that achieves significant performance improvement and power saving compared with conventional methods. This routing scheme utilizes the abundant wiring resources available on silicon and propagates the network contention information on dedicated control wires. This contention-look-ahead scheme improves the routing performance. Because the contention occurrence between packets is reduced, the on-chip buffer usage can be dramatically reduced. Consequently, the network power consumption is reduced as well.

3. **Packetization and its impact on system performance and power consumption.**

   On-chip network performance and power consumption are greatly affected by the packet dataflows that are transported on the network. In this research, I analyzed the packet size impact on system performance as well as power consumption. Particularly, I have proposed a quantitative method of analysis to evaluate the trade-off relationship between different design options (cache, memory, packetization scheme, etc.) at the architectural level.

4. **Physical planning of on-chip networks and switch fabrics**

   Silicon implementations of on-chip networks need to planarize the interconnect networks and switch fabrics onto the two-dimensional floorplan. On-chip network physical planning is particularly critical for multiprocessor systems-on-chip architectures that utilize regular network topologies. The floorplan requires to preserve the network regularity while minimizing the total interconnect wire length to save power and reduce delay.

   I have proposed an automated MPSoC physical planning methodology; a tool-implementation of this methodology, called *REGULAY*, has also been developed. REGULAY can generate an optimal floorplan for different topologies under different design constraints. Compared with traditional floorplanning approaches, REGULAY shows significant advantages in reducing the total interconnect wirelength while preserving the regularity and hierarchy of the network topology.

1.4 Assumptions and Limitation

The analysis and routing schemes proposed in this dissertation are based on the shared-memory multiprocessor SoCs. Each node contains a microprocessor and a local memory hierarchy that includes one or two levels of cache and a local memory. The memories on different nodes are globally addressed and accessible from remote nodes.

The experiments described in this dissertation are ported from parallel computer benchmarks, many of them are from the Stanford SPLASH project. Although the benchmarks were originally designed for parallel computer clusters, we found them also applicable to many on-chip implementations. Details of those benchmarks are described in the corresponding chapters.

The software and application layer is a very critical aspect on the NoC communication stack. However, in this dissertation, we focus mostly on the physical and network layer issues. The benchmark experiments performed in this research are loaded manually to each node processor, because there is no NoC-based operating system that can support executable programs.

1.5 Thesis Organization

This thesis is organized as follows. Chapter 2 first introduces the basis of on-chip network architectures used in multiprocessor systems-on-chip designs. In particular, we will focus on the shared-memory MPSoCs, although much of the analysis can also be used for other MPSoC architectures. Chapter 3 presents a bit-level power model that can be applied to the analysis of on-chip communication power consumption. Chapter 4 introduces a contention-look-ahead on-chip routing scheme that can reduce the contention occurrence with much smaller buffer space requirement. This routing scheme is particularly useful for on-chip communication design because it minimizes both the power consumption and packet latency. Chapter 5 analyzes different packetization schemes and the corresponding impact on MPSoC performance as well as power dissipation. Chapter 6 shifts the focus to the on-chip network implementation issues. In particular, it is shown that different network topologies can be planarized on a silicon floorplan under varies constraints. A tool-implementation of this floorplanning methodology, called *REGULAY* is also introduced. Chapter 7 will propose future works on networks-on-chip research.

Chapter 2

Networks-on-Chip

In this chapter, we will describe the basis of MPSoC architectures that includes the processor-memory hierarchy as well as inter-processor network topology. MPSoC networks have many special characteristics that are different from traditional computer networks, we will exploit these characteristics in the MPSoC silicon implementation.

2.1 MPSoC Architecture and Networks

The inter-node communication between multiprocessors can be implemented by either *message passing* or *memory sharing*. In the message passing MPSoCs, data transactions between nodes are performed explicitly by the communication APIs, such as *send()* or *receive()*. These API commands require special protocols to handle the transaction, and thus create extra communication overhead. In comparison, in shared memory (in some case, shared level-2 cache) MPSoCs, data transactions can be performed implicitly through memory access operation [11]. Therefore, shared-memory MPSoC architectures have been widely used in many of today's high performance multiprocessor systems. In our research, we will use the shared-memory MPSoC architectures to analysis the on-chip network issues, although many of the solutions can be applied to general MPSoC architectures as well.

We will first introduce some examples of shared-memory MPSoC architectures, then proceed to more detailed analysis of different aspects of on-chip network architectures.

2.1.1 Shared Memory MPSoC Examples

Daytona [1] is a single chip multiprocessor developed by Lucent. It consists of four 64-bit processing elements that generate transactions of different sizes. Daytona targets on the high performance DSP applications with scalable implementation choices. The inter-node communication is performed by the on-chip bus with split transactions. Piranha [4] project is developed by DEC/Compaq, it integrates eight alpha processors on a single chip and uses packet routing for the on-chip communication. The Stanford Hydra [23] chip contains four MIPS-based processors and uses shared level-2 cache for inter-node communication.

All these architectures utilize shared memory (or cache) approach to perform data transactions between processors, and thus achieve high performance with parallel data processing ability. In this chapter, we will use the shared memory MPSoC platform to analyze different aspects of on-chip network architectures.

2.1.2 MPSoC Architecture

A typical shared-memory on-chip multiprocessor system is shown in Figure 2.1. It consists of several node processors or processing elements connected by an on-chip interconnect network. Each node processor has its own CPU/FPU and cache hierarchy (one or two levels of cache). A read miss in L1 cache will create an L2 cache access, and a miss in L2 cache will then need a memory access. Both L1 and L2 cache may use write-through or write-back for cache updates.

The shared memories are associated with each node, but they can be physically placed into one big memory block. The memories are globally addressed and accessible by the memory directory. When there is a miss in L2 cache, the L2 cache will send a request packet across the network asking for memory access. The memory with the requested address will return a reply packet containing the data to the requesting node. When there is a cache write-through or write-back operation, the cache block

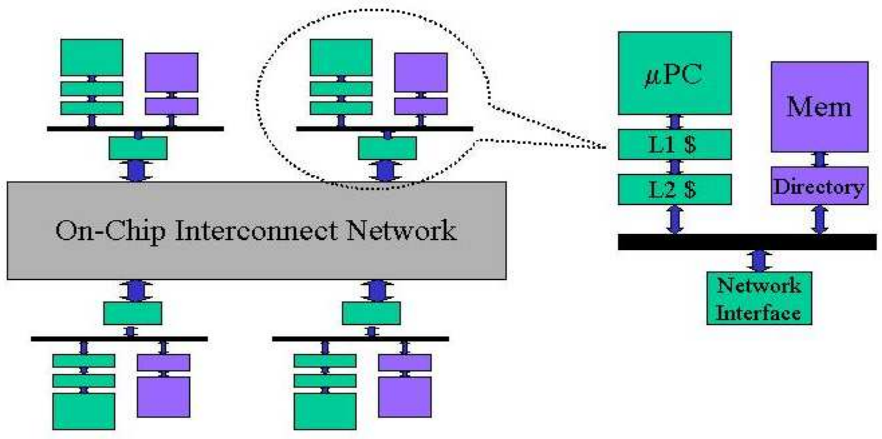

Figure 2.1: MPSoC Architecture

that needs to be updated is encapsulated in a packet and sent to the destination node where the corresponding memory resides.

Cache coherence is a very critical issue in shared-memory MPSoC. Because one data may have several copies in different node caches, when the data in memory is updated, the stale data stored in each cache needs to be updated. There are two methods of solving the cache coherence problem: 1) *cache update* updates all copies in cache when the data in memory is updated; 2) *cache invalidate* invalidates all copies in cache. When the data is read the next time, the read will become a miss and consequently need to fetch the updated data from the corresponding memory.

2.2 MPSoC Network Topologies

Because of different performance requirements and cost metrics, many different multi-processor network topologies are designed for specific applications. MPSoC networks can be categorized as *direct networks* and *indirect networks* [17]. In direct network MPSoCs, node processors are connected directly with each other by the network. Each node performs dataflow routing as well as arbitration. In indirect network MP-SoCs, node processors are connected by one (or more) intermediate node switches. The switching nodes perform the routing and arbitration functions. Therefore, indirect networks are also often referred to as *multistage interconnect networks* (MIN).

Although some direct networks and indirect networks may be equivalent in functionality, e.g., if each node processor has one dedicated node switch, this node switch can either be embedded inside the node processor, or be constructed outside. Nevertheless, direct and indirect topologies have different impact on network physical implementation. In this thesis, to avoid confusion, we call the intermediate switching nodes in indirect networks *switch fabrics*, and simply refer to both node processors and node switches as "nodes".

Direct networks and indirect networks can have different topologies [17]. It is not the objective of this chapter to discuss the functionalities and performance metrics of these different networks. Rather, we are going to give only a brief description of some of the popular network topologies. We will use these topologies as examples to formulate the MPSoC on-chip network problems in later chapters.

2.2.1 Direct Network Topologies

Orthogonal Topology

Nodes in orthogonal networks are connected in k-ary n-dimensional mesh (k-ary n-mesh) or k-ary n-dimensional torus (k-ary n-cube) formations, as shown in Fig. 2.2. Because of the simple connection and easy routing provided by adjacency, mesh and torus networks are widely used in parallel computing platforms [13]. Orthogonal networks are highly regular. Therefore, the interconnect length between nodes is expected to be uniform to ensure the performance uniformity of the node processors.

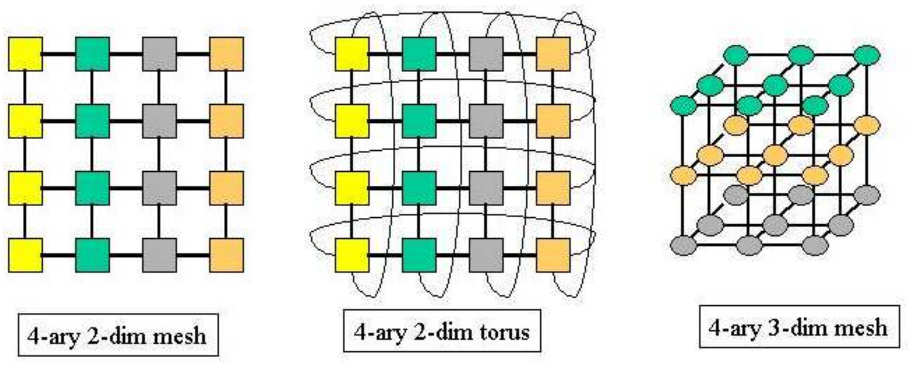

Figure 2.2: Mesh and Torus Networks

Cube-Connected-Cycles Topology

The *cube-connected-cycles* (CCC) topology is proposed as an alternative to orthogonal topologies to reduce the degree of each node [40], as shown in Fig. 2.3a. Each node has 3 degrees of connectivity as compared to 2^n degrees in mesh and torus networks. CCC networks have a hierarchical structure: the three nodes at each corner of the cube form a local ring.

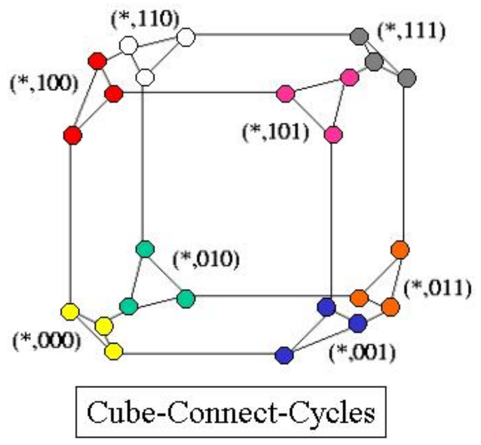

Figure 2.3: Cube-connected-cycles Networks

Octagon Topology

The Octagon network [29] introduced in Chapter 1 is another example of direct network topologies.

2.2.2 Indirect Network Topologies

Crossbar Switch Fabrics

An $N \times N$ crossbar network connects N input ports with N output ports. Any of the N input ports can be connected to any of the N output ports by a node switch on the corresponding crosspoint (Fig. 2.4).

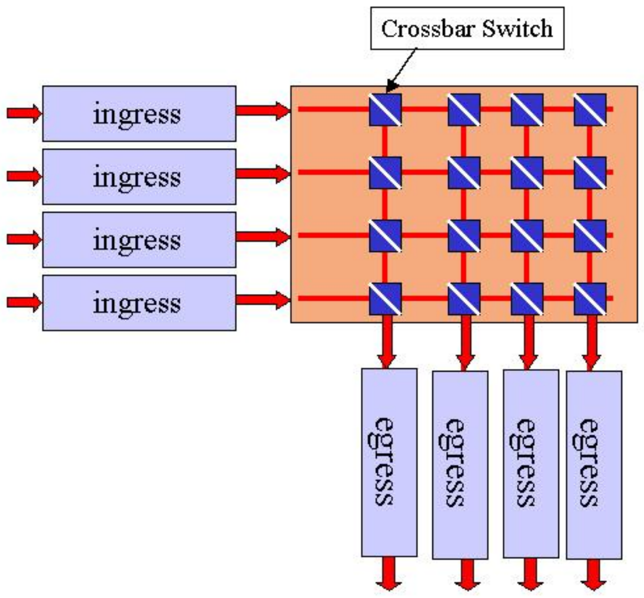

Figure 2.4: Crossbar Switch Fabrics

Fully-Connected Network

An $N \times N$ fully-connected network uses MUXes to aggregate every input to the output (Fig. 2.5). Each MUX is controlled by the arbiter that determines which input should be directed to the output.

Similar to the crossbar network (fully connected switch network is also often referred as crossbar), in fully connected switch network, each source-destination connection has its dedicated data path.

Butterfly Topology

The Butterfly network (Fig. 2.6) is an indirect network architecture. Inside the butterfly fabrics, each source-destination route uses a dedicated datapath. The delays between any two node processors are the same, and the delay is determined by the number of intermediate stages on the switch fabrics.

Butterfly topology has many different isomorphic variations, such as *Banyan Network*, *Batcher-Banyan Networks*, etc., as described as follows.

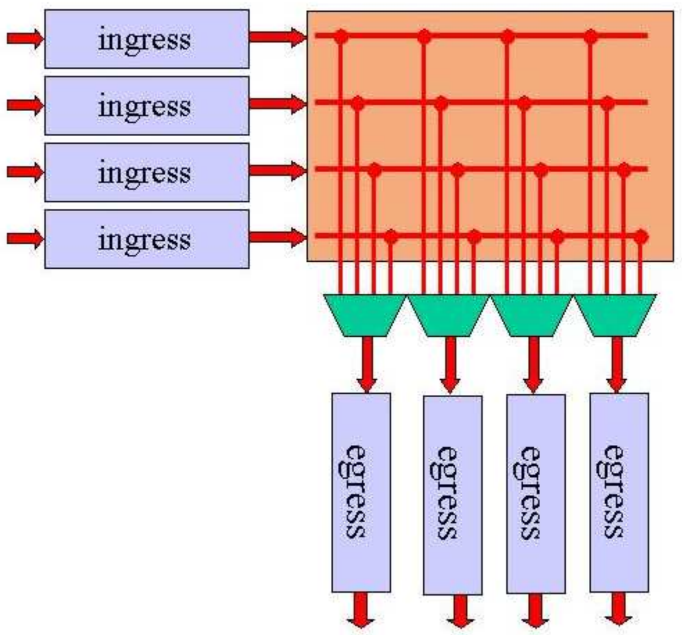

Figure 2.5: Fully-Connected Switch Fabrics

Banyan Network

Banyan network (Fig. 2.7) is an isomorphic variation of Butterfly topology. It has $N = 2^n$ inputs and $N = 2^n$ outputs, where n is called the dimension of Banyan. It has total of $\frac{1}{2}N\log_2 N$ switches in n stages, each stage is referred as stage i where $0 \leq i < n$ [9].

Batcher-Banyan Network

The Batcher-Banyan consists of a Batcher sorting network, followed by the Banyan network, as shown in Fig. 2.8. After sorting network, each input-output connection will have its own dedicated path. Batcher-Banyan network addresses the *interconnect contention* issues [9], which will be analyzed in details in the following chapters.

Fat-tree Topology

Unlike the Butterfly network, a *fat-tree* network provides multiple datapaths from source node to destination node. As shown in Fig. 2.9, the fat-tree network can be

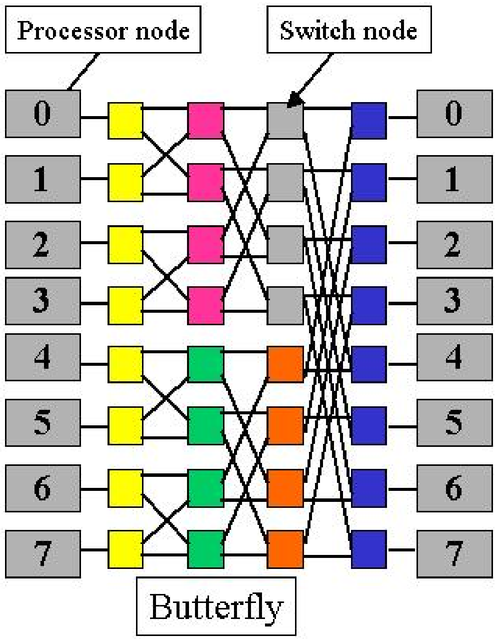

Figure 2.6: Butterfly and Fat-tree Network Switch Fabrics

regarded as an expanded n-ary tree network with multiple root nodes. The network delays are dependent on the depth of the tree. SPIN network [20] is one design example that uses 4-ary fat-tree topology for the MPSoC on-chip communication.

2.3 On-chip Network Characteristics

On-chip networks are fabricated on a single chip and benefit from data locality. In comparison, networked computers are physically distributed at different locations. Although many of the above on-chip network architectures adopt the topology from computer networks, e.g., system area networks and parallel computer clusters. Many assumptions in computer networks may no longer hold for on-chip networks.

2.3.1 Wiring Resources

In computer networks, computers are connected by cables. The number of wires encapsulated in a cable is limited (e.g., CAT-5 Ethernet cable has 8 wires, parallel

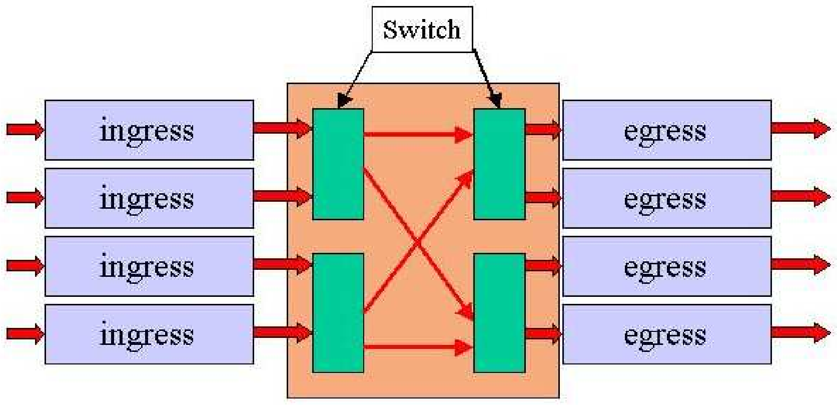

Figure 2.7: Banyan Switch Fabric Network

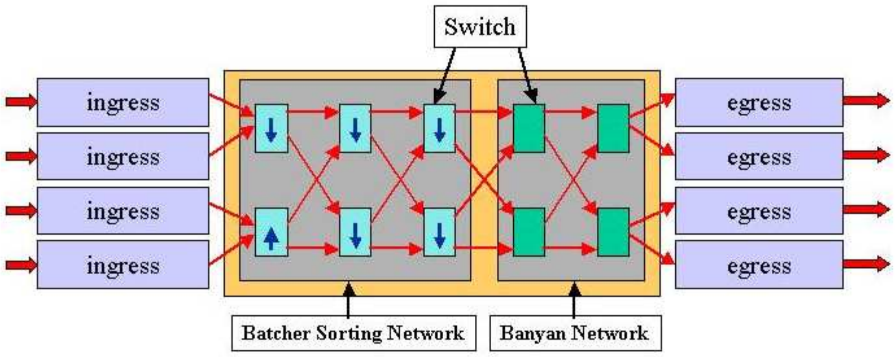

Figure 2.8: Batcher-Banyan Switch Fabric Network

cable in PC peripheral devices has 25 wires, etc.). Binding more wires in a cable is not physically and practically viable. Because of the wiring limitation, in many of today's computer networks, data are serialized in fixed quanta before transmission.

In comparison, the wire connection between components in SoC is only limited by the switching and routing resources. In today's $0.13\mu m$ semiconductor process, the metal wire pitch varies from $0.30\mu m$ to $0.50\mu m$, while 8 metal layers are available. Thus a $100\mu m \times 100\mu m$ switch-box can accommodate hundreds of wires in any direction (i.e., layers). The cost of adding more routing layers continues to decrease as the VLSI process technology advances. Therefore, physical wire density is not the limiting factor for future SoC designs.

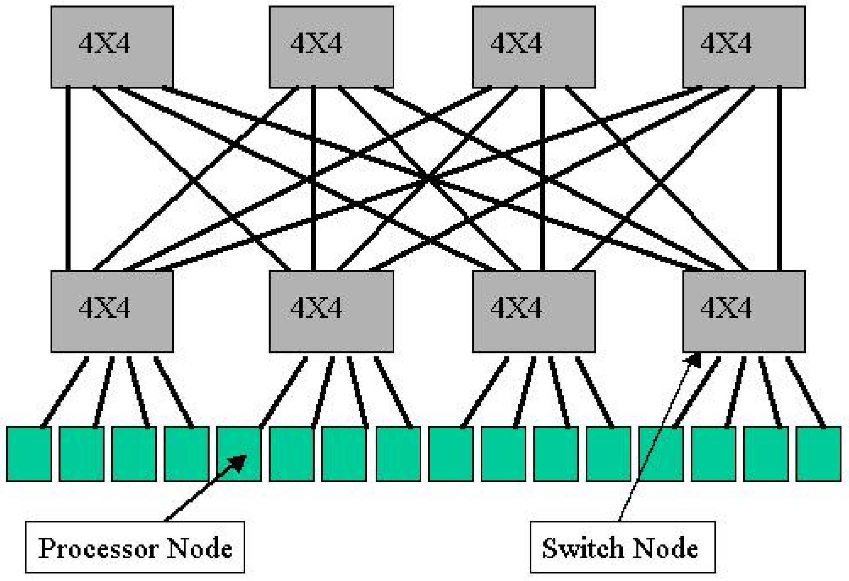

Figure 2.9: The Fat-Tree Networks

2.3.2 Buffers on Networks

Limited wiring resource tends to create contention and limit throughput. Computer networks use heavily buffers to compensate for wire limitation. Buffers provide temporary storage when contention occurs, or when the dataflow exceeds the network throughput. Network switches and routers use a fairly large amount of buffer spaces. These buffers are implemented with SRAMs and DRAMs. The buffer size can be as big as several hundred megabytes (e.g., in the case of network routers).

In comparison, on-chip networks should always balance the buffer usage with other architectural options, because on-chip buffers are implemented by SRAMs or DRAMs, and both memories consume significant power during operation. Besides, on-chip SRAMs occupy a large silicon area, and embedded DRAMs increase the wafer manufacturing cost. Since buffers are expensive to implement and power-hungry during operation, on-chip networks should reduce the buffer size on the switches as much as possible.

2.3.3 On-Chip Protocols

Communication protocols in computer networks are strictly regulated by compatibility and adaptability requirements. In comparison, on-chip communication is constrained less strictly because NoC is a self-contained system. NoC communication may benefit from on-chip locality and adopt application-specific or platform-specific protocols that can best exploit the hardware resources.

In Chapter 1, we have described different approaches for on-chip network protocol implementation. In the forthcoming chapters, we will show that many parameters in NoC communication protocol, e.g., packet size, segmentation schemes, buffer sizes, etc., can be adjusted to achieve optimal tradeoffs between performance and energy consumption. More detailed analysis will be performed in those chapters.

2.3.4 System Software and Application

Current and future NoC platform will be highly programmable, and therefore the NoC performance and power consumption will critically depend on software aspects. The system software provides the users with an abstraction of the underlying NoC hardware platform, which can be leveraged by the application developer to safely and effectively exploit the hardware's capabilities. Current SoC programming methodology is not capable for these requirements because of the following limitations.

1. The on-chip communication cost, which includes power consumption and latency, is not explicit in current SoC operating system and software development tools. Current SoC programming styles are based on a shared memory paradigm, which is appropriate for tightly coupled, small-scale SoCs with limited number of PEs and SEs. Shared memory abstraction tend to hide the cost and unpredictability of communication, which are destined to grow in a NoC platform.

2. Current SoC software development platforms are mostly geared toward single microcontroller with multiple coprocessors architectures. Most of the system

software runs on the control processor, which orchestrates the system activity and assigns computationally intensive tasks to domain-specific coprocessors. Microcontroller-coprocessor communication is usually not data-intensive (e.g., synchronization and re-configuration information), and most high-bandwidth data communication (e.g., coprocessor-coprocessor and coprocessor-IO) is performed via shared memories and DMA transfers. The orchestration activities in the micro-controller are performed via run-time services provided by single-processor real-time operating systems, which differentiate from standard operating systems in their enhanced modularity, reduced memory footprint and support for real-time scheduling and bounded time interrupt service times.

3. Current SoC application programming is mostly based on manual partitioning and distribution of the most computationally intensive kernels to data coprocessors (e.g., VLIW multimedia engines, digital signal processors, etc.). After partitioning, different code generation and optimization approaches are used for each target coprocessor and the control processor.

NoC system software needs a paradigm shift. On-chip communication cost (including latency and power consumption) should be made explicit throughout all step of the NoC software development flow. Software programming and analysis should identify communication bottlenecks and power dissipation hot spots on the system. We envision the NoC system and programming paradigm should be emphasized on the following issues.

1. The NoC operating system cannot be centralized. Truly distributed embedded OSes are required [8] to create a scalable run-time system. The NoC OS should natively support power management. Each PE node and SE node in NoC micronetworks should be power-manageable, with individually controllable clock speeds and supply voltages.

2. Future NoC system should optimize and automate the task mapping procedure. Because a communication-optimal task mapping leads to balanced throughput and/or shorter latency. The on-chip communication energy consumption can

also benefit from an optimized task mapping, because in many cases, local congestion requires more network resources (e.g., buffers or wires) to solve the contention, and consequently cost high power consumption.

System and application software are very critical issues for future NoC designs. We believe that the full potential of on-chip networks can be effectively exploited only if adequate software abstractions and programming aids are developed to support them.

2.4 Summary

In this chapter, we outlined many critical areas that need to be emphasized in designing NoC platforms. Many of these areas require a paradigm shift from the traditional SoC design methodologies. In the next few chapters, we will focus on the NoC communication architecture analysis and designs. In particular, we will discuss the power modeling, routing and packetization issues.

Chapter 3

On-Chip Network Energy Model

3.1 Introduction

Whereas computation and storage energy greatly benefits from device scaling (smaller gates, smaller memory cells), the energy for on-chip communication does not scale down. On the contrary, projections based on current delay optimization techniques for global interconnects [25, 45, 46] show that global communication on chip will require increasingly higher energy consumption. Hence, communication-energy minimization will be a growing concern in future technologies.

In this chapter, we introduce a framework to estimate the power consumption of on-chip networks. We propose different modeling methodologies for node switches, internal buffers and interconnect wires inside on-chip network architectures. A simulation platform is also implemented to trace the dynamic power consumption with bit-level accuracy. Using this framework, four on-chip network architectures are analyzed under different traffic throughput and different numbers of ingress/egress ports. This framework and analysis will be applied in later chapters in the NoC architectural exploration and performance analysis.

In direct networks, the routing and arbitration functions are embedded into each node processor. In comparison, indirect networks have dedicated node switches and buffers. Nevertheless, when we consider the power consumption of the on-chip networks (both direct and indirect), we can estimate the power from the following three components:

1. The internal node switches, located on the intermediate nodes between node processors.

2. The internal buffers, used to temporarily store the packets when contention between packets occurs.

3. The interconnect wires that connect node switches.

The power consumption on these three components changes differently under different traffic loads and configurations. Therefore, they need to be analyzed with different modeling methodologies.

In this chapter, we propose different power consumption models for each of the above three components. We then apply these models on four widely-used indirect switch fabric architectures:

1. Crossbar

2. Fully Connected

3. Banyan

4. Batcher-Banyan

We will also use these power models on direct networks in the later chapters when we analysis the routing and packetization issues of on-chip networks.

A Simulink based multi-threading simulation platform is created for this analysis. It performs a time domain simulation with bit-level accuracy. The power consumption of every bit in the traffic flow is traced as the bit moves among the components inside switch fabrics.

Previous approaches for switch network power estimation are either based on statistical traffic models [49], or analytical models [37] [33]. The simulation is performed on gate or circuit levels, it is time consuming and not practical for large on-chip network designs. Furthermore, these approaches are not suitable for tracing the power consumption dynamically in real-time network traffic conditions. For example, the power consumption on internal switch buffers depends on the dynamic contention between packets. Compared with previous approaches, our modeling is an architectural-level estimation with bit-level accuracy. It traces the power consumption based on dynamic packet dataflows. This approach is well-suited for architectural design exploration as well as application specific power analysis.

3.2 Indirect Networks Switch Fabrics

In indirect networks, switch fabrics are the intermediate circuits that are interconnected between node processors. Switch fabric circuits consist of the node switches, interconnect wires and buffers. Generally, the ports that connect switch fabrics with node processors are often referred as *ingress* and *egress* ports.

Different switch fabric architectures affect the network performance (e.g. throughput, delay, power, etc.) differently. In the following sections, we will focus on the power consumption issues of switch fabrics and estimate the power consumption of different switch fabric architectures with different numbers of ingress and egress ports.

3.3 Power Modeling with Bit Energy

As introduced previously, the power consumption on switch fabrics comes from the following three different sources:

1. The internal node switches.
2. The internal buffer queues.
3. The interconnect wires.

Inside the switch fabrics, different packets travel on different data paths concurrently, and the traffic load on each data path may change dramatically from time to time. To estimate the dynamic power consumption in this multi-process interconnect network, we propose a new modeling approach: the *Bit Energy* E_{bit}.

The bit energy E_{bit} is defined as the energy consumed by each bit when the bit is transported inside the switch fabrics from ingress ports to egress ports. The bit energy E_{bit} is the sum of the bit energy consumed on node switches, $E_{S_{bit}}$, on internal buffers, $E_{B_{bit}}$ and on interconnect wires, $E_{W_{bit}}$. We will analyze these three bit energies in details in the following sections.

3.3.1 Switch Power Consumption

Switches are located on the intermediate nodes inside switch fabrics. They direct the packets from one intermediate stage to the next stage until reaching the destination ports. In different switch fabric topologies, node switches may have different functions and node degrees. For example, in Banyan switch fabrics, the node switch is a 2×2 switch and has degree of 4.

When a data bit travels through a node switch, the logic gates on the data path inside the node switch consume power as they toggle between power rails. We will analyze the 2×2 switch used in Banyan switch fabrics as an example (Fig. 3.1).

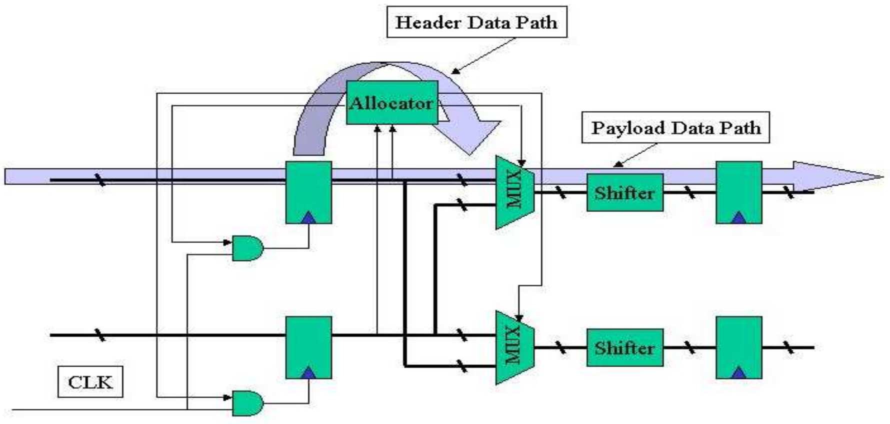

Figure 3.1: A 2×2 Node Switch in Banyan Switch

The 2×2 switch directs the packets from its two inputs to the outputs, according to the destination addresses of the packets. The ingress process unit had already parallelized the serial dataflow on the transmission line into a parallel bus dataflow (16-bit or 32-bit wide), so the destination address can be read out at one clock cycle. The destination bits are first detected by the allocator. If the destination port is available, the allocator allocates the output port to the packet and preserves the allocation throughout the packet transmission. The allocation process and packet transmission process are denoted in Fig. 3.1 as "header data path" and "payload data path" respectively. Both processes consume energy when packet bits travel through the data paths. The energy consumed by the header bits is actually different from the payload bits. However, the header normally occupies a small portion of the entire packet. Without loss of generality, we will use the payload bit energy as the node switch bit energy $E_{S_{bit}}$ in our analysis.

In reality, the bit energy $E_{S_{bit}}$ also depends on the presence or absence of packets on other ports of the node switch. For example, the switch will consume more power to process two packets at the same time, but the power consumption is not necessarily twice as much as that of processing a single packet. Therefore, the bit energy $E_{S_{bit}}$ is an input state-dependent value and should be expressed in an input vector indexed look-up table. For a switch with n input ports, there will be 2^n different input vectors with different $E_{S_{bit}}$ values.

In our analysis, the look-up table is pre-calculated from Synopsys Power Compiler simulation. The node switch circuit is first simulated based on input vectors. Then, the switching activities on every gate are also traced. Last, the power consumption of the entire circuit is estimated. We simulate different combinations of the input vectors and the results are shown in details in Section 3.5. Using the look-up table, the power consumption on the switch node can be estimated under different traffic conditions.

3.3.2 Internal Buffer Power Consumption

When contention occurs between packets, internal buffers are needed to temporarily store the packets with lower priorities (Fig. 3.2). There are two different types of contention between ingress packets, namely, the *destination contention* and *interconnect contention.*

- *Destination contention* – When there are two or more packets in the ingress ports requesting the same destination port at the same time, *destination contention* will occur. This type of contention is application dependent. In our analysis, we assume the arbiter had already solved this type of contention before the ingress process unit delivers the packets to the switch fabrics.

- *Interconnect contention (Internal Blocking)* – Inside switch fabric circuits, the same interconnect link may be shared by packets with different destinations. The contention on the shared interconnects is called *interconnect contention* or *internal blocking*. It happens inside switch fabric interconnect networks and it is architectural dependent.

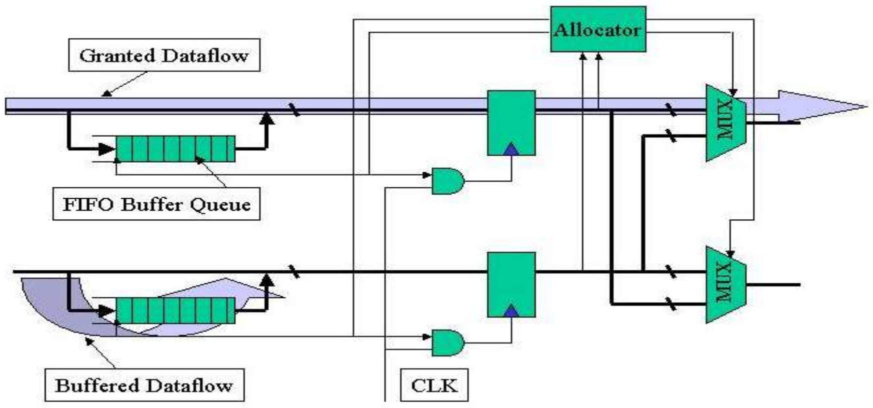

Figure 3.2: Buffers in a 2 × 2 Node Switch

In switch fabric circuits, the buffers are normally implemented with shared SRAM or DRAM memories. The energy consumption in buffers comes from two sources: 1) the data access energy, consumed by each READ or WRITE memory access operation, and 2) the refreshing energy, consumed by the memory refreshing operation (in the case of DRAM). The bit energy on the internal buffers $E_{B_{bit}}$ can be expressed by the following equation(Eq. 3.1):

$$E_{B_{bit}} = E_{access} + E_{ref} \tag{3.1}$$

where E_{access} is the energy consumed by each access operation and E_{ref} is the energy consumed by each memory refreshing operation. In reality, memory is accessed on word or byte basis instead of a single bit, the E_{access} is actually the average energy consumed for one bit.

The energy consumed by memory access is determined by the contention between the ingress packets. As discussed earlier, *destination contention* is application dependent, regardless of what switch fabric circuits are used. In comparison, *interconnect contention* is switch-fabric architecture dependent. Different architecture topologies result in different contention occurrence. In this chapter, we are interested in comparing the power consumption on different switch fabric architectures under the same network traffic, therefore, we assume the *destination contention* has already been resolved by the arbiter before the ingress packets are delivered to the switch fabrics. We only compare the internal buffer energy consumption occurred from *interconnect contention.*

3.3.3 Interconnect Wires Power Consumption

When the node switch delivers one bit with flipped polarity to the interconnect wires, the signal on the wire will toggle between logic "0" and logic "1". Energy is dissipated in this charging or discharging process. Only bits with flipped polarity consume energy, namely, $E_{W_{bit0\to 1}}$ or $E_{W_{bit1\to 0}}$ have bit energy values, $E_{W_{bit0\to 0}} = 0$, $E_{W_{bit1\to 1}} = 0$.

Assuming a rail-to-rail toggling, the bit energy on the interconnect wires $E_{W_{bit}}$ for bit 1→0 and 0→1 can be described by the following equation (Eq. 3.2).

$$E_{W_{bit}} = \frac{1}{2}C_{wire}V^2\alpha + \frac{1}{2}C_{input}V^2\alpha = \frac{1}{2}C_W V^2\alpha \tag{3.2}$$

Here C_{wire} is the wire capacitance on the interconnect, C_{input} is the total capacitance of the input gates connected to the interconnect. $C_W = C_{wire} + C_{input}$ is the total load capacitance propagated by that bit. The rail-to-rail voltage is denoted by V, assuming the CMOS gates are switching from Vdd to GND. The activity factor α ranges from 0 to 1. In our simulation, we assume a random bit-stream in the dataflow, therefore, $\alpha = 0.5$.

The bit transmitted on interconnect wires consumes energy only when its polarity is flipped from previous transmitted bit. The switching activities on interconnects can be traced by our simulation approach proposed in Section 3.5.

3.3.4 Interconnect Wire Length Estimation

The wire capacitance C_{wire} is a function of wirelength and coupling capacitance between adjacent wires [25]. Estimation of interconnect wirelength of the switch fabric network is essential for the bit energy calculation on wires. Here we adopt the *Thompson model* [47] for wirelength estimation.

The Thompson Model

The Thompson model is based on a graph embedding process that maps a network connectivity graph (referred as source graph G) to a two dimensional grid floorplan (referred as destination graph H). The mapping process is described below and also illustrated in Fig. 3.3.

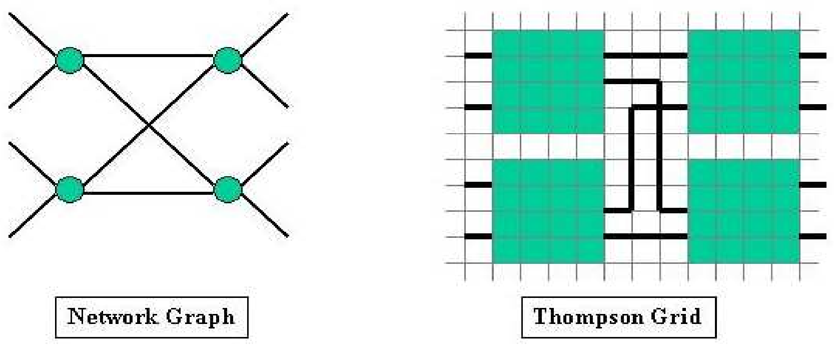

Figure 3.3: Thompson Wire Length Estimation Model and Mapping Process

We are given a source graph $G(V_G, E_G)$, where $V_{G_i}(0 < i \leq n)$ are the vertices of graph G and $E_{G_i}(0 < i \leq m)$ are the edges. The source graph represents the network topology. The target graph is denoted by $H(V_H, E_H)$, where V_H and E_H are the vertices and edges of H. Graph H is a 2-dimensional grid mesh consisting of p columns and q rows. An embedding of graph G into graph H is performed as follows. Each vertex in G is mapped into a $d \times d$ square of vertices in H, where d is the degree of vertex $v_i \in V_G$ and no more than one vertex in V_G occupies the same vertex in V_H. Each edge in G is mapped into one or more edges of graph H, and no more than one edge in E_G occupies the same edge in graph H. The optimal Thompson embedding of graph G into graph H is to find the minimum number of columns p_{min} and rows q_{min} in H that are needed for this embedding. The interconnect wirelength is defined as the number of grids that an edge E_G covers.

Wire Length Estimation Using Thompson Model

In our approach, we manually map the switch fabric topologies into Thompson grids and estimate the interconnect wirelength by counting the number of grids the interconnect covers. The detailed mapping of each particular switch fabric topology will be shown in Section 3.4.

The Thompson model is only a global wirelength estimation. It is not as accurate as detailed wire-routing, but it is an effective way of architectural planning for interconnect networks, especially when the network topology is regular. In the case of switch fabrics, it is straightforward to map the regular switch nodes and interconnects into a 2-dimensional mesh of regular rows and columns. Therefore, the Thompson model is an ideal model for switch fabric wirelength estimation.

For an interconnect wire of length equal to one Thompson grid, we define the wire bit energy consumption as $E_{T_{bit}}$. If an interconnect wire has its length equal to m Thompson grids, its wire bit energy is $E_{W_{bit}} = m \times E_{T_{bit}}$.

3.4 Switch Fabric Architectures

With the aforementioned power consumption model, we will analyze four widely-used switch fabric architectures in this section.

3.4.1 Crossbar Switch Fabrics

Crossbar topology (Fig. 3.4) uses space division multiplexing for input-output connection. Every input-output connection has its own dedicated data path, therefore, crossbar is *interconnect contention* free.

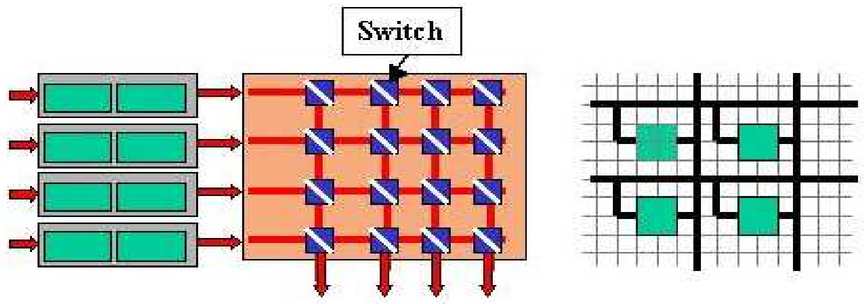

Figure 3.4: Crossbar Switch Fabrics

The node switch on the crosspoint of crossbar network can be a simple CMOS pass gate, or a tri-state CMOS buffer. Both are relatively simple compared to the node switches used in other network topologies.

Every bit will propagate throughout the long interconnect wires that connect to the input port (the row interconnect, in Fig. 3.4) and output port (the column interconnect). It also toggles the input gates of all the node switches connected to the same row. The load on the input port is the total of the wire capacitance and the sum of all input capacitances of N switches.

Some crossbar switch networks use buffers at every crosspoint to solve the *destination contention* problems. As discussed earlier, we assume the *destination contention* is already resolved by the arbiter, i.e., there are no buffers needed in the power modeling of crossbar network.

A Thompson embedding of crossbar switch network is also shown in Fig. 3.4. Under the Thompson model, the mapping is straightforward and the total bit energy for the crossbar switch fabrics is described in Eq. 3.3.

$$E_{bit_{crossbar}} = N \times E_{S_{bit}} + 8N \times E_{T_{bit}} \tag{3.3}$$

where $E_{S_{bit}}$ is the bit energy for the switch and $E_{T_{bit}}$ is the bit energy of a Thompson grid wire. Each crossbar node switch has degree of 4, however, two of the ports are used as feed-through ports, so we assume it occupies 2×2 Thompson grids. Two extra rows/columns are also needed for horizontal and vertical interconnects for each node switch. Each bit traveling from input i to output j will propagate both the interconnect wires connected to the input port i and output port j, each of the interconnect has length of $4N$ of a Thompson grid.

Crossbar has the benefit of being free of *interconnect contention*. However, the bit energy will increase linearly with the number of input and output ports N. The power consumption will be very high for switch fabrics with large port numbers.

3.4.2 Fully-Connected Network

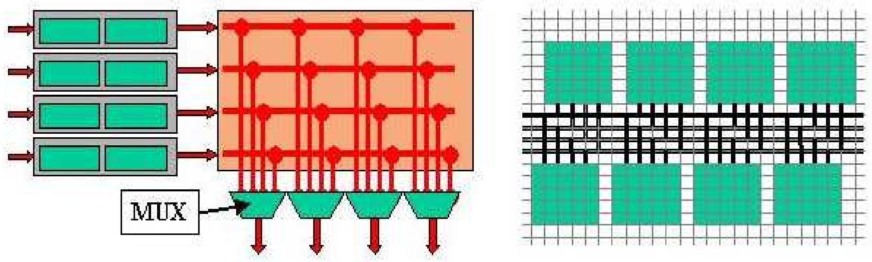

Figure 3.5: Fully Connected Switch Fabrics

Similar to the crossbar network, in fully connected switch network (Fig. 3.5), each source-destination connection has its dedicated data path (fully connected switch network is also often referred as crossbar). The network is also free of *interconnect contention*. There are no internal buffers needed in its power modeling.

The bit energy for a fully connected switch network is consumed on the interconnect wires and the MUXes. A Thompson embedding is shown in Fig. 3.5, where the MUXes are placed in a double-row fashion. The bit energy can be estimated with the following equation (Eq. 3.4).

$$E_{bit_{fullyconn}} = E_{S_{bit}} + \frac{1}{2}N \times N \times E_{T_{bit}} \tag{3.4}$$

where $E_{S_{bit}}$ is the bit energy on the MUX. Compared with crossbar switch, each bit only consumes energy on one of the MUXes, instead of N switches as in the case of crossbar. However, the N-input MUX has more complicated logic gates, and its power consumption and complexity scale up with the number of inputs N.

3.4.3 Banyan Network

Banyan network (Fig. 3.6) has $N = 2^n$ inputs and $N = 2^n$ outputs, where n is the dimension of Banyan. Therefore, the total number of switches is $\frac{1}{2}N\log_2 N$ in n stages, each stage is referred as stage i where $0 \leq i < n$ [9].

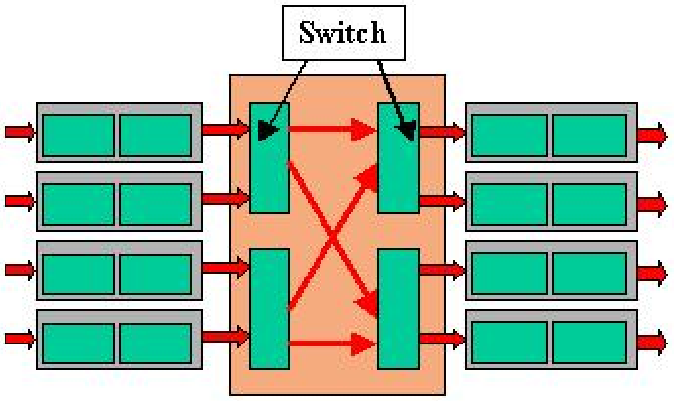

Figure 3.6: Banyan Switch Fabric Network

The switch used in a Banyan network is a binary switch, as described in Section 3.3. It has two inputs and two outputs. The input packet with destination bit *"0"* or *"1"* goes to output *"0"* and *"1"* respectively. Stage i in the Banyan network checks the i^{th} bit of the destination address of the packet, therefore, the packet will be routed automatically from stage 0 to stage $n-1$. This routing scheme is also called self-routing switch fabrics. If two input packets have the same destination bit coming at the same time, one of the input packets will be buffered.

Banyan network has *interconnect contention* problems [9]. The same interconnect might be shared by different data paths. A buffer is needed at each internal node switch.

The binary switches in Banyan network have more complex logic as compared to the crosspoint node switches in crossbar network. A binary switch also consumes more power when bit is switched from the input port to the output port.

A simple Thompson embedding of Banyan network is shown in Fig. 3.3. Detailed analysis of Thompson embedding of Banyan isomorphic networks can be found in [16]. Using the Thompson model, the longest interconnect wirelength for stage i of

Banyan network can be estimated as 4×2^i Thompson grid. Different input/output connections have different interconnect data paths. The worst case bit energy of Banyan network (the longest interconnects a bit may travel) can be estimated using Eq. 3.5 below.

$$E_{bit_{Banyan}} = \sum_{i=0}^{n-1} q_i E_{B_{bit}} + 4 \sum_{i=0}^{n-1} 2^i E_{T_{bit}} + nE_{S_{bit}} \tag{3.5}$$

where $N \geq 2$ and $n \geq 1$. q_i has the value of 0 or 1. When there is a contention at stage i, q_i is 1, otherwise it is 0. The value of q_i is determined by the contentions between packets on the interconnect.

3.4.4 Batcher-Banyan Network

To solve the *interconnect contention* problem, the Batcher-Banyan network architecture is introduced, as shown in Fig. 3.7. The contention is solved by the Batcher sorting network, followed by the Banyan network. After sorting network, each input-output connection will have its own dedicated path, therefore there is no *interconnect contention* [9].

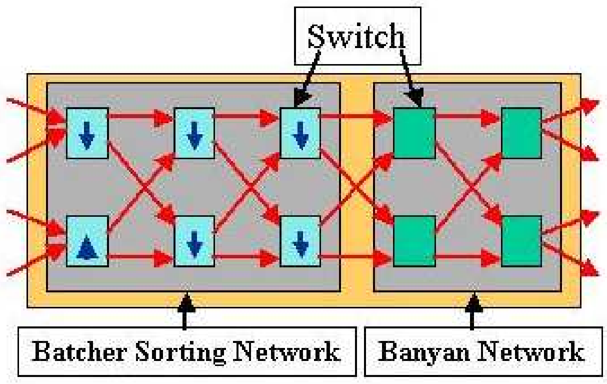

Figure 3.7: Batcher-Banyan Switch Fabric Network

Although Batcher-Banyan network solves the *interconnect contention* problem, it pays the price by increasing the number of stages between the inputs and outputs. It has total of $\frac{1}{2}(log_2 N)(log_2 N + 1)$stages, which will in turn increase the bit energy consumed on switches and interconnect wires.

The Thompson embedding of Batcher-Banyan is very similar to Banyan network, except it has more stages. The worst case bit energy for Batcher-Banyan network can be expressed in the following equation (Eq. 3.6).

$$E_{bit_{Batcher}} = 4\sum_{j=0}^{n-1}\sum_{i=0}^{j} 2^i E_{T_{bit}} + 4\sum_{i=0}^{n-1} 2^i E_{T_{bit}} \tag{3.6}$$

$$+\frac{1}{2}n(n+1)E_{SS_{bit}} + nE_{SB_{bit}} \tag{3.7}$$

where $N \geq 4$ and $n \geq 2$. Because the sorting switch used in Batcher sorting network is different from the binary switch used in Banyan network. we denote $E_{SS_{bit}}$ for the bit energy of sorting switches and $E_{SB_{bit}}$ for the bit energy of binary switches.

3.5 Bit-Energy Calculation and Experiments

In this chapter, we present a framework to estimate switch fabric power consumption. The calculation and results are based on case studies. The accurate values of each parameter for a specific switch fabric implementation depend on particular circuit design techniques and technologies. However, the methodology introduced here can be applied in other cases.

3.5.1 Bit Energy Calculation

1. Bit energy of Node Switches

As discussed in Section 3.3, the bit energy of node switches is state-dependent, it depends on the presence or absence of the packets on other input ports. For a node switch with n-input ports, there are 2^n different input vectors with different bit energy values. Normally switch fabrics with a large input/output ports are constructed from node switches with smaller number of degrees (2×2 or 4×4), therefore, the vector number 2^n is not prohibitively large. In this chapter, the bit energy is pre-calculated from Synopsys Power Compiler simulation. We build each of the node switches with 0.18μm libraries, apply different input vectors and calculate the average energy consumption on each bit. The circuits for each node switch range from a few hundred gates to 10K gates, therefore, the simulating time is very quick (tens of seconds, in most cases). The bit energy results for crossbar switches, the N-input MUXes, the Banyan binary switches, the Batcher sorting switches are listed in Table 3.1.

Table 3.1: Bit Energy Under Different Input Vectors

Switch Fabric Architectures	Input Vector	Bit Energy	Input Vector	Bit Energy
		$10^{-15} joule$		$10^{-15} joule$
Crossbar 1×1	[0]	0	[1]	220
Banyan 2×2	[0,0]	0	[0,1]	1080
	[1,0]	1080	[1,1]	1821
Batcher 2×2	[0,0]	0	[0,1]	1253
	[1,0]	1253	[1,1]	2025
N-input MUX	N = 4	431		
	N = 8	782		
	N = 16	1350		
	N = 32	2515		

The input vectors for the 2×2 switches in the above table indicate presence or absence of the packets on the corresponding input ports, e.g. [1,0] means only input port 0 has a packet coming in. For the N-input MUXes, bit energy values are very close among different input vectors, but they increase with the number of inputs N, as shown in the table.

2. Bit Energy of Buffer Queues

We use SRAM as the shared buffers inside Banyan switch fabrics. We adopt techniques similar to those proposed by [19] and [42] to estimate the memory access power consumption. SRAMs with different sizes have different access time and current, therefore they have different memory access power dissipation. The buffer size at each node switch greatly affects the performance of Banyan switch. The trade-off analysis between buffer size and switch throughput is beyond the scope of this research. Researches in [53] and [36] show that buffer size of a few packets will actually achieve ideal throughput under most network traffic conditions. In our experiments, we use 4K bit buffer queue for each Banyan node switch. Based on the buffer size of each switch, we calculate the size of the shared memory. The memory access energy consumption can then be estimated based on selected memory size.

An off-the-shelf 0.18μm 3.3V SRAM is used as a reference for bit energy estimation. The calculation of access energy is based on 133MHz operation with access time specified in the data sheet. The results are shown in Table 3.2.

Table 3.2: Buffer Bit Energy of $N \times N$ Banyan Network

In/Out Size	Number of Switches	Shared SRAM Size	Bit Energy ($10^{-12} joule$)
4×4	4	16K	140
8×8	12	48K	140
16×16	32	128K	154
32×32	80	320K	222

3. Bit Energy of Interconnect Wires

The length of the Thompson grid can be estimated from the bus pitch distance of the interconnect. In Thompson model, each interconnect is a signal bus and occupies one grid square. Assuming the bus width is 32 bit, in 0.18μm technology, the wire pitch of global buses is around 1μm, therefore, the Thompson grid is around 32μm. The interconnect wire capacitance can be calculated from the method presented in [25]. For a global wire in 0.18μm technology, the wire capacitance is around 0.50fF/μm. Using these estimations, under 3.3V, the bit energy on interconnect wire of a Thompson grid length $E_{T_{bit}} = 87 \times 10^{-15}$ joule.

Comparing the buffer bit energy $E_{B_{bit}}$ values in Table 3.2 with the interconnect wire bit energy $E_{T_{bit}}$ value calculated above, we can see that storing a packet in buffer consumes far more energy than transmitting the packet on interconnect wires. This "buffer penalty" indicates that energy consumed in buffers is a significant part of total energy consumption of switch fabrics, and the buffer energy will increase very fast as the packet flow throughput increases. Our experiments in Section 3.6 will show this result.

3.5.2 Simulation Platform

The bit energy introduced in Sections 3.3 and 3.4 is the energy consumption for one bit. In switch fabric interconnect networks, different packets will travel on different data paths. To calculate the total power consumption of the entire switch fabrics, we need to trace the dataflow of every packet and summarize the bit energy of every bit on nodes switches, internal buffers and interconnect wires. In this chapter, we describe a Simulink based bit-level multi-threading simulation platform.

The complete switch fabric architecture is implemented in Simulink. The ingress process units, the egress process units, the global arbiter and different switch fabric architectures are all written in C++ and then compiled into Simulink S-functions. The switch fabric architecture is constructed hierarchically. The activities of every bit in the packet are traced at every node switches, every buffers as well as interconnect wires (all implemented in S-functions).

A randomized packet traffic flow is generated as inputs for the network router. Because we only need to simulate the switching activities inside the switch fabrics, the packet payloads are random binary bits. The address of each packet has been translated into destination port address by the ingress process unit. The arbiter uses the first-come-first-serve arbitration with round robin policy. The destinations of the packets are random. We use input buffer scheme to store the packets when there is *destination contention*. The input buffers are located at each ingress process unit. Because the input buffers are outside the switch fabric network, they are not counted for switch fabric power consumption.

Power consumption is measured under three metrics: 1) different traffic throughput, 2) different switch fabric architectures, and 3) different numbers of ingress and egress ports.

We implement four switch fabric architectures with different numbers of input/output ports, namely, 4×4, 8×8, 16×16 and 32×32. Packet dataflow is generated at each input port. The throughput of the packet dataflow can be adjusted by controlling the packet generation intervals. The throughput is measured at the egress process units. The throughput indicates the traffic loads that flow through the switch fabric networks.

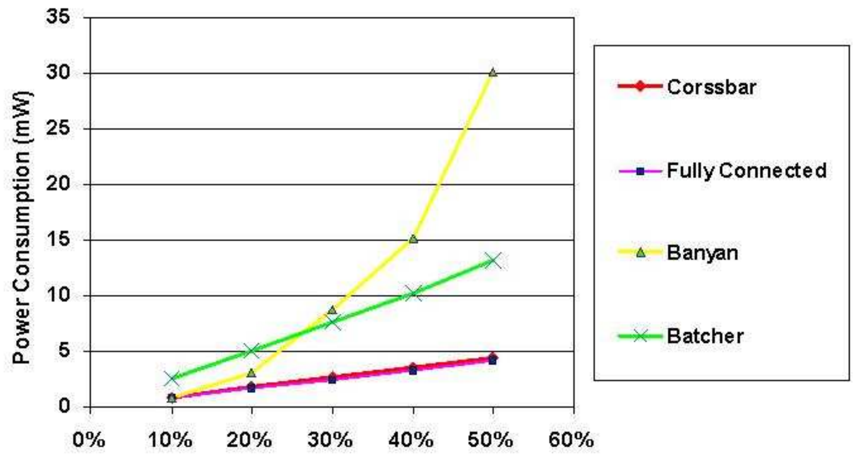

Figure 3.8: 4x4 Switch Power Consumption Under Different Traffic Throughput

3.6 Results and Analysis

Fig. 3.8 3.9 3.10 3.11 shows the power consumption of different switch fabric architectures under traffic throughput from 10% to 50%. The figures also show the power consumption/throughput relationship under different numbers of ingress/egress ports. Because we use input buffering scheme to store the packets with *destination contention*, the theoretical maximum throughput is 58.6% (measured at egress ports). In reality, the 58.6% throughput is not achievable [9].

From these results, we have the following observations:

1) *Interconnect contention* has a dramatic impact on the power consumption of Banyan switch. Banyan switch has the lowest power consumption under low traffic throughput, as the throughput increases, the power consumption increases exponentially. This is caused by the "buffer penalty" as discussed in Section 3.5. However, as the number of ingress and egress ports increases, the interconnect wirelength and bit energy also increase, the "buffer penalty" domination will have less impact. This can be seen by comparing the "Banyan curve" in the figures with the curves of other architectures. Actually, in the 32×32 configuration, Banyan had the lowest power consumption when the traffic throughput is less than 35%.

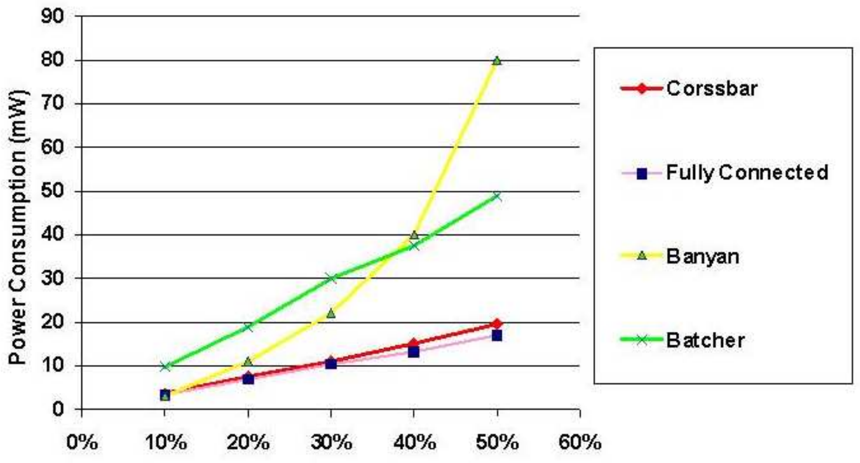

Figure 3.9: 8x8 Switch Power Consumption Under Different Traffic Throughput

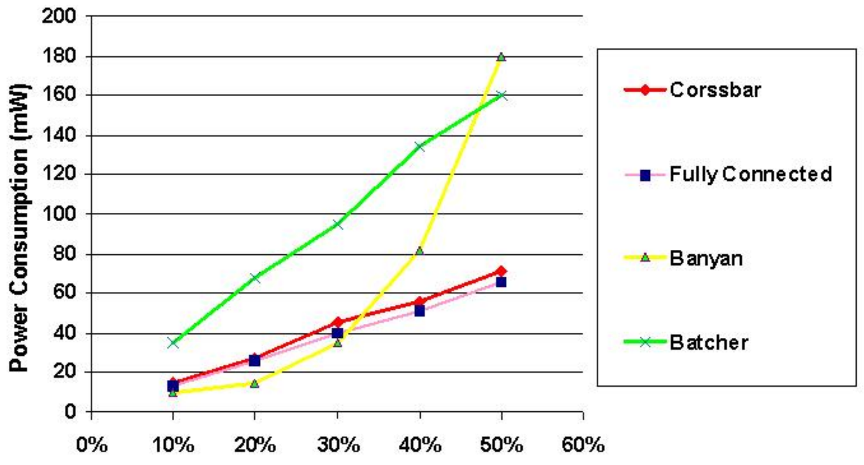

Figure 3.10: 16x16 Switch Power Consumption Under Different Traffic Throughput

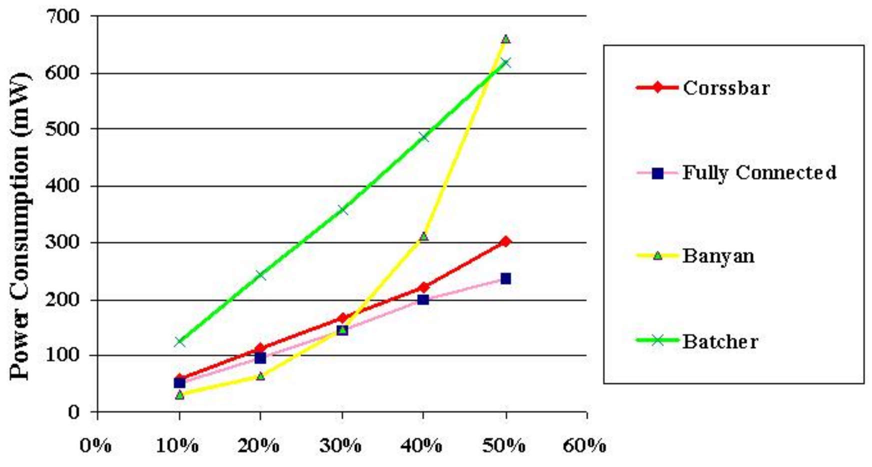

Figure 3.11: 32x32 Switch Power Consumption Under Different Traffic Throughput

2) *Fully connected switch* has the lowest power consumption among all four architectures with different numbers of ports. But the difference with Batcher-Banyan narrows down as the number of ports increases. This is because in switch fabrics with a small number of ports, the power consumption on the internal node switches dominates, as the switch fabrics is getting bigger (more ports), interconnect power consumption will dominate.

The impact of number of ports on power can be better seen from Fig. 3.12. In this figure, power consumption of each architecture is compared with different number of ports. The traffic throughput is 50%. The power consumption difference between fully connected switch and Batcher-Banyan switch decreases from 37% in 4×4 switches to 20% in 32×32 switches.

3) The power consumption of crossbar, fully connected and Batcher-Banyan networks increases almost linearly with the increase of the traffic throughput, except the Banyan network, which is dominated by the power consumption on internal buffers. This observation is not surprising because data flow with higher throughput needs more power to process the bits along the data path.

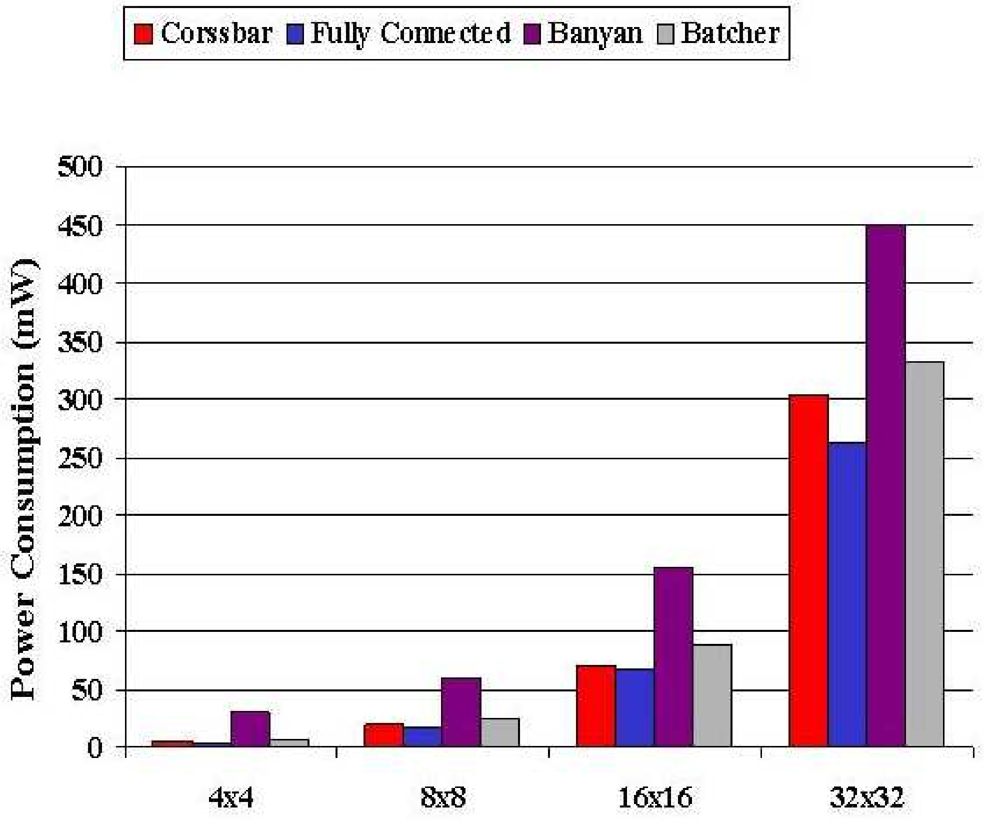

Figure 3.12: Power Consumption Under Different Number of Ports

3.7 Summary

A power consumption estimation framework of on-chip networks is proposed in this chapter. Using the framework, we analyze the power consumption of different indirect switch fabric architectures. From our analysis, we draw the following conclusions:

1) *Interconnect contention* (internal blocking) induces significant power consumption on internal buffers, and the power consumption on buffers will increase sharply as throughput increases.

2) For switch fabrics with a small number of ports, internal node switches dominate the power consumption, for switch fabrics with a larger number of ports (e.g. beyond 32×32), interconnect wires will gradually dominate the power consumption.

The analysis and comparison above are only based on case studies with specific technology parameters. Different implementations of switch fabrics will have different comparison results. However, the methodology presented in this chapter is general, it will be applied to different on-chip network power analysis in the following chapters.

Chapter 4

On-Chip Network Routing Algorithms Analysis

4.1 Introduction

On-chip networks may borrow features and design methods from those used in parallel computing clusters and computer system area networks. Nevertheless, they differ from traditional networks because of larger on-chip wiring resources and flexibility, as well as constraints on area and energy consumption (in addition to performance requirements). We believe these different aspects will require new methodologies for both the on-chip switch designs as well as the routing algorithm designs. In particular, we explore the following directions in the MPSoC networks-on-chip design.

- **Network architecture** – The on-chip network architecture should utilize the abundant wiring resources available on silicon. Control signals need not be serialized and transmitted along with data, but can run on dedicated control wires (Fig 4.1). The usage of buffers should be limited to reduce the area and energy consumption.

- **Routing algorithm** – On-chip routing should use those algorithms that do not require substantial on-chip buffer usage. At the same time, the network state

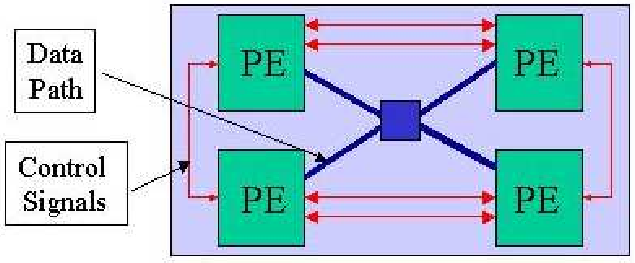

Figure 4.1: Dedicated Control Wires and Data Paths for On-Chip Network

(including contention information) can be made available through dedicated control wires. From this perspective, it is possible to achieve *contention-look-ahead* to reduce contention occurrence and increase bandwidth.

In this chapter, we analyze different routing schemes for packetized on-chip communication on a mesh network architecture, with particular emphasis on specific benefits and limitations of silicon VLSI implementations. A contention-look-ahead on-chip routing scheme is proposed. It reduces the network delay as well as power consumption with significantly smaller buffers. The improvement is also quantified by the network/multiprocessor co-simulation benchmarks results.

4.2 Packet Switching Techniques

In computer networks, different techniques are used to perform packet switching between different nodes. Popular switching techniques include *store-and-forward*, *virtual cut-through* and *wormhole*. When these switching techniques are implemented in on-chip networks, they have different performance metrics along with different requirements on hardware resources.

4.2.1 Store-and-Forward Switching

In many computer networks, packets are routed in a *store-and-forward* fashion from one router to the next. Store-and-forward routing enables each router to inspect the passing packets, and therefore perform complex operations (e.g., content-aware packet routing). When the packet size is big enough, store-and-forward routing not only introduces extra packet delay at every router stage, but it also requires a substantial amount of buffer spaces because the switches may store multiple complete packets at the same time.

In on-chip networks, storage resources are very expensive in terms of area and energy consumption. Moreover, the point-to-point transmission delay is very critical. Therefore, store-and-forward approaches are disadvantageous for on-chip communications.

4.2.2 Virtual Cut Through Switching

Virtual cut through (VCT) switching reduces the packet delays at each routing stage. In VCT switching, one packet can be forwarded to the next stage before its entirety is received by the current switch. Therefore, VCT switching reduces the store-and forward delays. However, when the next stage switch is not available, the entire packet still needs to be stored in the buffers of the current switch.

4.2.3 Wormhole Switching

Wormhole routing was originally designed for parallel computer clusters [17] because it achieves the minimal network delay and requires less buffer usage. In wormhole routing, each packet is further segmented into *flits* (flow control unit). The header flit reserves the routing channel of each switch, the body flits will then follow the reserved channel, the tail flit will later release the channel reservation.

One major advantage of wormhole routing is that it does not require the complete packet to be stored in the switch while waiting for the header flit to route to the next stages. Wormhole routing not only reduces the store-and-forward delay at each switch, but it also requires much less buffer spaces. One packet may occupy several intermediate switches at the same time. Because of these advantages, wormhole routing is an ideal candidate switching technique for on-chip multiprocessor interconnect networks.

4.3 Wormhole Routing Issues

Since wormhole switching has many unique advantages for on-chip network implementation, we will discuss the *deadlock* and *livelock* issues in this context, although these issues exist in other routing schemes as well.

4.3.1 Deadlock

In wormhole routing, one packet may occupy several intermediate switches at the same time. Packets may block each other in a circular fashion such that no packets can advance, thus creating a deadlock.

To solve the deadlock problem, the routing algorithms have to break the circular dependencies among the packets. *Dimension-ordered* routing [17][51] is one simple way to solve the deadlock: the packets always route on one dimension first, e.g., column first, upon reaching the destination row (or column), and then switch to the other dimension until reaching the destination. Dimension-ordered routing is deterministic: packets will always follow the same route for the same source-destination pair. Therefore, it cannot avoid contention. Whenever contention occurs, the packets have to wait for the channel to be free.

Another way to solve the deadlock problem is to use *virtual channels* [17][14]. In this approach, one physical channel is split into several virtual channels. Virtual channels can solve the deadlock problem while achieving high performance. Nevertheless, this scheme requires a large buffer space for the waiting queue of each virtual channel. For example, if one channel is split into four virtual channels, it will use four times as much buffer spaces as a single channel. The architecture proposed in [15] requires about 10K-bit of buffer space on each edge of the tile. The virtual channel arbitration also increases the complexity of circuit design.

4.3.2 Livelock

Livelock is a potential problem in many adaptive routing schemes. It happens when a packet is running forever in a circular motion around its destination. We will use the *hot potato* routing as an example to explain this issue.

Hot potato or *deflection* routing [18] is based on the idea of delivering a packet to an output channel at each cycle. It requires the assumption that each switch has an equal number of input and output channels. Therefore, input packets can always find at least one output exit. Under this routing scheme, when contention occurs and the desired channel is not available, the packet, instead of waiting, will pick any alternative available channels to continue moving to the next switch. However, the alternate channels are not necessarily along the shortest routes.

In hot potato routing, if the switch does not serve as the network interface to a node PE, packets can always find a way to exit, therefore the switch does not need buffers. However, if the PE nodes send packets to the network through the switch, input buffers are still needed, because the packet created by the node PE also needs an output channel to be delivered to the network. Since there may not be enough outputs for all input packets, either the packets from one of the input or the packets from the node processor have to be buffered [17].

In hot potato routing, if the number of input channels is equal to the number of output channels at every switch node, packets can always find an exit channel and they are deadlock free. However, *livelock* is a potential problem in hot potato routing. Proper deflection rules need to be defined to avoid livelock problem. The deflected routes in hot potato routing increase the network delays. Therefore, performance of hot potato routing is not as good as other wormhole routing approaches [17]. This is also confirmed by our experiments, as shown in Section 4.6.

4.4 Contention-Look-Ahead Routing

One big problem of the packet forwarding scheme in Section 4.2 and 4.3 is that the routing decision for a packet (or header flit) at a given switch ignores the status of the upcoming switches. A *contention-look-ahead* routing scheme is one where the current routing decision is helped by monitoring the adjacent switches, thus possibly avoiding blockages.

4.4.1 Contention Awareness

In computer networks, contention information in neighboring nodes cannot be transmitted instantaneously, because inter-node information can only be exchanged through packets. In comparison, on-chip networks can take advantage of dedicated control wires to transmit contention information.

A contention-aware hot-potato routing scheme is proposed in [35]. It is based on a two-dimensional mesh on-chip network. The switch architecture is similar to that in [31]. Each switch node also serves as network interface to a node processor (also called resource). Therefore, it has five inputs and five outputs. Each input has a buffer that can contain one packet. One input and one output are used for connecting the node processor. An internal FIFO is used to store the packets when the output channels are all occupied. The routing decision at every node is based on the "stress values", which indicate the traffic loads of the neighbors. The stress value can be calculated based on the number of packets coming into the neighboring nodes at a unit time, or based on the running average of the number of packets coming to the neighbors over a period of time. The stress values are propagated between neighboring nodes. This scheme is effective in avoiding "hot spots" in the network. The routing decision steers the packets to less congested nodes.

In the next section, we will propose a wormhole-based contention-look-ahead routing algorithm that can "foresee" the contention and delays in the coming stages using a direct connection from the neighboring nodes. It is also based on a mesh network topology. The major difference from [35] is that information is handled in flits, and thus large and/or variable size packets can be handled with limited input buffers. Therefore, our scheme combines the advantages of wormhole switching and hot potato routing.

4.4.2 Contention-look-ahead Routing

Fig. 4.2 illustrates how contention information benefits the routing decision. When the header flit of a packet arrives at a node, the traffic condition of the neighboring nodes can be acquired through the control signal wires. The traffic signal can be either a one-bit wire, indicating whether the corresponding switch is busy or free, or multiple-bit signal, indicating the buffer level (queue length) of the input waiting queue. Based on this information, the packet can choose the route to the next available (or shortest queue) switch. The local routing decision is performed at every switch once the header flit arrives. It is stored to allow the remaining flits to follow the same path until the tail flit releases the switch.

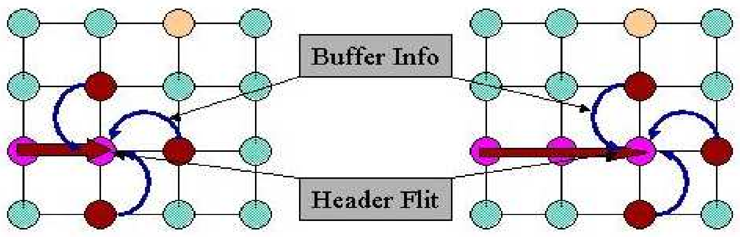

Figure 4.2: Adaptive Routing for On-Chip Networks

There are many alternate routes to the neighboring nodes at every intermediate stage. We call the route that always leads the packet closer to the destination a *profitable route*. Conversely, a route that leads the packet away from the destination is called a *misroute* [17] (Fig. 4.3). In mesh networks, profitable routes and misroutes can be distinguished by comparing the current node ID with the destination node ID. In order to reduce the calculation overhead, the profitable route and misroute choices for every destination are stored in a look-up table, and the table is pre-coded once the network topology is set up.

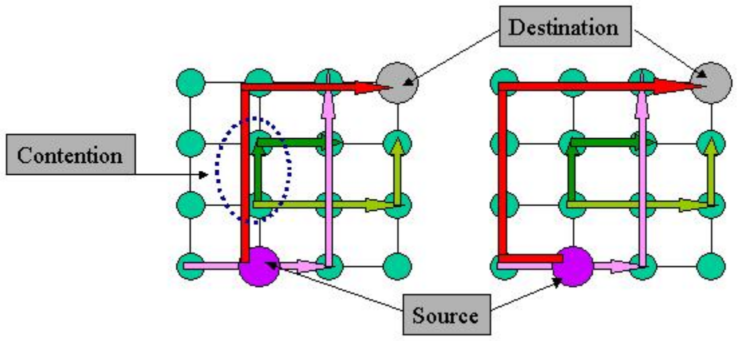

Figure 4.3: Profitable Route and Misroute

Profitable routes will guarantee the shortest path from source to destination. Nevertheless, misroutes do not necessarily need to be avoided. Occasionally, the buffer queues in all available profitable routes are full, or the queues are too long. Thus, detouring to a misroute may lead to a shorter delay time. Under these circumstances, a misroute may be more desirable.

4.4.3 Wormhole Contention-Look-Ahead Algorithm

For any packet entering an intermediate switch along a path, there are multiple output channels to exit. We call C the set of output channels. For a 2-dimensional mesh, $C = \{North, South, East, West\}$. We further partition C into profitable routes P and misroutes M. We define the buffer queue length of every profitable route $p \in P$ as Q_p. Similarly, we define the buffer queue length of every misroute $m \in M$ as Q_m.

Assume the flit delay of one buffer stage is D_B, and the flit delay of one switch stage is D_S. The delay penalty to take a profitable and a misroute is defined as D_{profit} and $D_{misroute}$, respectively, in the following equations.

$$D_{profit} = min(Q_p, \forall p \in P) \times D_B \tag{4.1}$$

$$D_{misroute} = min(Q_m, \forall m \in M) \times D_B + 2D_S \tag{4.2}$$

In a mesh network, when a switch routes a packet to a misroute, the packet moves away from its destination by one switch stage. In the subsequent routing steps, this packet needs to get back on track and route one more stage back towards its destination. Therefore, the delay penalty for a misroute is $2 \times D_S$, plus potential extra buffering delays at the misrouted stages. In our experiment, we use $2 \times D_S$ as the misroute penalty value. This value can be adjusted to penalize (or favor) more on misroute choices. In on-chip networks, the switch delay of one routing stage consists of the gate delays inside the switch logic plus the arbitration delays. The delay D_S can be estimated beforehand, and, without loss of generality, we assume the same D_S value for all switches in the network.

If all profitable routes are available and waiting queues are free, the packet will use dimension-ordered routing decision. If the buffer queues on all of the profitable routes are full or the minimum delay penalty of all the profitable routes is larger than the minimum penalty of the misroutes, it is more desirable to take the misroute. The routing decision evaluation procedure is described in the pseudo code below:

$$(D_{profit} \leq D_{misroute}) AND (Q_p \leq Q_{p_{max}})? ProfitRoute : Misroute \tag{4.3}$$

where $Q_{p_{max}}$ is the maximum buffer queue length (buffer limit). Fig. 4.4 illustrates how the queue length information is evaluated at each stage of the routing process.

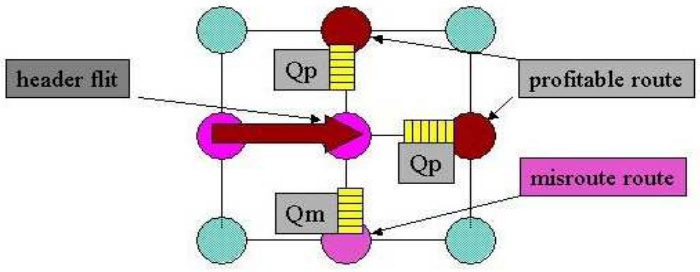

Figure 4.4: Adaptive Routing Algorithm

This routing algorithm is heuristic, because it can only “foresee” one step ahead of the network. It provides a local best solution but does not guarantee the global optimum. Nevertheless, we believe the proposed algorithm have many unique advantages. Compared to dimension-ordered routing, the proposed routing algorithm induces shorter delays on buffers because it is smarter in avoiding contention. Compared to hot-potato routing, the proposed routing algorithm will route faster because it evaluates the delay penalties in the forthcoming stages. This can be verified experimentally, as shown in Section 4.6.

4.5 On-chip Switch Design

We have designed a 2-dimensional mesh network to test the proposed routing scheme. The node processors are tiled on the floorplan (Fig. 4.5a). Each side of the tile has one input and one output. The switch also serves as network interface for the node

PE located at the center of the tile (Fig. 4.5b). The four inputs and four outputs of each tile are interconnected as shown in Fig. 4.5c. The switch supports concurrent links from any input channels to any output channels.

Because of the wormhole switching approach, the switch network can have limited storage and can accommodate packets with variable sizes. Because packets are segmented into flits, only one flit is processed at each input in one cycle. Each switch needs to store only a fraction of the whole packet. Long packets can be distributed over several consecutive switches and will not require extra buffer spaces. In comparison, the hot potato routing switch network described in [31] and [35] needs to handle the whole packet at every switch.

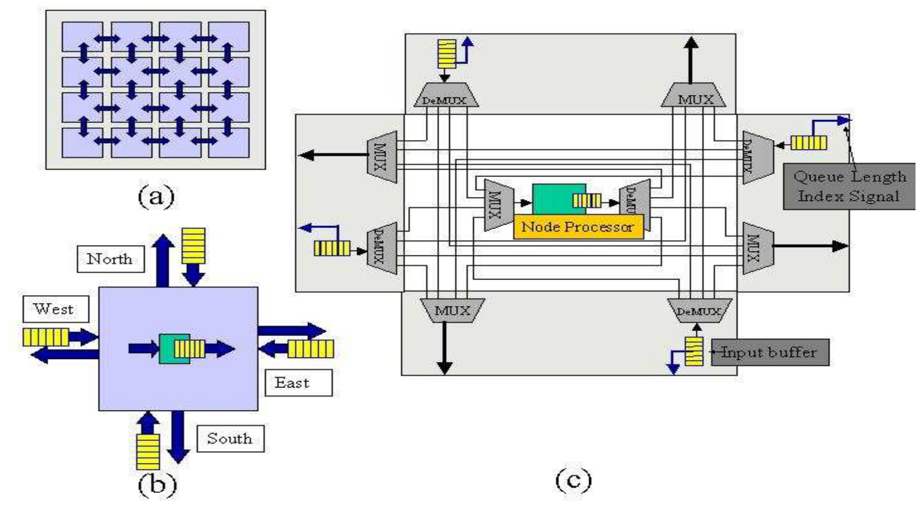

Figure 4.5: Switch Fabrics for On-Chip Networks

If the local PE is the source of the packet, the same contention-look-ahead algorithm is used. If no output is available, the node will hold the packet transmission. If the node is the destination of the packet, it will “absorb” this packet. Incoming packets will take priority over those generated/absorbed by the local PE.

The proposed switch network architecture and contention-look-ahead scheme can be applied to many existing wormhole routing algorithms. Because it foresees the contention occurrence and buffer queue length in the neighboring nodes, it helps the local nodes to make better decision to avoid potential livelock or deadlock problems.

The control signal wires are connected between any pair of neighboring nodes. The signal wires carry the input buffer queue length information of the corresponding route. The queue length value is encoded in a binary word, e.g., 1011 means the buffer queue length is 11 flit. The flit size is 64-bit, if each side of the tile uses a 2-flit buffer, with the internal queue included, the total buffer size for the switch is 640-bit.

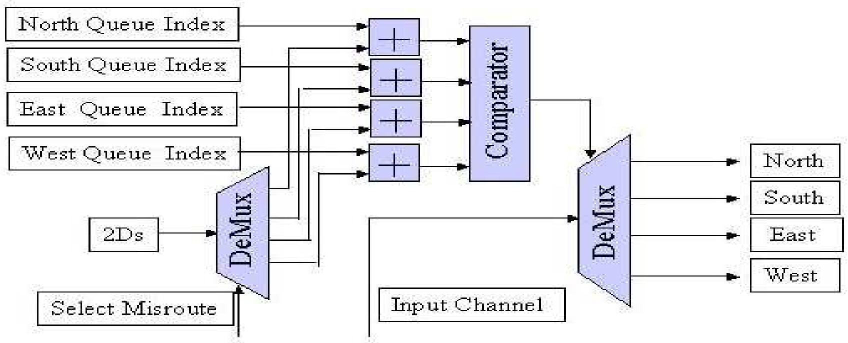

Figure 4.6: Allocator Circuit That Implements the Routing Algorithm

The control portion of the routing algorithm, defined by Eq. 4.1 to Eq. 4.3, is realized by a combinational logic module called *allocator*, shown in Fig. 4.6. The output channel is selected by DeMUX, and the selection is based on the comparator results of the delay penalty of each output channels. The delay penalty is either the buffer queue length of the corresponding input of the next node, or, in the case of a misroute channel, the sum of the queue length and $2 \times D_s$, which is the extra switch delay incurred with the misroute. Another DeMUX selects the misroute channels, because there could be multiple misroutes for a packet at each switch. This calculation involves two 4-input DeMUX delays, one adder delay and one comparator delay. It can be performed immediately after the address code in the header flit is available, thus minimizing the delay overhead. The switch also uses registers to store the decision taken by the header flit of a packet to keep a reserved path, until the tail flit resets it.

4.6 Experiments and Results

We perform both qualitative as well as quantitative analysis on MPSoC and its on-chip networks. The quantitative analysis is measured from the benchmark results. Therefore, we will describe our experimental platform first before proceeding to the detailed analysis.

We use RSIM, a multiprocessor instruction level simulator as our experiment platform [28]. The proposed on-chip network switch module as well as the routing algorithm are written in C and integrated into RSIM routing function. The multi-processor architecture is shown in Fig. 4.7.

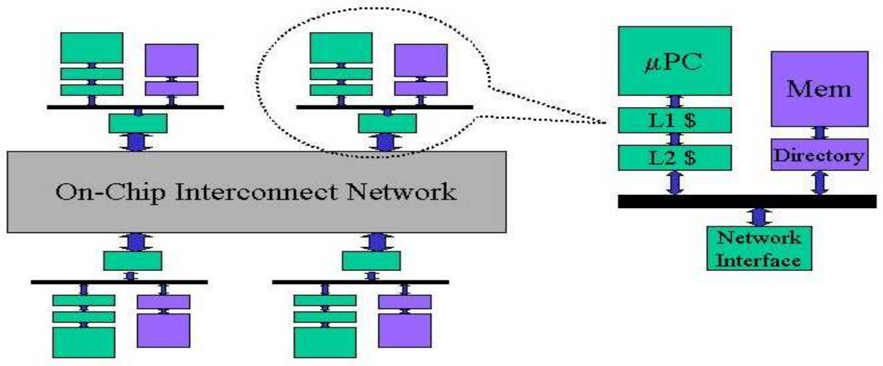

Figure 4.7: RSIM Multiprocessor Architecture

In our simulation, each processor element is connected to the interconnect network as a node. RSIM uses distributed shared memories. Memory modules are associated with each node, each module is globally addressed and accessible from any processors. Beside the interconnect data paths on the network, adjacent processors are also connected by control wires. The control wires deliver the input buffer information to the adjacent switches.

In multiprocessor systems-on-chip, the performance of node processors is closely coupled with the interconnect networks. On one hand, the delay of packet transmission on the network greatly affects the instruction execution of the node processors. On the other hand, the performance of the node processors will consequently affect the packet generation and delivery onto the network. Therefore, to evaluate our proposed on-chip communication architecture and routing algorithm, we need to measure the packet delay of the network as well as the performance of the processors.

In our model, 16 RISC processors are connected in a 4×4 (4-ary 2-cube) mesh network. We compared NoC routine schemes on four benchmarks: *quicksort, fft, lu* and *sor*. These benchmarks are ported from Stanford SPLASH suite [43] and running on the RSIM simulation platform. RSIM integrates detailed instruction-level models of the processors and a cycle-accurate network simulator. Both the network packet delays and the execution cycles of the node processors can therefore be measured and compared.

The proposed contention-look-ahead routing algorithm is compared with dimension-ordered routing and hot potato routing. The experiments are performed in the following metrics: 1) performance improvement, 2) buffer requirements, and 3) power consumption on the network. As mentioned earlier, virtual channel wormhole routing requires substantial buffer spaces ($40K$ bits, in the example in [15], as opposed to $512 \sim 4K$ bits for the switch buffers in our experiment). Therefore, we do not consider the virtual channel approach in our experiments.

4.6.1 Performance Improvements

Fig. 4.8, 4.9, 4.10 and 4.11 show the average packet delay on the interconnect network under the three routing schemes. The packet size is 64Byte. Contention-look-ahead routing is compared with the dimension-ordered routing with different input buffer sizes (2-flit, 4-flit, 8-flit, 16-flit). The hot potato routing input buffer size is fixed and is equal to one packet. The delays of the other two routing schemes are normalized to the hot potato results. The packet delay is measured from the header flit entering the network until the tail flit leaves the network. Delays are expressed in clock cycles. In all four benchmarks, the hot potato routing scheme has the longest network delays. This is because deflections create extra latency. The contention-look-ahead routing scheme achieves the shortest network delay under the same buffer size in all the benchmarks.

Larger buffer sizes help reducing the packet network delays. Although the buffer on the input channel of the switch is not big enough to store the entire packet, it can

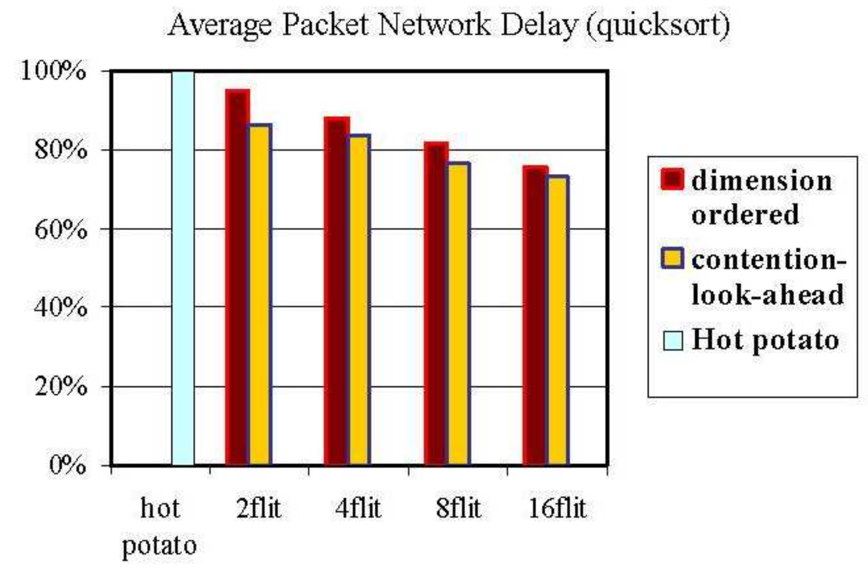

Figure 4.8: Averaged Packet Network Delays (quicksort) under Different Routing Schemes

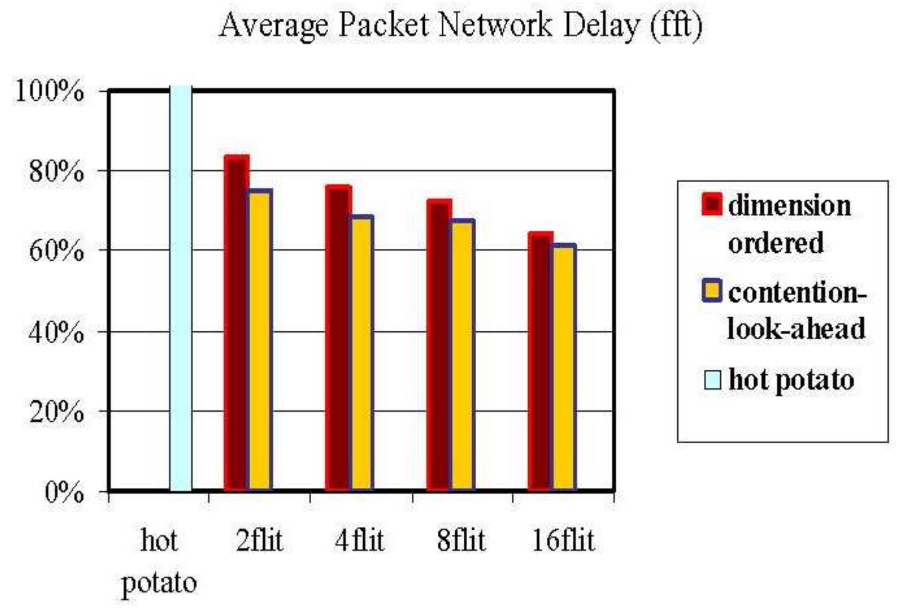

Figure 4.9: Averaged Packet Network Delays (fft) under Different Routing Schemes

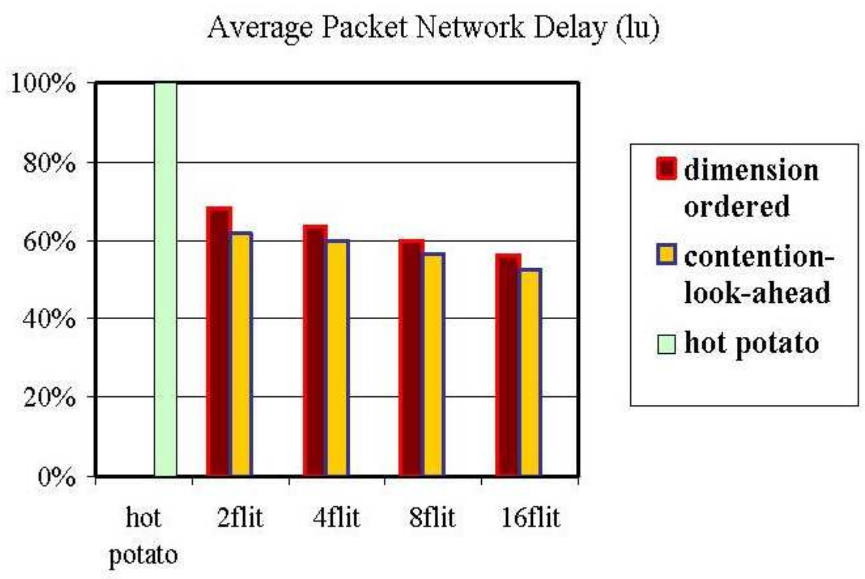

Figure 4.10: Averaged Packet Network Delays (lu) under Different Routing Schemes

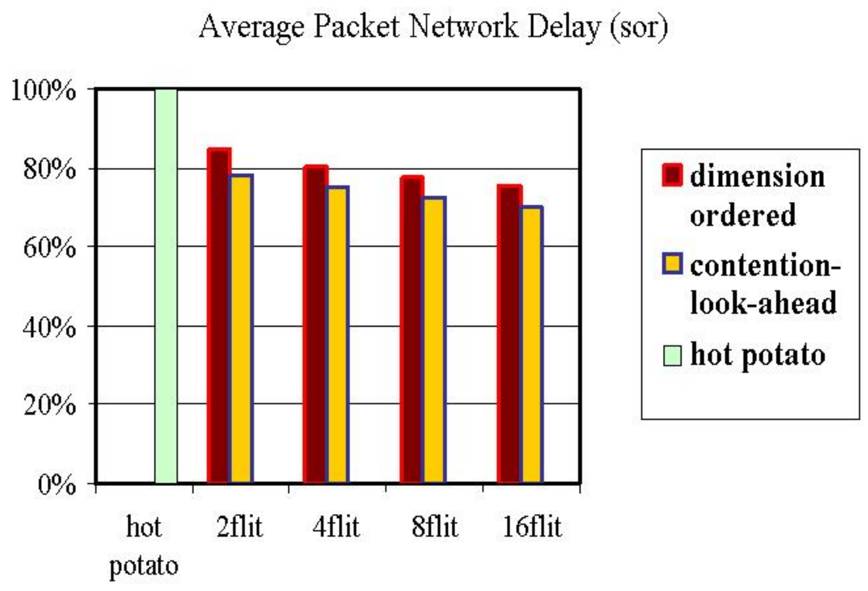

Figure 4.11: Averaged Packet Network Delays (sor) under Different Routing Schemes

still reduce the number of intermediate switches a packet occupies when it is waiting for the next switch. This effect can also be seen from Fig. 4.8, 4.9, 4.10 and 4.11, as packet delays are reduced with larger buffer sizes.

The overall performance (total benchmark execution time) of the multiprocessor system follows the same trend as the network delays, because short network delay helps to accelerate the execution process of node processors. Fig. 4.12, 4.13, 4.14 and 4.15 show the results of the three routing schemes on the benchmarks. Again, results are normalized to the hot potato execution time. Hot potato routing has the longest execution time. Contention-look-ahead routing outperforms dimension-ordered routing in all cases.

4.6.2 Buffer Requirements

In order to obtain deeper insight in the comparison between dimension-ordered routing and contention-look-ahead routing, results from Fig. 4.12, 4.13, 4.14 and 4.15 are re-drawn in Fig. 4.16. The figure shows execution time reduction of each benchmark with various buffer sizes. With the proposed routing scheme, total running time on the multiprocessor platform can be reduced as much as 7.6%. Actually, contention-look-ahead routing shows greater improvement when the buffer sizes are small. As seen from Fig. 4.16, execution time reduction is more significant with smaller buffer size (2-flit, in the figure) than with larger buffer sizes (8-flit). This result is expected because larger buffers "help" the dimension-ordered routing to reduce the network contention and narrow its performance gap between the contention-look-ahead routing.

4.6.3 Network Power Consumption

Power consumption is a major bottleneck for on-chip network designs. Therefore, we further compare the power consumption between the proposed contention-look-ahead routing scheme with the dimension-ordered routing scheme. On-chip network power consumption comes from three contributors, 1) the interconnect wires, 2) the buffers and 3) the switch logic circuits. In this chapter, we adopt the network power

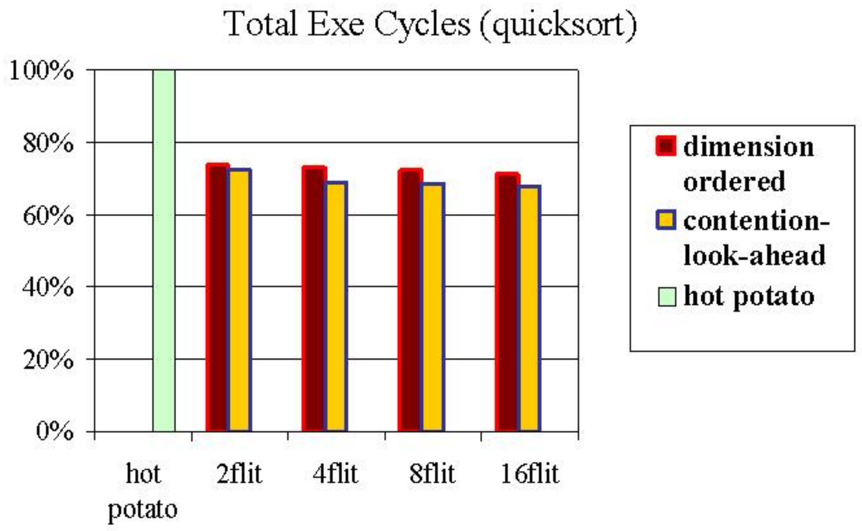

Figure 4.12: Total Execution Time Comparison Between Different Routing Schemes

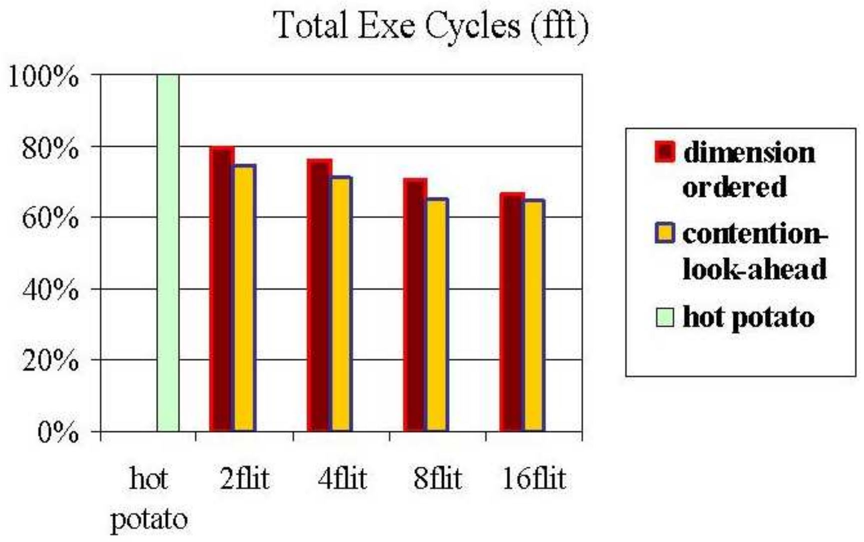

Figure 4.13: Total Execution Time Comparison Between Different Routing Schemes

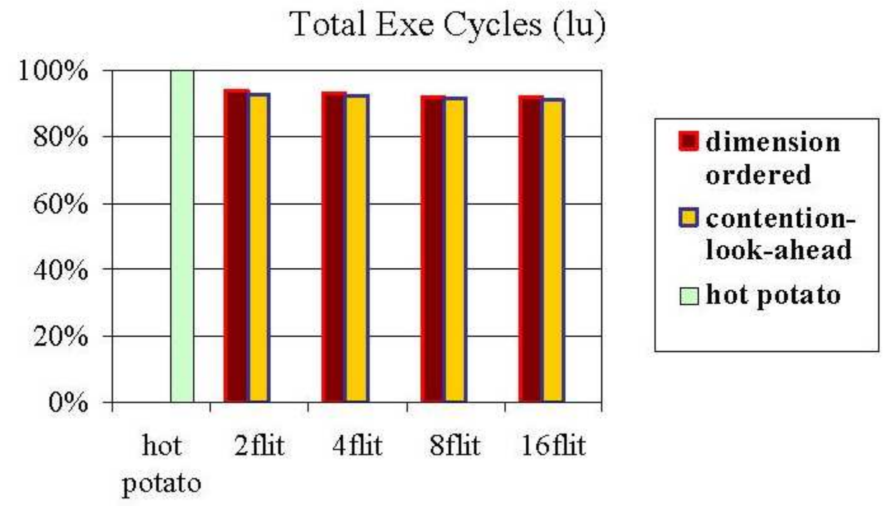

Figure 4.14: Total Execution Time Comparison Between Different Routing Schemes

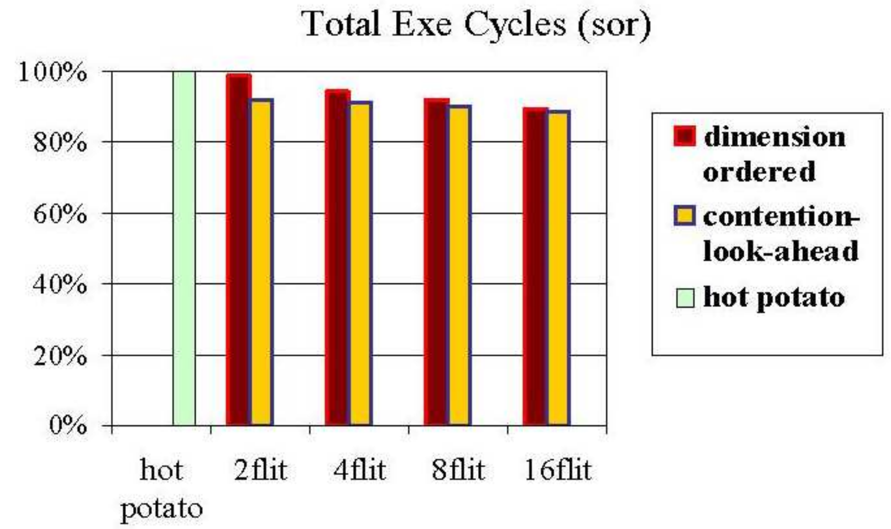

Figure 4.15: Total Execution Time Comparison Between Different Routing Schemes

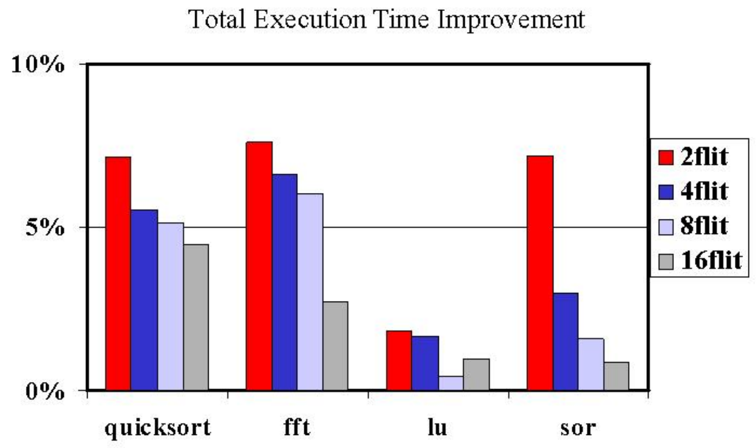

Figure 4.16: Contention-look-ahead Routing Achieves Better Performance with Less Buffers

consumption estimation technique proposed by Chapter 3.

Our proposed routing scheme can "foresee" the contention in the forthcoming stages. Therefore, it can reduce the contention occurrence on the network and shorten the buffer queue length. Consequently, the power consumption on the buffers will be reduced. In the experiments, we traced the buffer activities on the network and compared the buffer power consumption, the results are shown in Fig. 4.17. The figure shows the averaged buffer power reduction of different benchmarks. The reduction is more significant under larger buffer sizes. This is because larger buffers consume more power, and the power consumption is more sensitive with contention occurrence.

The power consumption on the interconnect is determined by the total wire length the packets travel by. In our experiments, the processor nodes are interconnected by the mesh network. The Thompson modeling of the network becomes straightforward, i.e., each *hop* (the segment of interconnect between two adjacent nodes) has the same wire length and the total wire length can be estimated by counting the average number of hops a packet travels from source to destination. In dimension-ordered routing, packets are always routed along the shortest path. In comparison, our proposed routing scheme may choose the misroute when contention occurs. Therefore, the contention-look-ahead routing has larger average hop count per packet than the

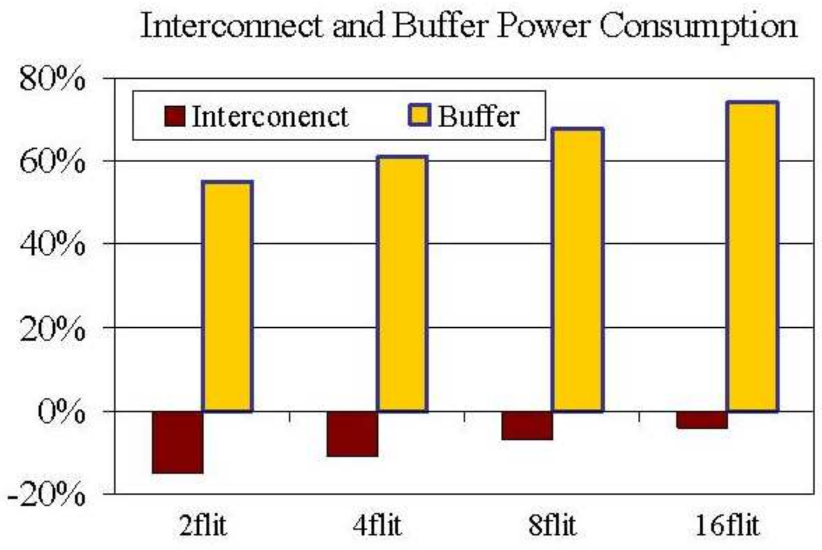

Figure 4.17: Power Consumption Comparison on Interconnect Wires and Buffers

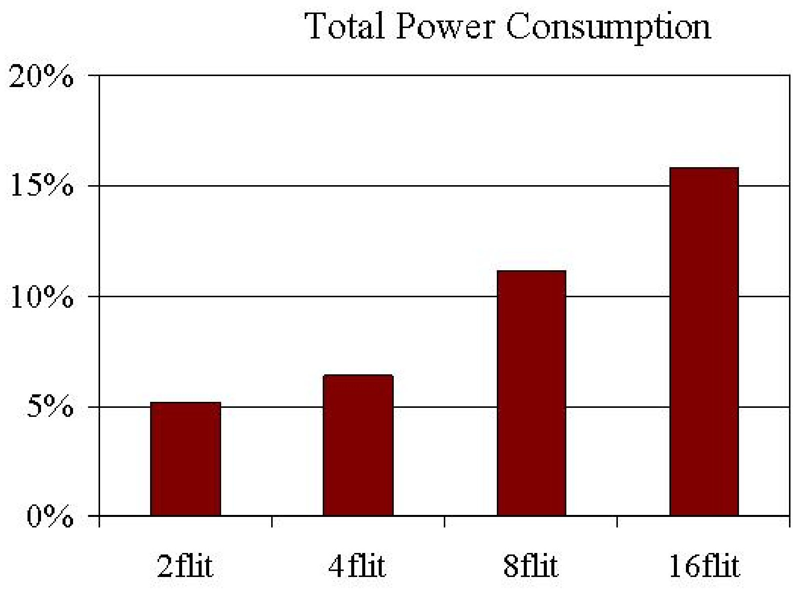

Figure 4.18: Total Power Consumption Reduction

dimension-ordered routing, and consequently consumes more power on the interconnect. This can be seen from Fig. 4.17. The proposed routing scheme consumes more power (shown as negative values) with smaller buffer size. This is because smaller buffer sizes will cause more contention and induce more misroutes.

The contention-look-ahead routing switch needs more logic gates than dimension-ordered routing. From Synopsys Power Compiler simulation, the proposed switch circuit consumes about 4.8% more power than dimension-ordered switch. Combining the power consumption on the interconnects and buffers, the total network power consumption is shown in Fig. 4.18. It shows the total network power reduction compared with dimension-ordered routing, the reduction is more significant with larger buffer sizes (15.2% under 16-flit buffers).

4.7 Summary

On-chip network can benefit from the abundant wiring resources as well as floor-planning locality among PEs. Network routing strategies are limited by on-chip buffers that are expensive to implement and power-hungry during operation. In this chapter, we proposed a contention-look-ahead routing scheme that exploits increased wiring resources, while reducing on-chip buffer requirements. The scheme achieves better performance with significantly less buffer space usage. The network power consumption is also reduced greatly compared to traditional routing algorithms. Although results are reported on a mesh network, the methodology presented is general, and can be extended to different on-chip network architectures.

Chapter 5

On-Chip Communication with Different Packet Sizes

5.1 Introduction

The dataflow traffic on MPSoC interconnect network comes from the processor-processor and processor-memory transactions. Therefore, the performance and power consumption of on-chip communication are not only determined by the physical characteristics of the network (e.g. voltage swing, the wire delay and fan-out load capacitance, etc.), but are also dependent on the interactions between the processor nodes.

In order to analyze how different components of the MPSoC interact with the on-chip network, in this chapter, we use the homogeneous shared-memory MPSoC as a case study. The processors are interconnected by the on-chip mesh network, each processor is identical and has its own memory hierarchy. The memories are globally addressed and accessible.

Although homogeneous shared-memory MPSoC is only a specific example of MPSoC architectures, it is a good platform to analyze many characteristics of MPSoC network traffic. Generally, the on-chip traffic is coming from the following sources:

1. **Cache and memory transactions.** Every cache miss needs to fetch data from the shared memories, and consequently creates traffic on the interconnect.

2. **Cache coherence operations.** In MPSoC, one data may have multiple copies in the caches of different node processors. When the data in memory is updated, its cache copies also need to be updated or invalidated. This synchronization operation will create traffic on the interconnect as well.

3. **Packet segmentation overheads.** When dataflows are segmented into packets, traffic on the interconnect will carry additional overhead. The overhead is dependent on the packet size and header/tail size.

4. **Contentions between packets.** When there is contention between packets on the interconnect, the packets need to be re-routed to another datapath or buffered temporarily. This effect will again change the traffic pattern on the interconnect.

The above factors are not independent. Instead, the performance and power tradeoff is determined by the interactions of all factors dynamically, and the variation of one factor will impact other factors. For example, the changes of packet size will affect the cache block size that can be updated during each memory access, and consequently change the cache miss rate.

While the MPSoC performance issues have been addressed by many researchers in the parallel computing field [11], the power consumption for on-chip network communication has not been quantitatively analyzed. Previous researches either use statistical traffic model, or calculate the power consumption in analytical methods [49][37][33]. Those researches did not address the packetization impact on the network power consumption, and they did not specify how on-chip network designers need to trade-off different options in CPU, cache and memory designs at the architectural level.

In this chapter, we introduce a MPSoC interconnect communication energy model and apply this model on RSIM, a multi-processor simulator [28]. We will analyze quantitatively the relationship between different packetization factors, and their impact on the power consumption as well as system performance. Furthermore, based on the analysis, we will show the trade-offs between MPSoC performance and its interconnect power consumption.

5.2 On-chip Network Traffic

Before we start to discuss the characteristics of MPSoC interconnect networks, we need to first study the traffic on the network; in particular, we should analyze the composition of the packetized dataflows that are exchanged between MPSoC nodes.

Packets transported on the MPSoC network consist of three parts. The header contains the destination address, the source address, and the requested operation type (READ, WRITE, INVALIDATE, etc). The payload contains the transported data. The tail contains the error checking or correction code.

5.2.1 Sources of Packets

Packets traveling on the network come from different sources, and they can be categorized into the following types:

1. **Memory access request packet.** The packet is induced by L2 cache miss that requests data fetch from memories. The header of these packets contains the destination address of the target memory (node ID and memory address) as well as the type of memory operation requested (memory READ, for example). Because there is no data being transported, the payload is empty.

2. **Cache coherence synchronization packet.** The packet is induced by the cache coherence operation from the memory. This type of packet comes from the updated memory, and it is sent to all caches that have a copy of the updated data. The packet header contains the memory tag and block address of the data. If the synchronization uses the “update” method, the packet contains updated data as payload. If the synchronization uses the “invalidate” method, the packet header contains the operation type (INVALIDATE, in this case), and the payload is empty.

3. **Data fetch packet.** This is the reply packet from memory, containing the requested data. The packet header contains the target address (the node ID of

the cache requesting for the data). The data is contained in the packet payload.

4. **Data update packet**. This packet contains the data that will be written back to the memory. It comes from L2 cache that requests the memory write operation. The header of the packet contains the destination memory address, and the payload contains the data.

5. **IO and interrupt packet.** This packet is used by IO operations or interrupt operations. The header contains the destination address or node ID. If data exchange is involved, the payload contains the data.

5.2.2 Data Segmentation and Packet Size

From the analysis in Section 5.2.1, we can see that most packets travel between memories and caches, except those packets involved in I/O and interrupt operations. Although packets of different types originate from different sources, the length of the packets is determined by the size of the payload. In reality, there are two differently sized packets on the MPSoC network, *short packet* and *long packet*, as described below.

Short packets are the packets with no payloads, such as the memory access request packets and cache coherence packets (invalidate approach). These packets consist only header and tail. The request and control information can be encoded in the header section.

Long packets are the packets with payloads, such as the data fetch packets, the data update packets and the cache coherence packets used in update approach. These packets travel between caches and memories. The data contained in the payload are either from cache block, or they are sent back to the node cache to update the cache block. Normally, the payload size equals the cache block size, as shown in Fig. 5.1.

Packets with payload size different than the cache block size will increase cache miss penalty. The reasons are two. 1) If each cache block is segmented into different

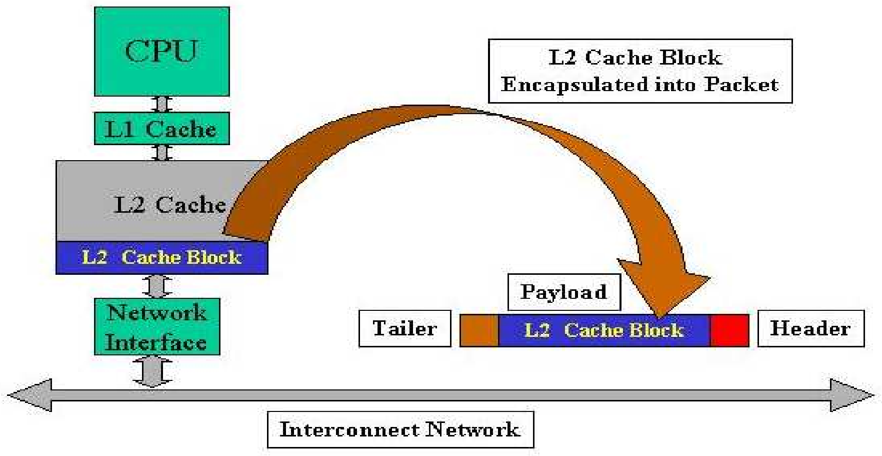

Figure 5.1: Packet Size and Cache Block Size

packets, it is not guaranteed that all packets will arrive at the same time, and consequently the cache block cannot be updated at the same time. 2) If several cache blocks are to be packed into one packet payload, the packet needs to hold its transmission until all the cache blocks are updated. This will again increase the cache miss delay penalty.

In our analysis, we assume all the long packets contain the payload of one cache block size. Therefore, the length of the long packets will determine the cache block size of each node processor.

5.3 MPSoC Power Consumption

The MPSoC power consumption originates from three sources: the node power consumption, the shared memory power consumption and the interconnect network power consumption.

5.3.1 Node power consumption

Node power consumption comes from the operations inside each node processor, these operations include:

1. CPU and FPU operations. Instructions like *ADD, MOV, SUB etc* consume power because these operations cause the logic gates to toggle on the datapath of processor.

2. L1 cache access. L1 cache is built with fast SRAMs. When data is loaded or stored in the L1 cache, it consumes power.

3. L2 cache access. L2 cache is built with slower but larger SRAMs. Whenever there is a read miss in L1 cache, or when there is write back from L1 cache, L2 cache is accessed, and consequently consumes power.

5.3.2 Shared memory power consumption

Data miss in L2 cache requires data to be fetched from memory. Data write back from L2 cache also needs to update the memory. Both operations will dissipate power when accessing the memories.

5.3.3 Interconnect network power consumption

Operations like cache miss, data fetch, memory updates and cache synchronization all need to send packets on the interconnect network. When packets are transported on the network, energy is dissipated on the interconnect wires as well as the logic gates inside each switch. Both wires and logic gates need to be counted when we estimate the network power consumption.

Among the above three sources, the node power consumption and memory power consumption have been studied by many researches. In the following sections, we will only focus our analysis on the power consumption of interconnect networks. Later in this chapter, when we compare the network power consumption with the total MPSoC power consumption, we will reference the results from other researches for node processor and memory power estimation.

5.4 Network Energy Modeling

5.4.1 Bit Energy of Packet

When a packet travels on the interconnect network, both the wires and logic gates on the datapath toggle as the bit-stream flips its polarity. In this chapter, we use the approach presented in Chapter 3 to estimate the energy consumption for the packets traveling on the network.

We adopt the concept of bit energy E_{bit} to estimate the energy consumed for each bit when the bit flips its polarity from previous bit in the bit stream. We further decompose the bit energy E_{bit} into bit energy consumed on the interconnect wires $E_{W_{bit}}$ and the bit energy consumed on the logic gates inside the node switch $E_{S_{bit}}$. The local memory on each node also serves as the buffer. The energy consumed on the buffer can be estimated from the memory access energy, which will be discussed in a later part of the chapter.

The bit energy consumed on the interconnect wire can be estimated from the total load capacitance on the interconnect. The total load capacitance can be calculated from Thompson model, as described in Chapter 3.

The bit energy consumed on the switch logic gates can be estimated from Synopsys Power Compiler simulation. Without loss of generality, we use random bit-stream as the packet payload content. Details of the estimation can also be found in [52].

5.4.2 Packets and Hops

When the source node and destination node are not placed adjacent to each other on the network, a packet needs to travel several intermediate nodes (or hops) until reaching the destination, as shown in Fig. 5.2.

In the mesh or torus network, there are several different alternate datapaths between source and destination, as shown in Fig. 5.2. When contention occurs between packets, the packets may be re-routed to different datapaths. Therefore, packet datapath will vary dynamically according to the traffic condition. Packets with the same source and destination may not travel through the same number of hops, and they

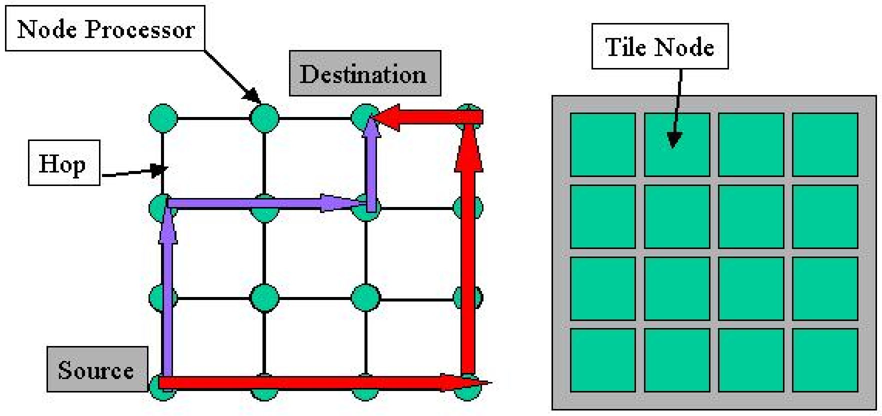

Figure 5.2: Hops and Alternate Routes of Packets

may not necessarily travel on the datapath with the minimum number of hops.

The number of hops a packet travels greatly affects the total energy consumption needed to transport the packet from source to destination. For every hop a packet travels, the interconnect wires between the nodes will be charged and discharged as the bit-stream flows by, and the logic gates inside the node switch will toggle.

We assume a tiled floorplan implementation for MPSoC, similar to those proposed by [15] and [31], as shown in Fig. 5.2. Each node processor is placed inside a tile, and the mesh network is routed in a regular topology. Without loss of generality, we can assume all the hops in mesh network have the same interconnect length. Therefore, if we pre-calculate the energy consumed by one packet on one hop, E_{hop}, by counting the number of hops a packet travels, we can estimate the total energy consumed by that packet.

We use the hop histogram to show the total energy consumption by the packet traffic. In Fig. 5.3 below, histograms of the packets traveling on an 8-processor SoC are shown. The 8 processors are connect by a 2-dimensional mesh interconnect network. The histograms are extracted from the trace file of a *quicksort* benchmark. The histogram has n bins with *1, 2, .., n* hops, the bar on each bin shows the number of packets in each bin. We count long packets and short packets separately in the histograms.

Without loss of generality, we can assume packets of the same length will consume

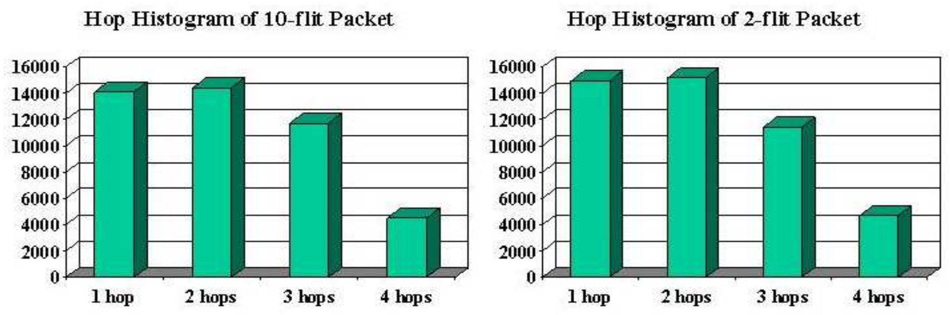

Figure 5.3: Hop Histogram of Long and Short Packets

the same energy per hop. Using the hop histogram of the packets, we can calculate the total network energy consumption with the following equation (Eq. 5.1).

$$E_{packet} = \sum_{h=1}^{maxhops} h \times N(h)_{packet} \times L_{long} \times E_{flit} \tag{5.1}$$

$$+ \sum_{h=1}^{maxhops} h \times N(h)_{packet} \times L_{short} \times E_{flit} \tag{5.2}$$

where $N(h)_{packet}$ is the number of packets with h number of hops in the histogram. L_{long} and L_{short} are the lengths of long and short packets respectively, in the unit of flit. E_{flit} is the energy consumption for one flit on each hop. Because the packets are actually segmented into flits when they are transported on the network, we only need to calculate the energy consumption for one flit, E_{flit}. The energy of one packet per hop E_{hop} can be calculated by multiplying the number of flits the packet contains.

5.5 Experiments

5.5.1 Platform

We again use RSIM as our shared memory MPSoC simulation platform in this analysis. The architecture and network is very similar to that used in Chapter 4. Here we just describe more details on the memory hierarchy.

Each node processor contains two levels of cache hierarchy. L1 cache is 16K bytes, and L2 cache is 64K bytes. Both L1 and L2 cache use write-through methods for memory updates. We use the *invalidate* approach for cache coherence synchronization. Wormhole routing is used, and the flit size is 8 bytes.

5.5.2 Energy Model

As discussed in Section 5.4, to calculate the power consumption on the network, we need to calculate the value of E_{flit}, which is the energy consumed by one flit traveling on one hop. We assume each tile of node processor is $2mm \times 2mm$ in dimension, and they are placed regularly on the floorplan, as shown in Fig. 5.2. We assume $0.13\mu m$ technology is used, and the wire load capacitance is 0.50fF per micron. Under these assumption, the energy consumed by one flit on one hop interconnect is 0.174nJ.

The energy consumed on the switch logic gates of one hop is calculated from Synopsys Power Compiler. We calculate the bit energy on the logic gates in a way similar to that used in [52]. We use $0.13\mu m$ standard cell library, and the energy consumed by one flit on one hop switch is 0.096nJ. Based on these calculation, the flit energy per hop $E_{flit} = 0.27nJ$.

5.5.3 Experiments and Benchmarks

We tested five applications on our RSIM MPSoC simulation platform, they are *sor*, *water*, *quicksort*, *lu* and *mp3d*. These applications are also ported from the Stanford SPLASH project. To analyze how different packetization schemes will affect the performance and power, we change the dataflow with different packet sizes. The packet payload sizes are varied from 16Byte, 32Byte, 64Byte, 128Byte to 256Byte. Because the short packets are always 2-flit in length, therefore, the change of packet size is applied to long packets only. The results are discussed quantitatively in the following sections.

5.6 Packetization and MPSoC Performance

As we mentioned in Section 1, MPSoC performance is determined by many factors. Different packetization schemes affect these factors differently, and consequently, result in different performance metrics.

5.6.1 Cache Miss Rate

Changing the packet payload size (for long packets) will change the L2 cache block size that can be updated in one memory fetch. If we choose larger payload size, more cache contents can be updated. Therefore, the cache miss rate will decrease. This effect can be observed from Fig. 5.4 and 5.5. As the packet payload size increases, both the L1 cache (Fig. 5.4 and L2 cache (Fig. 5.5 miss rates decrease. Decreased cache miss rate will reduce the number of packets needed for memory access.

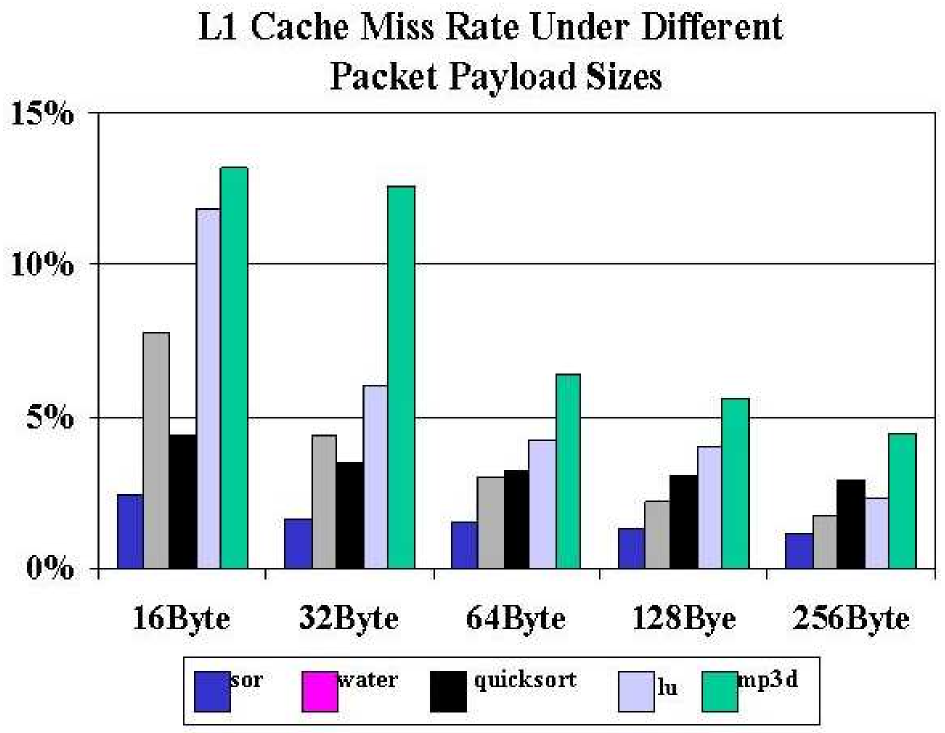

Figure 5.4: MPSoC L1 Cache Miss Rate under Different Packetization Schemes

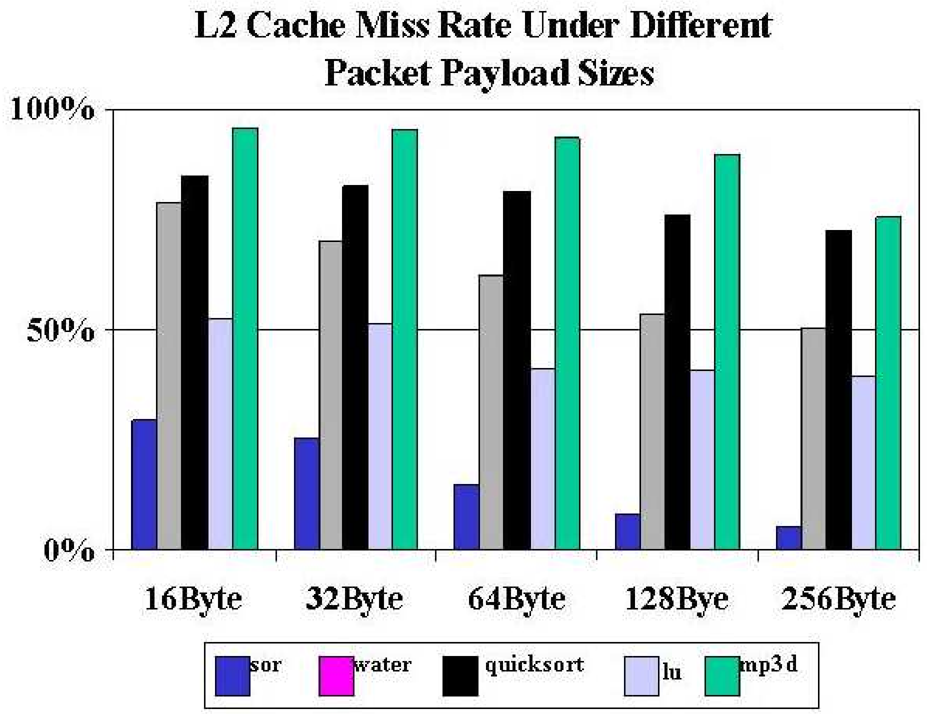

Figure 5.5: MPSoC L2 Cache Miss Rate under Different Packetization Schemes

5.6.2 Cache Miss Penalty

Whenever there is a L2 cache miss, the missed cache block needs to be fetched from the memories. The latency associated with this fetch operation is called a miss penalty. When we estimate the cache miss penalty, we need to count all the delays occurred within the fetch operation. These delays include: 1) packetization delay, 2) interconnect delay, 3) store and forward delay on each hop for one flit, 4) arbitration delay, 5) memory access delay and 6) contention delay. Among these six factors, 2), 3) and 4) will not change significantly for packets with different sizes, because we use wormhole routing. However, delays on 1) and 5) will become longer because larger packets need longer time for packetization and memory access. Longer packets will actually cause more contention delay. This is because when wormhole routing is used, longer packet will hold more intermediate nodes during its transmission. Other packets have to wait in the buffer, or choose alternative datapaths, which are not necessarily the short routes. Combining all these factors, the overall cache penalty will increase as the packet payload size increases, as shown from Fig. 5.6.

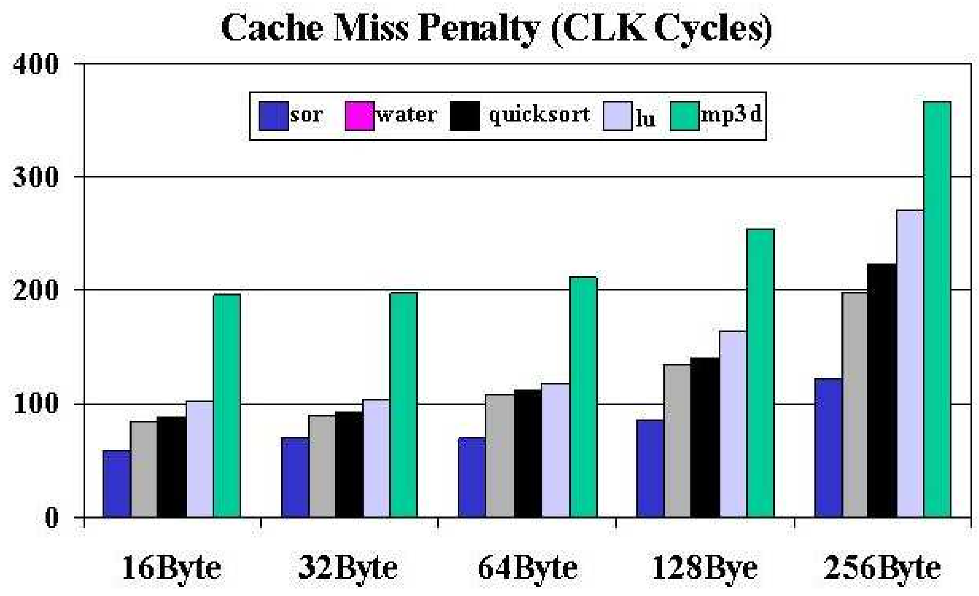

Figure 5.6: Cache Miss Penalty under Different Packetization Schemes

5.6.3 Overall Performance

From the above analysis, we know that although larger payload size helps to decrease the cache miss rate, it will increase the cache miss latency. Combining these two factors, there exists an optimal payload size that can achieve the minimum execution time, as seen from Fig. 5.7. In order to illustrate the variation of performance, we normalized the figure to the minimum execution time of each benchmark. In our experiments, all the five benchmarks achieve the best performance with 64 bytes of payload size.

5.7 Packetization and Power Consumption

Eq. 5.1 in Section 5.4 shows that the power consumption of packetized dataflow on MPSoC network is determined by the following three factors: 1) the number of

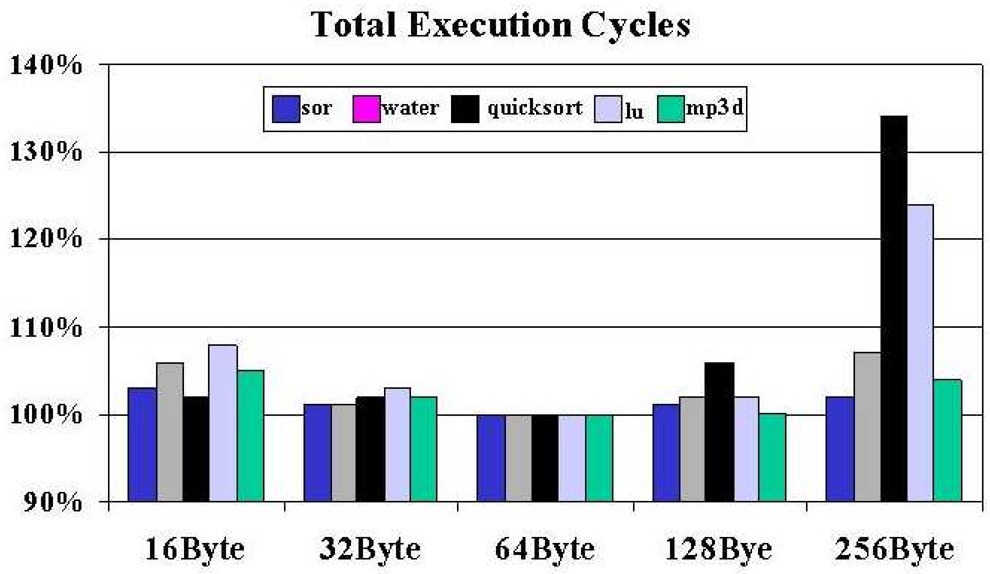

Figure 5.7: MPSoC Performance under Different Packetization Schemes

packets on network, 2) the energy consumed by each packet on one hop, and 3) the number of hops each packet travels. Different packetization schemes will have different impact on these factors, and consequently affect the network power consumption. We summarize these effects and list them below.

1. Packets with larger payload size will decrease the cache miss rate and consequently decrease the number of packets on the network. This effect can be seen from Fig. 5.8. It shows the average number of packets on the network (traffic density) at one clock cycle. As the packet size increases, the number of packets decreases accordingly. Actually, with the same packet size, the traffic density of different benchmarks is consistent with the miss penalty. By comparing Fig. 5.8 with Fig. 5.6, we see that if the packet length stays the same, higher traffic density causes longer miss latency.

2. Larger packet size will increase the energy consumed per packet, because there are more bits in the payload.

3. As discussed in Section 5.6, larger packets will occupy the intermediate node

switches for a longer time, and cause other packets to be re-routed to non-shortest datapaths. This leads to more contention that will increase the total number of hops needed for packets traveling from source to destination. This effect is shown in Fig.5.9. It shows the average number of hops a packet travels between source and destination. As packet size (payload size) increases, more hops are needed to transport the packets.

Actually, increasing the cache block size will not decrease the cache miss rate proportionally [38]. Therefore, the decrease of packet count cannot compensate the increase of energy consumed per packet caused by the increase of packet length. Larger packet size also increases the hop counts on the datapath. Fig. 5.10 shows the combined effects of these factors under different packet sizes. The values are normalized to the measurement of 16Byte. As packet size increases, energy consumption on the interconnect network will increase.

Although the increase of packet size will increase the energy dissipated on the network, it will decrease the energy consumption on cache and memory. Because larger packet sizes will decrease the cache miss rate, both cache energy consumption and memory energy consumption will be reduced. This relationship can be seen from Fig. 5.11 and 5.12. It shows the energy consumption on cache and memory under different packet sizes respectively. The access energy of each cache and memory instruction is estimated based on the work from [19] and [42]. The energy in the figure is normalized to the value of 256Byte, which achieves the minimum energy consumption.

The total energy dissipated on MPSoC comes from non-cache instructions (instructions that do not involve cache access) of each node processors, caches, shared memories as well as the interconnect network. In order to see the packetization impact on the total system energy consumption, we put all MPSoC energy contributors together and see how the energy changes under different packet sizes. The results are shown in Fig. 5.13. From this figure, we can see the overall MPSoC energy will decrease as packets size increases. However, when the packets are too large, as in the case of 256Byte in the figure, the total MPSoC energy will increase. This is because when the packet is too large, the increase of interconnect network energy will

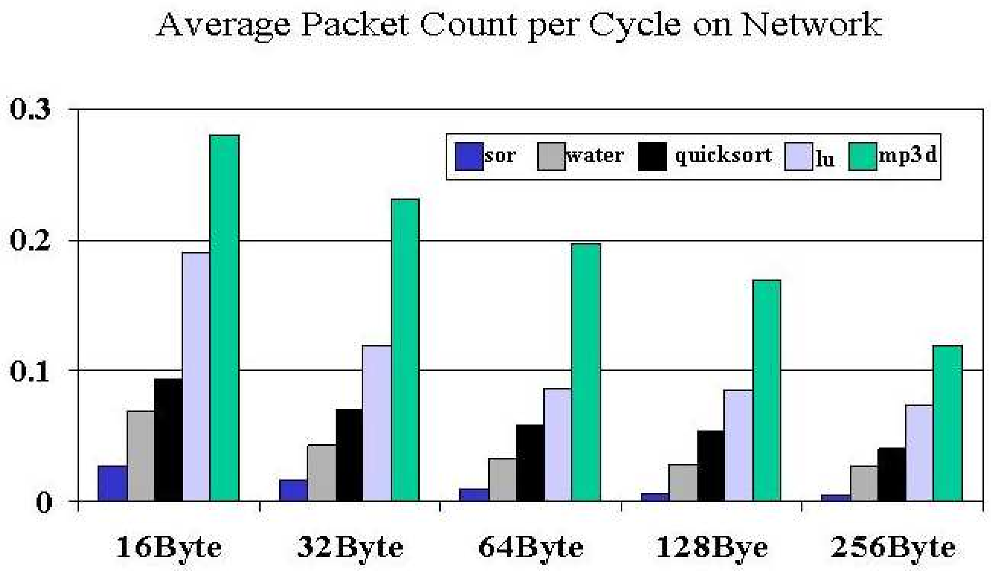

Figure 5.8: Packet Count Changes as Packet Payload Size Increases

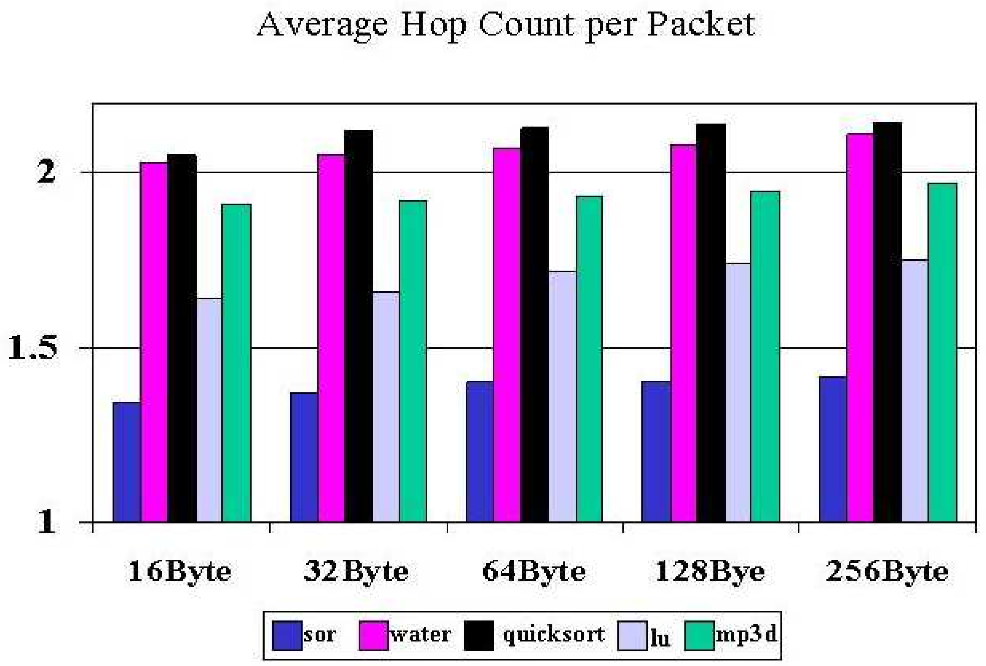

Figure 5.9: Contention Occurrence Changes as Packet Payload Size Increases

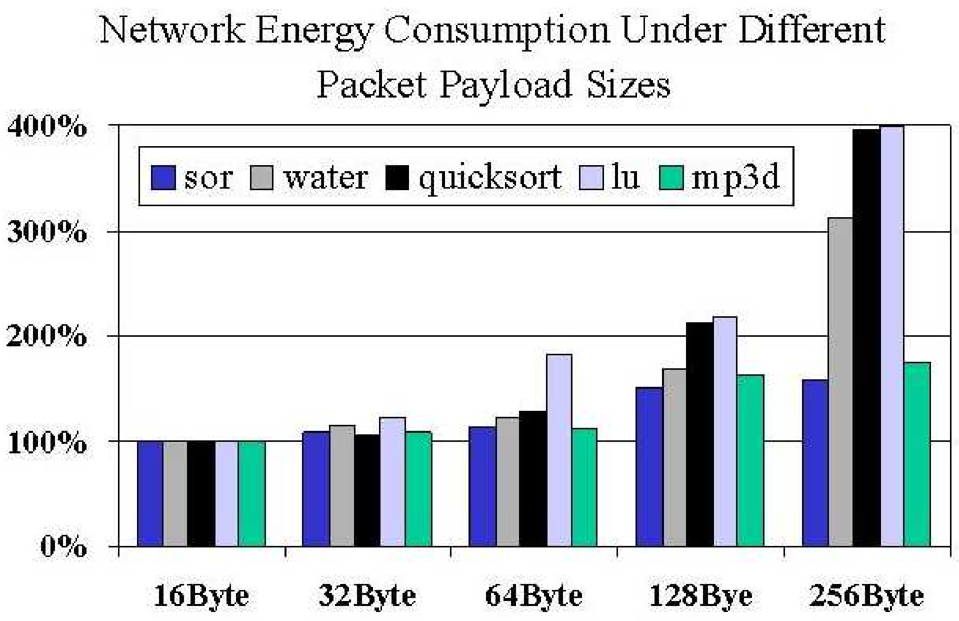

Figure 5.10: Network Energy Consumption under Different Packet Payload Sizes

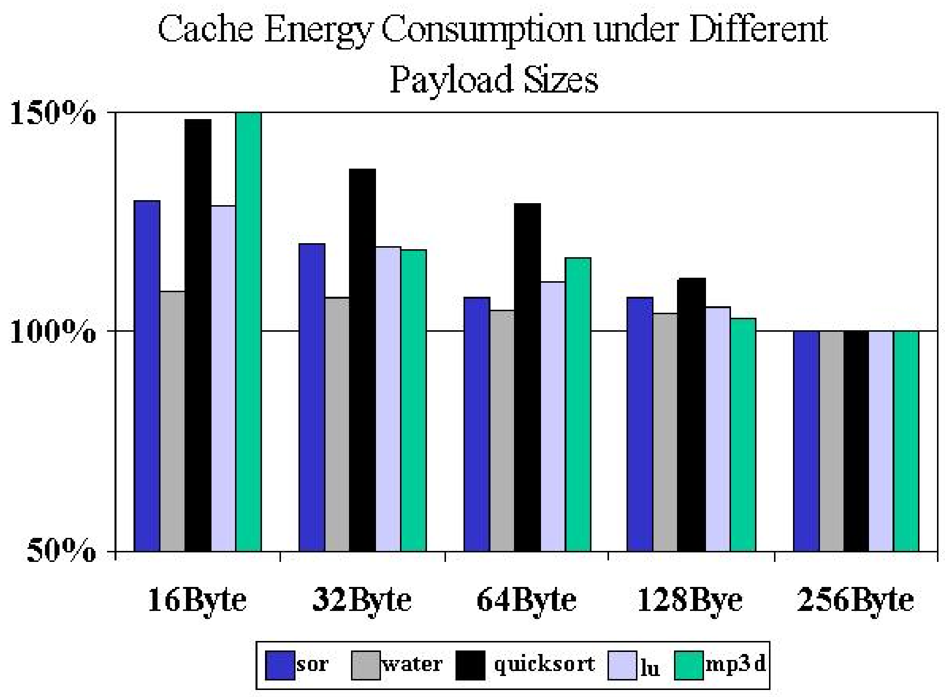

Figure 5.11: Cache Energy Decrease as Packet Payload Size Increases

outgrow the decrease of energy on cache, memory and non-cache instructions. In our simulation, the non-cache instruction energy consumption is estimated based on the techniques presented in [7], and it does not change significantly under different packet sizes.

5.8 Packetization Impact Analysis

Although the specific measurement values in the experiments are technology and platform dependent, we believe the analysis will hold for different MPSoC implementations. We summarize our analysis qualitatively as follows (Fig. 5.14).

Large packet size decreases the cache miss rates of MPSoC but increases the miss penalty. The increase of miss penalty is caused by the increase of packetization delay, memory access delay, as well as contention delay on the network. As shown qualitatively in Fig. 5.14a, the cache miss rate saturates with the increase of packet size. Nevertheless, the miss penalty increases faster than linearly. Therefore, there exists an optimal packet size to achieve best performance.

The energy spent on the interconnect network increases as the packet size increases. Three factors play roles in this case (Fig. 5.14b). 1) Longer packets, i.e. larger cache lines, reduce the cache miss rate, hence reduce the packet count. Nevertheless, the packet count does not fall linearly with the increase of packet size. 2) The energy consumption per *packet* × *hop* increases in a linear fashion with the increase of packet length. If we ignore the overhead of packet header and tail, this increase is proportional to packet size. 3) The average number of hops per packet on the network also increases with the packet length. The combined effect causes the network energy to increase as the packet size increases.

The total MPSoC system energy is dominated by the sum of three factors as the packet size increases (Fig. 5.14c). 1) Cache energy will decrease. 2) Memory energy will decrease as well. 3) Network energy will increase over-linearly. In our benchmarks, the non-cache instruction energy does not change significantly. The overall trend depends on the breakdown among the three factors. Our experiments show that there exists a packet size that minimizes the overall energy consumption.

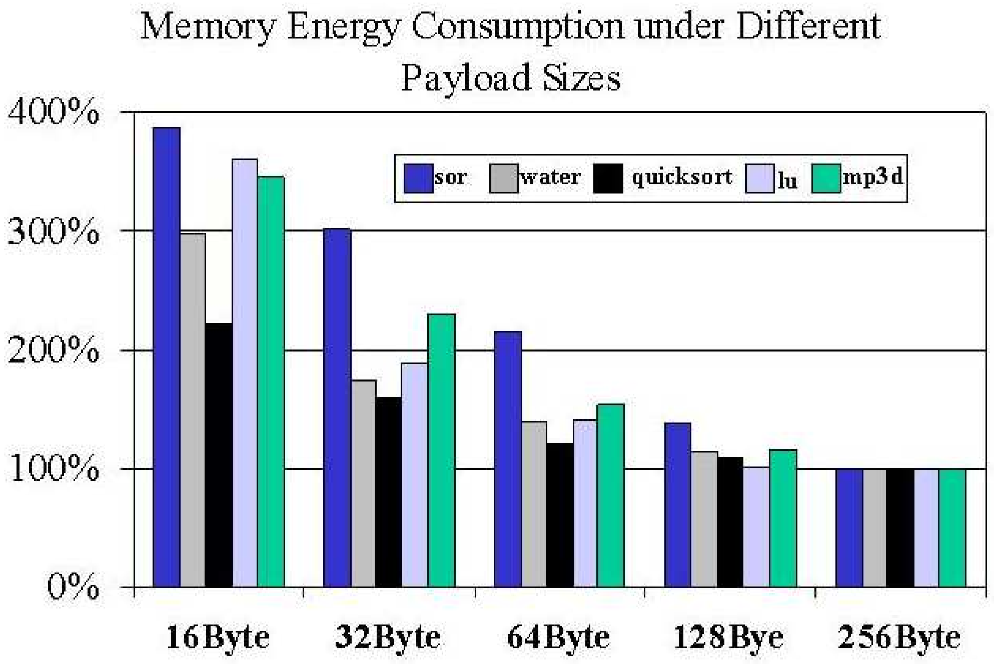

Figure 5.12: Memory Energy Decrease as Packet Payload Size Increases

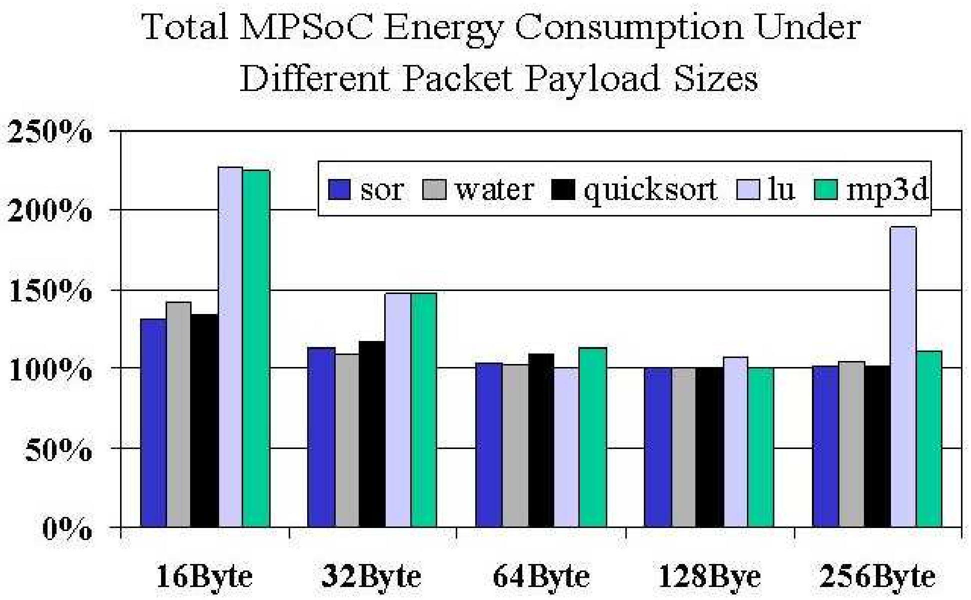

Figure 5.13: Total MPSoC Energy Consumption under Different Packet Payload Sizes

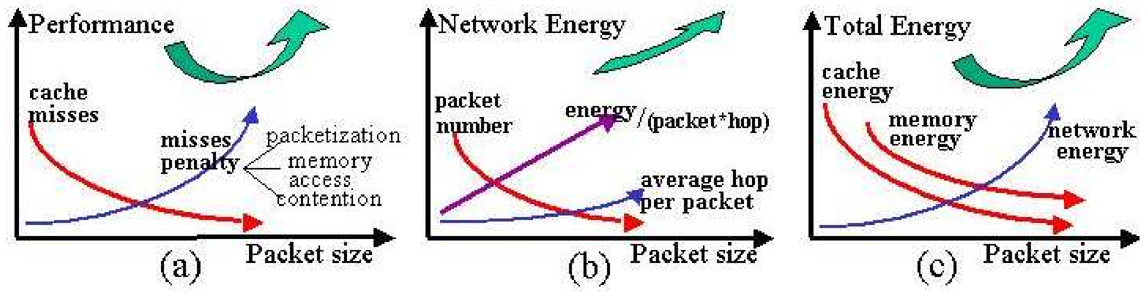

Figure 5.14: Qualitative Analysis of Packet Size Impact

Moreover, if the network energy contributes a major part of the total system energy consumption, which is expected to happen as VLSI technology moves to nanometer domain, the MPSoC energy will eventually increase with the packet size.

5.9 Summary

In this chapter, we analyzed the effect of packet size variation on MPSoC performance and energy consumption. We further show that in the MPSoC platform, the behavior of node processors is closely coupled with the activity on the network. The analysis presented will help designers to select the appropriate architectures and communication schemes in their system level design.

Chapter 6

Physical Planning of On-Chip Networks

6.1 Introduction

Designing the on-chip network will become a major task for future MPSoCs. A large fraction of the timing delay is spent on the signal propagation on the interconnect, and a significant amount of energy is also dissipated charging and discharging the load capacitance on the wires [25]. Therefore, an optimized interconnect network floorplan will be of great importance to MPSoC performance and energy consumption.

With the ever-increasing complexity of MPSoC integration, manual floorplanning of the processing elements and switches will become even more time consuming and inefficient. Automated methods are needed for large-scale MPSoC designs. Unlike traditional floorplanning that deals with the circuit macro block placement and wire routing [39], MPSoC floorplanning needs to solve the problems from a different perspective, as illustrated in Fig. 6.1. Namely:

1. *Folding and planarization* – MPSoC network topologies are multi-dimensional. MPSoC planar layout requires that PE blocks are tiled and abutted on the floorplan in a two-dimensional tile array [15]. The planarization process is also constrained by the pre-defined aspect ratio and row/column numbers of the tile

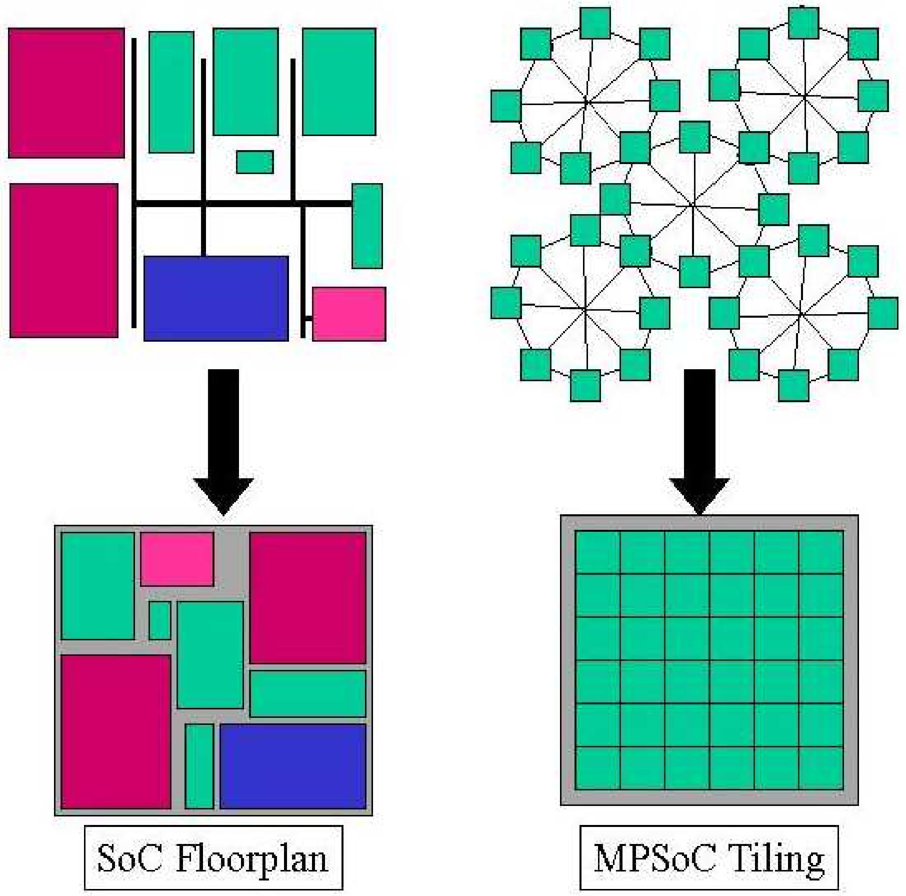

Figure 6.1: MPSoC Tiling Is Different From Traditional Floorplan

array.

2. *Regularity and hierarchy* – MPSoC network topologies are often regular and hierarchical. The planarization of the network is not only a simple packing process: it has to preserve the regularity and hierarchy on the floorplan.

3. *Critical path and total wirelength* – Interconnect delays and power consumption are the two critical issues in MPSoC network design. On one hand, inter-node communication latencies are dominated by the wire propagation delays. Therefore, the wirelength of the timing-critical links needs to be minimized. On the other hand, interconnect wires are the main contributors of the total system power consumption. Reducing the total wirelength helps reducing the power

dissipated on the interconnect.

Prior network graph planarization approaches either targeted only some specific topologies, or they were not flexible enough to adapt to many of the floorplan constraints imposed by the silicon implementation [16] [21]. Therefore, those approaches are not suitable for an automated design flow.

In this chapter, we propose a floorplanning method and a tool called *REGULAY* that can automatically place regularly-configured MPSoC node processors as well as switch fabrics onto a user-defined tile floorplan. Given the MPSoC network topology and the physical dimension of the network nodes as inputs, along with the floorplan specification (locations of the I/O tiles, number of rows and columns of the tiles), *REGULAY* can create a floorplan that best satisfies different design constraints.

6.2 MPSoC Network Floorplan

Although quite different in their topologies, many MPSoC networks have some important aspects in common: *regularity and hierarchy.* Regular and hierarchical topologies help to distribute the network traffic and balance the workload of node processors. Therefore, preserving the regularity and hierarchy formations in the silicon floorplan is critical in MPSoC implementation.

Furthermore, on-chip interconnect delays and power consumption add additional requirements in MPSoC floorplan design. To reduce wiring delays, MPSoC floorplans need to limit the wirelength of the critical links (links that are timing sensitive). To reduce the interconnect energy dissipation, the total network wirelength needs to be minimized.

6.3 Problem Formulation

In an MPSoC floorplan, each node processor or node switch is placed as a dedicated hard block tile. For example, in direct networks, as in the case of the Octagon network design, the node processors can be tiled in a two-dimensional array, e.g., a 6×6 array in Fig. 6.2. In indirect networks, as in the case of the Butterfly network, the tiling of the switch fabrics will be constrained by the locations of the node processors, as shown in Fig. 6.2.

Formally, we are given a source network S connecting a set of modules M, $M = \{m_i, i = 1, 2, 3, ..., p\}$, and a target two-dimensional tile array T with $col \times row$ tiles. Since modules cannot overlap, we assume $p \leq col \times row$. Each net in N connects two (or more) modules in M, and has a weighting factor. For example, net $n_{ij,...,k} \in N$ connects modules in $m_i, m_j, ..., m_k$ and has weight $w_{ij,...,k}$.

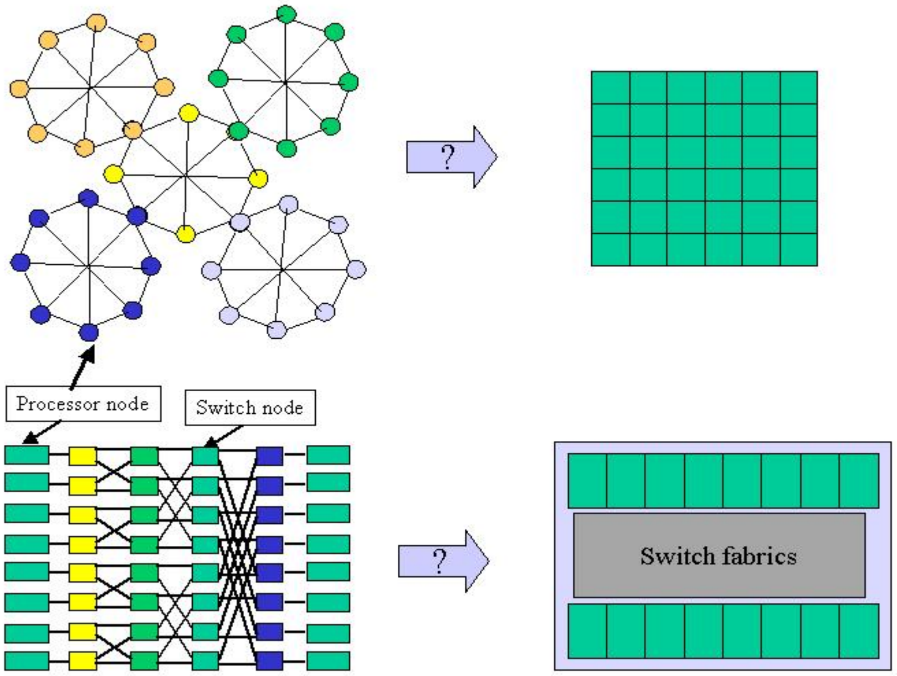

Figure 6.2: Constraints of Floorplan Tiling

Different network topologies and application requirements set different constraints on MPSoC floorplanning problems. To be more specific, we summarize the constraints that are relevant for MPSoC floorplanning:

1) *Regularity constraints* – As shown in Section 2.2, MPSoC placement should preserve the regularity of the original network topology.

2) *Hierarchy constraints* – MPSoC networks may have hierarchical topologies (clusters), e.g., a cascaded Octagon network consists of multiple local rings. The placement should also preserve this hierarchical clusters.

3) *I/O constraints* – An MPSoC is implemented on a single chip. Some node processors (or node switches, in the case of switch fabrics) serve as I/O nodes; therefore, they need to be placed at the peripherals of the floorplan. An MPSoC floorplan needs to accommodate those nodes at their proper locations.

4) *Aspect-ratio constraints* – Chip die size is limited by the silicon area and aspect ratio. Therefore, node processor blocks and node switch blocks need to be packed into a two-dimensional array with predefined numbers of rows and columns.

5) *Critical-path constraints* – The links between some node processors may be the critical paths, e.g. the center ring in the cascaded Octagon network. Therefore, the nodes connected by the critical paths need to be placed closer to each other.

6) *Total net-length constraints* – Reducing the total net length will achieve shorter interconnect delays with lower power consumption.

The floorplanning problem is to determine a mapping from S to T, such that the constraints are met and the overall wiring length is minimal. Such a problem is computationally intractable, and has been the object of extensive investigation in the ASIC domain. We propose a two-step heuristic approach that takes into account the special properties of MPSoC topologies.

The proposed approach consists of two steps: 1) regularity extraction and determination of tentative locations, and 2) legalization. The first step generates the relative locations of the modules based on the regularity and hierarchical information extracted from the network topology. If some modules have pre-fixed locations in T, these locations are used as placement constraints. The total weighted net length is used as objective function. The second step will pack the modules onto the floorplan constrained by the I/O locations and aspect ratio.

6.4 Regularity Extraction

We represent the network topology as a connectivity matrix, where each element off-diagonal of the matrix corresponds to an edge of the topology graph. We use the total square wirelength among the nodes as the objective function. The minimization of the objective function can be calculated through a series of matrix operations.

The matrix representation preserves the topological regularity information, i.e., if the nodes in the original topology are symmetrical, the corresponding elements in the matrix are symmetrical as well. Furthermore, all subsequent matrix operations (e.g., transposition, vector multiplication, etc.) will preserve regularity. Therefore, by minimizing the total square wirelength with this model, the regularity information is preserved in the optimization process.

6.4.1 Forming the Objective Function

We generalize this problem by assigning weights on the nets, thus allowing us to privilege the proximity/distance of some node pairs. Thus the weighted total square wirelength objective function can be formed in the following way.

Giving a set of modules M, $M = \{m_i, i = 1, 2, 3, ..., p\}$,with locations on (x_1, y_1), $(x_2, y_2), ..., (x_p, y_p)$, the total weighted square wire length objective function can be expressed as

$$\Phi(x, y) = \sum_{i,j=1}^{p} w_{ij}((x_i - x_j)^2 + (y_i - y_j)^2) = \mathbf{x^T Q x} + \mathbf{y^T Q y} \tag{6.1}$$

where $\mathbf{x} \in \mathbf{R}^p$ and $\mathbf{y} \in \mathbf{R}^p$ are the location vectors for the modules on X and Y dimensions. $\mathbf{Q} \in \mathbf{R}^{p \times p}$ is the matrix that represents the weighted connectivity of the topology graph, where the weight factors $\{w_{ij}, i = 1, 2, ..., p, j = 1, 2, ..., p\}$ are the matrix elements. $\mathbf{Q}$ is generated in the following way: 1) w_{ij} is 0 if there is no connection between modules m_i and m_j. 2) When modules m_i and m_j are connected, the value of w_{ij} is the weighting factor of the net between m_i and m_j. 3) The diagonal elements $\{w_{ii}, i = 1, 2, ..., p\}$, etc. of the matrix are the opposite of the sums of all off-diagonal elements on the same row.

$$w_{ij} = \begin{cases} \mathit{0} & i \neq j \text{ and no connection} \\ & \text{between } m_i \text{ and } m_j \\ \mathit{weighting\_factor(i,j)} & i \neq j \text{ and } m_i,\ m_j \text{ are connected} \\ & \\ -\sum_{k=1,k\neq i}^{p} w_{ik} & i = j \text{ (The sum of } w_{ik} \\ & \text{in the row i)} \end{cases}$$

When the network topology graph is connected, it can be proved that the matrix $\mathbf{Q} \in \mathbf{R}^{p\times p}$ constructed from this graph has the following properties [22]:

1. $\mathbf{Q}$ is positive positive semi-definite, and
2. is of rank $p-1$.

As mentioned in Section 6.3, MPSoC floorplanning is sometimes constrained by pre-defined I/O locations. To address these two different scenarios (with and without I/O constraints), we develop two approaches, as described in the following sections.

6.4.2 Floorplan Without I/O Constraints

Eq. 6.1 shows that the $\mathbf{x}$ and $\mathbf{y}$ location vectors are independent of each other; therefore, we can optimize the positions on the X and Y coordinates separately.

X-dimension Optimization

Since there is no I/O or boundary condition, we need to further normalize the objective function on the X-dimension by using the inner product $\mathbf{x^T x}$.

Now the normalized objective function of Eq. 6.1 can be rewritten as

$$\Phi\text{'}(x) = \frac{\mathbf{x^T Q x}}{\mathbf{x^T x}} \tag{6.2}$$

From the construction of matrix $\mathbf{Q}$, we note that the row sums of $\mathbf{Q}$ are zero, thus $\mathbf{Q}$ has a unit eigenvector $\mathbf{u} = (1, 1, 1, ..., 1)^T$. The associated eigenvalue is zero. We also note that $\mathbf{Q}$ is symmetrical and has rank $p-1$. Therefore, it has p non-negative real eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq, ... \leq \lambda_p \in \mathbf{R}$. The smallest eigenvalue is $\lambda_1 = 0$.

It can be proved that the first partial derivative with respect to the vector $\mathbf{x}$ of the normalized objective function is zero when

$$(\mathbf{Q} - \lambda\mathbf{I})\mathbf{x} = 0 \tag{6.3}$$

which yields a non-trivial solution of $\mathbf{x}$ if and only if $\mathbf{x}$ is the eigenvector of the corresponding eigenvalue of λ. Here $\mathbf{I}$ is the identity matrix.

It can be shown that the normalized objective function is bounded between the minimum and maximum eigenvalues, or

$$\lambda_{min} \leq \Phi\text{'}(x) = \frac{\mathbf{x^T Q x}}{\mathbf{x^T x}} \leq \lambda_{max} \tag{6.4}$$

The minimum eigenvalue, zero, yields the trivial solution of unit vector, where all nodes are to be placed at one single point. Therefore, the second smallest eigenvalue and the associated eigenvector yield the optimal solution.

Y-Dimension Optimization

Since we already use e_1 to form the location vector $\mathbf{x}$ on the X coordinate, the Y-dimension location vector $\mathbf{y}$ has to be formed from other eigen-vectors, otherwise, the modules will be placed in a diagonal line on the floorplan. This condition add one extra constraint to $\mathbf{y}$ vector.

$$\mathbf{y^T e_1} = 0 \tag{6.5}$$

Since the eigenvectors of the symmetrical matrix $\mathbf{Q}$ are orthogonal, we will choose for $\mathbf{y}$ the eigenvector corresponding to the third smallest eigenvalue.

Fig. 6.3 shows the screen-shot of the initial node locations of the five-ring Octagon network without I/O constraints. The locations on the X-Y plane are obtained directly from the first two non-zero eigenvectors of $\mathbf{Q}$. From the locations of the nodes, we can see that not only the regularity formation of the nodes is preserved, but also the hierarchical clustering of the cascaded Octagon rings is shown as well.

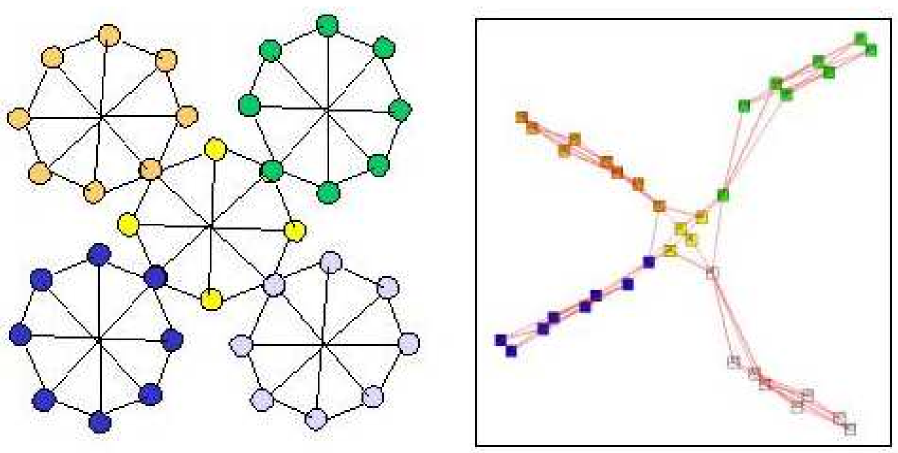

Figure 6.3: Initial Eigenvector Locations of 5-Ring Octagon Network Without I/O Constraints

Fig. 6.4 shows the initial locations of the cube-connect-cycles obtained from the eigenvectors of the matrix. Again, the formation of the nodes preserves the regularity as well as the hierarchy of the original topology.

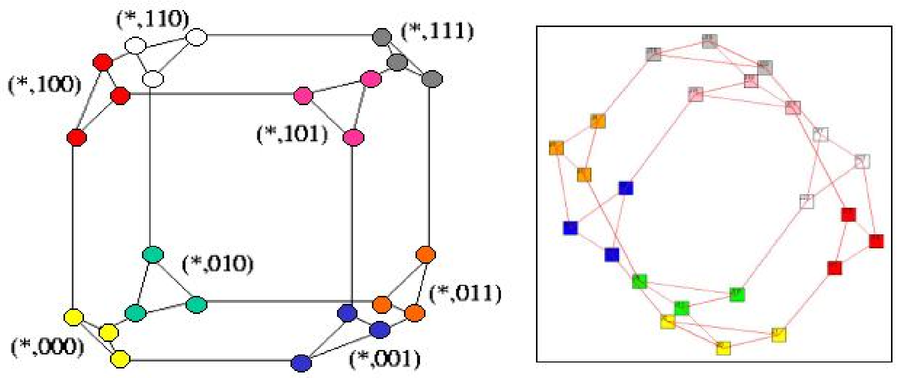

Figure 6.4: Initial Eigenvector Locations of Cube-Connected-Cycles Without I/O Constraints

6.4.3 Floorplan With I/O Constraints

For this problem, we can again decouple the optimization in the X and Y directions. We will first describe the optimization in the X direction, the optimization in the Y direction is similar.

If the positions of some modules are pre-fixed by the I/O constraints, we denote these modules as $M_f \subset M$, and their corresponding location vector in the X direction is denoted as $\mathbf{x}_f \subset \mathbf{x}$. Similarly, the locations of all the movable modules are denoted as vector $\mathbf{x}_c \subset \mathbf{x}$. The objective function can then be re-written as:

$$\Phi(x) = (\mathbf{x_c} \quad \mathbf{x_f}) \begin{pmatrix} Q_{cc} & Q_{cf} \\ Q_{fc} & Q_{ff} \end{pmatrix} (\mathbf{x_c} \quad \mathbf{x_f})^{\mathbf{T}} \tag{6.6}$$

Solving the zeros of the derivative of the objective function, we have

$$\mathbf{Q_{cc}x_c} = -\mathbf{Q_{cf}x_f} \tag{6.7}$$

Here $\mathbf{Q_{cc}}$ is a subset of the original matrix $\mathbf{Q}$. We know that $\mathbf{Q}$ is a symmetrical matrix with the rank of $p-1$. Therefore, if at least one node is fixed, $\mathbf{Q_{cc}}$ is non-singular and invertible, and the Eq. 6.7 has real solutions [3].

Fig. 6.5 shows the initial locations of the five-ring cascaded Octagon network with I/O constraints. The four bridge I/O nodes in the center Octagon ring are used as I/O nodes and placed at the four corners of the floorplan. The four I/O nodes are used as the fixed locations in the quadratic equation Eq. 6.6. Under the I/O constraints, the Octagon network shows a different formation than that without I/Os (Fig. 6.3). Nevertheless, the regularity as well as the hierarchical formation of the network is still preserved.

Fig. 6.6 shows the initial locations of the 2-ary 3-fly Butterfly switch fabrics. There are 8 node processors and 32 node switches in the network. The node processors are numbered from 0 to 7, as shown in Fig. 6.6. One node processor connects to two node switches, serving as input switch and output switch respectively. On the floorplan, we place the node processors 0, 1, 2, 3 on left side of the floorplan, while the node processors 4, 5, 6, 7 on the right side. This arrangement of node processors imposes I/O constraints on the Butterfly switch fabrics, because the switching nodes that serve as input and output have to be placed next to the corresponding node processors. The regularity formation of the switch fabrics is still preserved under these I/O constraints, as shown in the figure.

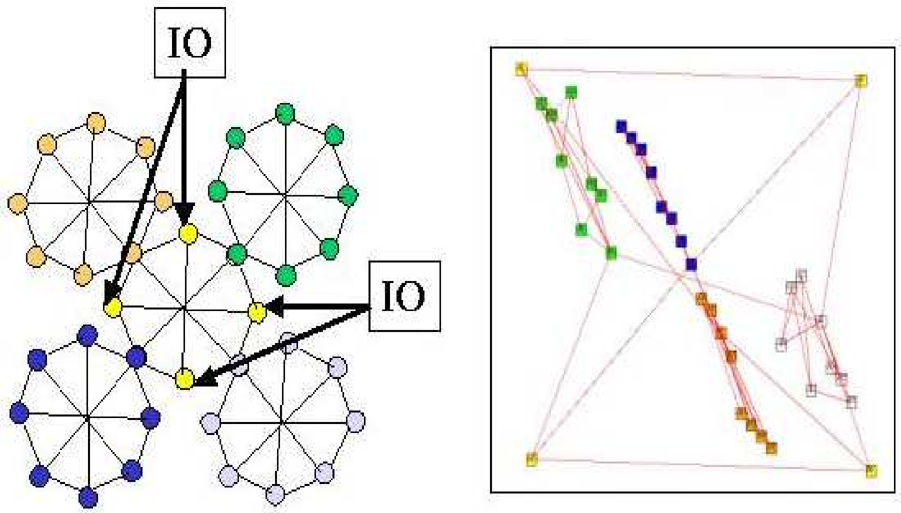

Figure 6.5: Initial Locations of 5-ring Octagon Network with I/Os on the Corners

6.5 Legalization

The node positions solved from the quadratic objective function optimization are real-valued numbers. However, the PE modules of each node are rectangular tiles and have to be abutted next to each other. The position from the location vector cannot be used directly in the tiling placement. Nevertheless, we can still use these values as relative locations and further legalize (quantize) the node positions.

The legalization procedure also consists of two steps: 1) sorting, where the nodes are ordered by the X and Y coordinates, and 2) packing, where the node blocks are assigned to the corresponding tiles (row and column positions).

In the sorting step, as shown in Fig. 6.7, the nodes are first sorted according to their X coordinates and evenly partitioned into several bins. The number of bins is equal to the number of columns. Then the nodes in each bin are further sorted according to their Y coordinates. After this step, the nodes are ordered in both the X and Y coordinates. The packing step will assign the nodes into the corresponding tiles in the $col \times row$ tile floorplan. The legalization procedure involves two linear sorting operations, which can be implemented with any existing sorting algorithms.

Fig. 6.8 shows the legalization results. Fig. 6.8a is the legalized floorplan from

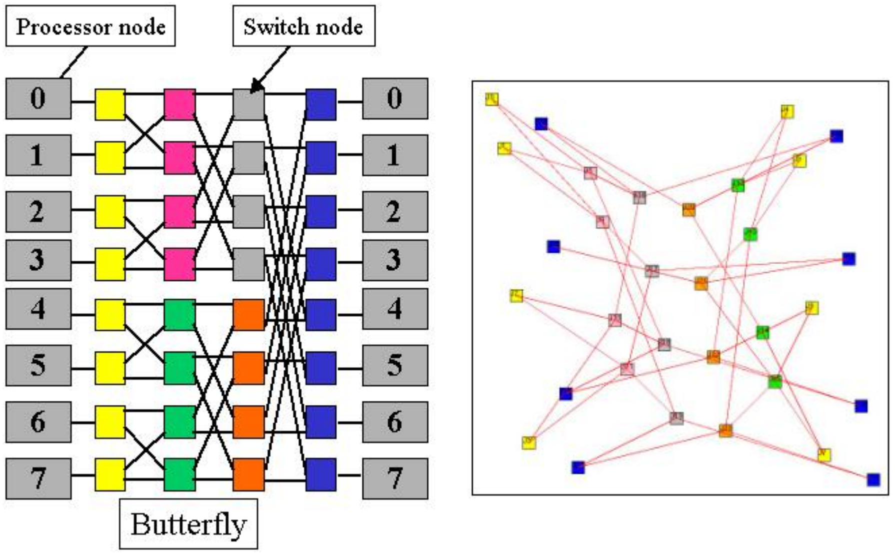

Figure 6.6: Initial Locations of Butterfly Network with I/O Constraints

Fig. 6.3 and Fig. 6.8b is legalized from Fig. 6.5. Both floorplans preserve the regularity and hierarchy formations of the original topologies. Furthermore, the floorplan also achieves a shorter total interconnect wirelength compared with other macro cell floorplanning approaches. We will show this comparison in details through several experiments.

6.6 Experiments

We have built a tool called *REGULAY* that implements the proposed floorplanning method. *REGULAY* is written in C++ with GUI written in Tcl/Tk. To the best of the my knowledge, there were no prior tools that target specifically on the MPSoC network floorplanning applications. Therefore, we compare the resulting floorplan and the total interconnect wirelength with the results obtained from ASIC floorplanning approaches. We choose UCLA MCM floorplanner [10] for this comparison. MCM is an open-source non-commercial tool that was originally designed to solve general ASIC floorplanning problems. Nevertheless, we perform this comparison to show that our method is particularly advantageous for MPSoC floorplans.

Fig. 6.9 shows the floorplan result of the cube-connected-cycles by *REGULAY*.

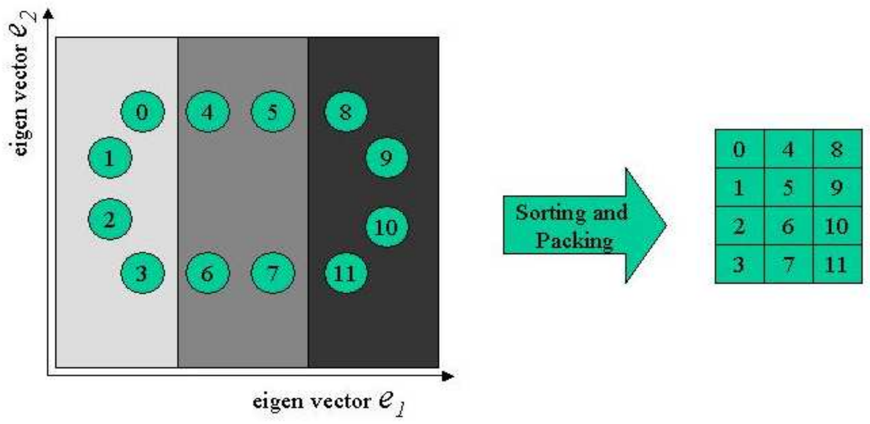

Figure 6.7: Legalization of the Node Locations by Sorting and Packing

There are total 24 nodes and 36 nets in the topology. For a better visualization of the regularity and hierarchy of the resulting floorplan, we assign different colors to different groups of nodes. There are no I/O constraints for the floorplan. From the floorplan formation, we can see that regularity information of the topology is well preserved by *REGULAY*.

The 4-ary 3-mesh network floorplan result is shown in Fig. 6.10. There are 64 nodes and 144 interconnects in this network, and the floorplan is an 8×8 tile array. Both the original 4-ary 3-mesh topology and the resulting floorplan are shown in the figure. Again, different groups of nodes are assigned different colors for a better visualization. As shown from the figure, *REGULAY* creates a satisfying results for this topology. All the nodes are placed into a regular and clustered formation on the floorplan. The locations of yellow nodes and blue nodes are symmetrical to each other, and the green nodes and white nodes are symmetrical too. This is because that yellow and blue nodes are "sandwiched" between green and white nodes in the original topology.

Fig. 6.11 shows the floorplan of 4-ary 3-cube torus network. There are total 64 node processors and 192 nets in this network, and they are also mapped into the same 8×8 tile floorplan. For a clearer view of the original network topology, we do not show all the 192 nets in the figure. No I/O locations are constrained. Compared with the 4-ary 3-mesh network, the torus floorplan shows a different formation of regularity

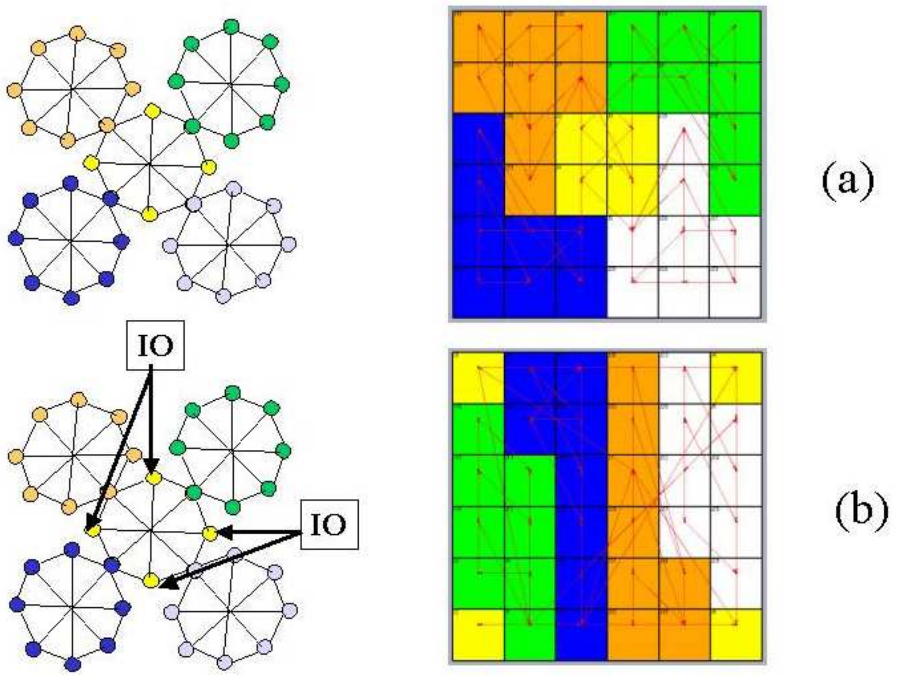

Figure 6.8: Legalized Floorplan of Octagon Networks with and without I/O Constraints

and clustering. As shown in this figure, the green and yellow nodes locations are symmetrical to each other, while the blue and white nodes are symmetrical too. This difference is caused by the "wrap around" nets added in the torus topology.

A 2-ary 3-fly Butterfly switch fabrics is tested as an example for indirect network. We use the same I/O constraints as described in Section 6.4.3, and the floorplan is legalized from the initial locations shown in Fig. 6.6. As illustrated in Fig. 6.12, under these I/O constraints, *REGULAY* creates a very dense arrangement of the switch fabrics, the regularity of the topology, as well as the locality of the I/O switches are well preserved.

Furthermore, we compare the total network wirelength and average net wirelength between *REGULAY* and UCLA MCM. Each PE in the network is $100\mu m \times 100\mu m$ in size. The wirelength is the Manhattan distance between two connected PEs. The results are compared in Table 6.1. We further calculate the wirelength reduction (both total and average) achieved by *REGULAY* over UCLA MCM. As shown in the table, the wirelength created by *Regular* is 1.8× to 6.5× smaller in all the benchmarks. Particularly, *REGULAY* shows even greater advantages in those more complex networks:

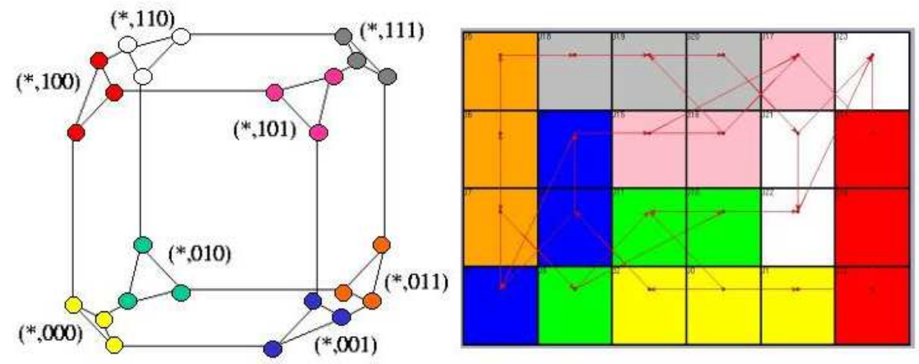

Figure 6.9: Floorplan of Cube-Connected-Cycles Network

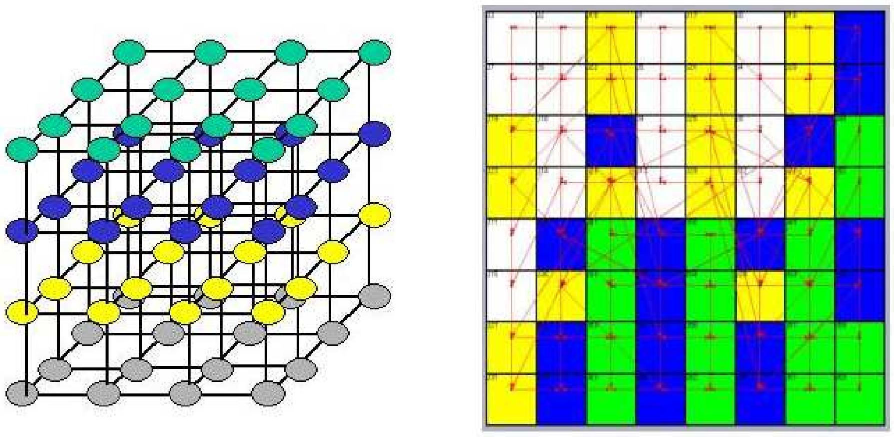

Figure 6.10: Floorplan of 4-ary 3-mesh Network

the 4-ary 3-mesh and torus network and the Octagon network achieve much higher wirelength reduction than those simpler networks.

6.7 Summary

In this chapter, we proposed a physical floorplanning method for MPSoC on-chip network and switch fabrics, and introduced *REGULAY*, a network floorplanning tool that implements the proposed methodology. Experiments show that *REGULAY* can automatically create an optimal floorplan that preserves the regularity and hierarchy formation of the network topology, while achieving significantly reduced total wirelength compared to traditional floorplanning tools.

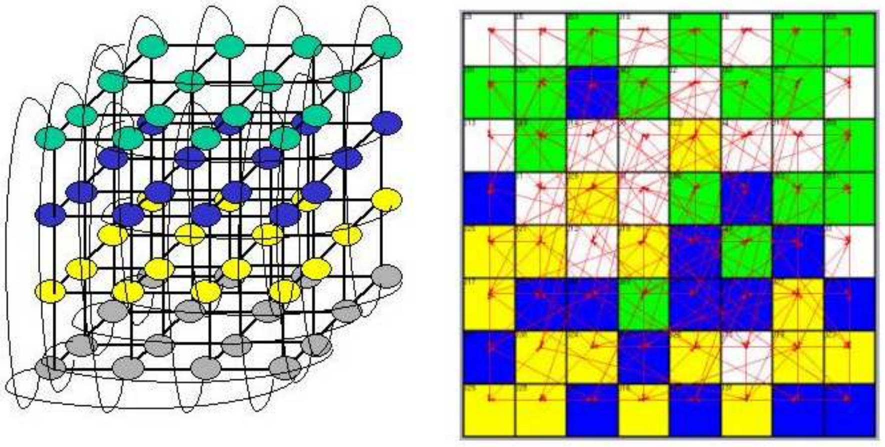
Figure 6.11: Floorplan of 4-ary 3-cube Torus Network

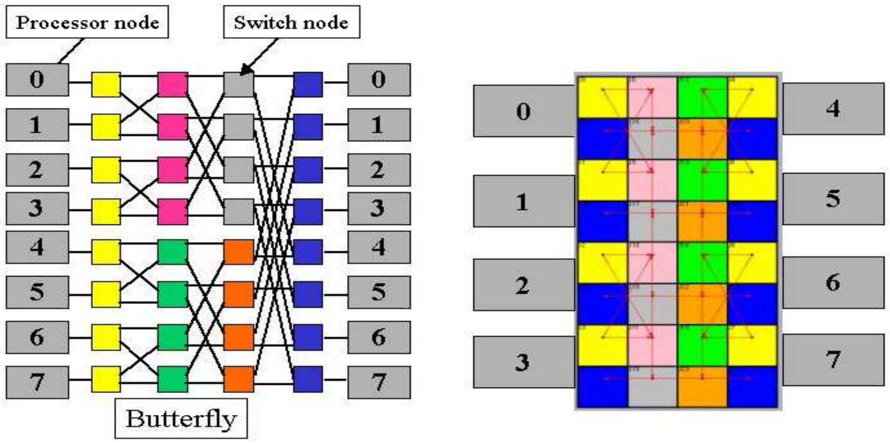

Figure 6.12: Floorplan Comparison of Constrained Butterfly Network

Table 6.1: Wirelength Comparison Between REGULAY and UCLA MCM

	REGULAY		*UCLA MCM*		improvement
	total wirelength	average wirelength	total wirelength	average wirelength	
5ring Oct	12400	206	54000	900	4.4
CCC	6000	166	10800	300	1.8
4ary 3mesh	28800	200	115200	800	4.0
4ary 3torus	60800	422	393600	2733	6.5
2ary 3fly	9600	200	19200	400	2.0

Chapter 7

Conclusions and Future Directions

The challenges of designing SoCs in 50-100nm technologies include coping with design complexity and providing reliable, high-performance operation and minimizing energy consumption. Starting from the observation that interconnect technology will be the limiting factor for achieving the operational goals, we envisioned a communication-centric view of design. In particular, we focused on energy and performance aspects for designing the communication infrastructure for future SoCs.

In this thesis, we analyzed several issues in the physical and network layers of the on-chip communication stack, and we proposed several strategies to effectively tackle the performance and energy challenge for on-chip communication networks.

1. On-chip packet routing should explore the schemes that can reduce the traffic contention without extensively using the buffers. Contention-look-ahead routing scheme, as proposed in this thesis, is proved to be an suitable candidate. It utilizes the abundant on-chip wiring resources to deliver and propagate the contention information among the neighboring nodes, thus helps the local nodes to make ‘”smarter” routing decision and reduces the contention occurrence.

2. On-chip communication can benefit from application-specific and platform-specific architectures, because the on-chip communication is less restrictively constrained from compatibility and adaptability issues. In this thesis, we show that changing the packet size can achieve a better performance-power trade-off.

The packet size impact on networks and node processors is also analyzed in detail.

3. Many researchers proposed a tile-based on-chip network architecture for future MPSoC platform. Therefore, physical planning of the network topology is an important issue, because different network topologies need to be planarized onto the two-dimensional silicon floorplan. Furthermore, the on-chip network floorplan needs to preserve the topological regularity and minimize the total wirelength. These requirements are conflicting with each other. In this thesis, we proposed a floorplanning methodology and a tool to solve this problem automatically.

Nevertheless, the methodologies presented in this thesis are far from serving complete on-chip network solutions. As communication-energy minimization will be a growing concern in future MPSoC technologies, an on-chip network implementation needs different layers to be integrated and interacted effectively. We outline some of the areas that, in our opinion, are particular critical for current and future networks-on-chip research.

- *Fault-tolerant and robust transmission.* Ever since the VLSI process technology moves into the deep sub-micron domain, on-chip data transmission can no longer be assumed reliable. The signal integrity problem in future NoC will become even worse as the device feature size continues to shrink. Increasing robustness can be achieved by 1) increasing the noise margin, 2) increasing the sensitivity of the receiver, or 3) error detection and correction. While increasing the noise margin will inevitably increase the power consumption, the later two options will likely be the directions for future NoC design exploration.

- *Heterogeneous network routing algorithms.* Most current NoC architectures adopt regular network topologies, i.e., two-dimensional mesh or torus. While packet routing and task mapping on these networks can benefit from regularity formation, more dedicated approaches need to be developed for heterogeneous NoCs. The diversity of future NoC applications will require application-specific node processors, i.e., image processing units and general CPUs may coexist on the same chip. NoC routing and mapping algorithms must be flexible enough to support different platform configurations.

- *Programming abstractions.* Developing adequate abstractions for NoC programming is a critical objective. Energy efficiency can be pursued by minimizing redundant communication, and by carefully balancing local computation and communication costs. A critical need in this area is the definition of hardware platform dependent high-level metrics, such as energy per local operation and energy per transmitted bit, which can help in first-cut exploration of the communication vs. computation trade-off during algorithm development.

- *Task-level analysis and optimization.* Future networks-on-chip design flow will need high-level optimization tools that can help designers mapping data-flow specifications onto target hardware platforms. The issues in this area include task splitting and merging (i.e., distributing the computation performed by a task among two or more computational nodes, and collapsing two or more tasks onto the same node), task allocation, as well as communication splitting and merging over available physical NoC links.

- *Code optimization.* Code optimization will hold an important place in future NoC software development. Many techniques are critical for further developments, such as: 1) techniques for parallelism extraction from a single task or a legacy applications, 2) techniques that reduce the memory footprint and improve access locality for both code and data, and 3) development of highly-efficient communication primitives.

- *Distributed operating systems.* The operating system that supports NoC operation cannot be centralized. Truly distributed embedded OSes are required to create a scalable run-time system. In addition to traditional functions (i.e., scheduling, interrupt handling), the NoC OS should natively support power management, bandwidth allocation and resource sharing.

The idea of networks-on-chip provides a new paradigm for future systems-on-chip architectural design and methodology. Many new challenges will emerge and thus create exciting research opportunities for the years to come.

Bibliography

[1] B. Ackland; et.al, "A single Chip, 1.6-Billion, 16-MAC/s Multiprocessor DSP", *IEEE J. Solid-State Circuits*, March 2000, pp. 412-424.

[2] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, C. A. Zeferino, "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network" *Proceedings of the Design Automation and Test in Europe*, March 2003, pp. 70-73.

[3] C. J. Alpert, T. Chan, D. J.-H. Huang, I. L. Markov and K. Yan, "Quadratic Placement Revisited", *Proceedings of the Design Automation Conference*, June 1997, pp. 752-757.

[4] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing", *Proceedings of 27th Annual International Symposium on Computer Architecture*, 2000, pp. 282-293.

[5] L. Benini, G. De Micheli, "Networks on Chips: A New SoC Paradigm", *IEEE Computer* January 2002, Volume: 35 Issue: 1, pp. 70-78.

[6] D. Bertozzi, L. Benini, G. De Micheli, "Low Power Error Resilient Encoding for On-Chip Data Buses", *Proceedings of the Design Automation and Test in Europe*, March 2002, pp. 102-107.

[7] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, R. Zafalon "Energy Estimation and Optimization of Embedded VLIW Processors Based on Instruction Clustering", *Proceedings of 39th Design Automation Conference*, June 2002, pp. 886-891.

[8] W.O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, L. Gauthier, M. Diaz-Nava, A.A. Jerraya, "Multiprocessor SoC Platforms: A Component-Based Design Approach", *IEEE Design & Test of Computers*, Vol.19 Nr.6, Nov-Dec, 2002, pp. 52-63

[9] H. Jonathan Chao, Cheuk H. Lam, Eiji Oki *Broadband Packet Switching Technologies: A Practical Guide to ATM Switches and IP Routers*, Wiley-Interscience Press, 2001.

[10] J. Cong, et al., "Relaxed Simulated Tempering for VLSI Floorplan Designs", *Proceedings of ASP Design Automation Conference*, Jan. 1999, pp. 13-16.

[11] D. E. Culler, J. P. Singh, A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, 1998.

[12] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, L. Benini, "Xpipes: a Latency Insensitive Parameterized Network-on-chip Architecture for Multi-Processor SoCs" *Proceedings of the International Conference on Computer Design*, Oct. 2003, pp. 80-85.

[13] W. J. Dally, "Performance Analysis of a k-ary n-cube Interconnect Networks", *IEEE Transactions on Computers*, June 1990, pp. 775-785.

[14] W. J. Dally, H. Aoki, "Deadlock -free adaptive routing in multicomputer networks using virtual channels", *IEEE Trans. on Parallel and Distributed Systems*, April 1993, pp. 466-475.

[15] W. Dally, B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", *Proceedings of the 38th Design Automation Conference*, June 2001, pp. 684-689.

[16] A. Dehon, "Compact, Multilayer Layout for Butterfly Fat-Tree", *ACM Symposium on Parallel Algorithms and Architectures*, 2000, pp. 206-215.

[17] J. Duato, S. Yalamanchili, L. Ni, *Interconnection Networks, an Engineering Approach*, IEEE Computer Society Press, 1997.

[18] U. Feige, P. Raghavan, “Exact analysis of hot-potato routing”, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, October 1992, pp. 553-562.

[19] E. Geethanjali, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, ”Memory System Energy: Influence of Hardware-Software Optimizations”, *Proceedings of International Symposium on Low Power Design and Electronics*, July 2000, pp. 244-246.

[20] P. Guerrier, A. Greiner, “A Generic Architecture for On-Chip Packet-Switched Interconnections”, *Proceedings of Design Automation and Test in Europe*, March 2000, pp. 250-255.

[21] R. I. Greenberg, C. E. Leiserson, “A Compact Layout for the Three-Dimensional Tree of Meshes”, *Applied Math Letters*, 1998, pp. 171-176.

[22] K. Hall, ”An r-dimensional Quadratic Placement Algorithm,” *Management Science*, vol. 17, no. 3, November 1970, pp. 219-229.

[23] L. Hammond, B Hubbert, M. Siu, M. Prabhu, M. Chen, K. Olukotun, “The Stanford Hydra CMP”, *IEEE MICRO Magazine*, March-April 2000, pp.71-84.

[24] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, “Network on chip: An architecture for billion transistor era”, *Proceeding of the IEEE NorChip Conference*, November 2000, pp. 166-173.

[25] R. Ho, K. Mai, M. Horowitz, “The Future of wires,” *Proceedings of the IEEE*, April 2001, pp. 490-504.

[26] J. Hu, R. Marculescu, “Energy-Aware Mapping for Tile-based NOC Architectures Under Performance Constraints” *Proceedings of ASP-Design Automation Conference*, Jan. 2003, pp. 233-239.

[27] J. Hu, R. Marculescu, “Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures” *Proceedings of Design Automation and Test in Europe*, March 2003, pp. 688-693.

[28] C. J. Hughes, V. S. Pai, P. Ranganathan, S. V. Adve, “Rsim: simulating shared-memory multiprocessors with ILP processors”, *IEEE Computer*, Volume: 35 Issue: 2 , Feb. 2002, pp. 40-49.

[29] F. Karim, A. Nguyen, S. Dey, “On-chip Communication Architecture for OC-768 Network Processors” *Proceedings of 38th Design Automation Conference*, June 2001, pp. 678-683.

[30] T. Kogel, M. Doerper, A. Wieferink, R. Leupers, G. Ascheid, H. Meyr, S. Goossens, “A modular simulation framework for architectural exploration of on-chip interconnection networks” *Proceedings of the 1st IEEE international conference on Hardware/software codesign & system synthesis*, Oct. 2003, pp. 7-12.

[31] S. Kumar, A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrj, and A. Hemani, “A network on chip architecture and design methodology”, *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, April 2002, pp. 105-112.

[32] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno. “Efficient Power Estimation Techniques for System-on-Chip Design”, *Proceedings of Design Automation and Test in Europe*, March 2000, pp. 27-32.

[33] D. Langen, A. Brinkmann, U. Ruckert, “High level estimation of the area and power consumption of on-chip interconnects”, *Proceedings of 13th IEEE International ASIC/SOC Conference*, Sep. 2000, pp. 297-301.

[34] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, John & Sons, Sep. 1990

[35] E. Nilsson; M. Millberg, J. Oberg, A. Jantsch, “Load Distribution with the Proximity Congestion Awareness in a Networks on Chip”, *Proceedings of Design Automation and Test in Europe*, March 2003, pp. 1126-1127.

[36] S. F. Oktug, M. U. Caglayan, “Design and performance evaluation of a Banyan network based interconnection structure for ATM switches“, *IEEE Journal on Selected Areas in Communications*, June 1997, pp. 807-816.

[37] C. Patel, S. Chai, S. Yalamanchili, D. Shimmel, “Power constrained design of multiprocessor interconnection networks”, *Proceedings of IEEE International Conference on Computer Design*, 1997, pp. 408-416.

[38] D. A. Patterson, J. Hennessy, *Computer Organization and Design, The Hardware/Software Interface*, Morgan Kaufmann Publishers, 1998

[39] Bryan Preas and Michael Lorenzetti *Physical Design Automation of VLSI Systems*, The Benjamin Cummings Publishing Company, 1988.

[40] F. P. Preparata, J. Vuillemin, “The Cube-Connected Cycles: A Versatile Network for Parallel Computation”, *Comm. of the ACM*, May 1981, pp. 300-309.

[41] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, ”Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip”, *Proceedings of Design Automation and Test Conference in Europe*, March 2003, pp. 350-355.

[42] W. T. Shiue, C. Chakrabarti, “Memory exploration for low power, embedded systems”, *Proceedings of the 36th Design Automation Conference*, June, 1999, pp. 140-145.

[43] J. P. Singh, W. Weber, A. Gupta, “SPLASH: Stanford Parallel Applications for Shared-Memory”, *Computer Architecture News*, vol. 20, no. 1, March 1992. pp. 20(1):5-44.

[44] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, A. Sangiovanni-Vincentelli, “Addressing System-on-a-Chip Interconnect Woes Through Communication-Based Design”,*Proceedings of the Design Automation Conference*, June 2001, pp. 667-672.

[45] D.Sylvester and K.Keutzer, “A Global Wiring Paradigm for Deep Submicron Design,” *IEEE Transactions on CAD/ICAS*, Vol.19, No. 2, Feb. 2000, pp. 242-252.

[46] T. Theis, "The future of Interconnection Technology," *IBM Journal of Research and Development*, Vol. 44, No. 3, May 2000, pp. 379-390.

[47] C. D. Thompson, *A Complexity Theory for VLSI, PhD thesis*, Carnegie-Mellon University, August 1980.

[48] J. Walrand, P. Varaiya, *High-Performance Communication Networks*, Morgan Kaufman, 2000.

[49] A. G. Wassal, M. A. Hasan, "Low-power system-level design of VLSI packet switching fabrics", *IEEE Transactions on CAD of Integrated Circuits and Systems*, June 2001. pp. 723-738.

[50] F. Worm, P. Ienne, P. Thiran, G. De Micheli, "An Adaptive Low Power Transmission Scheme for On-chip Networks" *Proceedings of the 15th international symposium on System Synthesis*, Aug. 2002, pp. 92-100.

[51] J. Wu; "A deterministic fault-tolerant and deadlock-free routing protocol in 2-D meshes based on odd-even turn model" *Proceedings of the 16th international conference on Supercomputing*, 2002, pp. 67-76.

[52] T. T. Ye, L. Benini, G. De Micheli, "Analysis of power consumption on switch fabrics in network routers" *Proceedings of the 39th Design Automation Conference*, June 2002, pp. 524-529.

[53] M. A. Youssef, M. N. El-Derini and H. H. Aly, "Structure and Performance Evaluation of a Replicated Banyan Network Based ATM Switch", *IEEE Symposium on Computers and Communications*, July 1999, pp. 258-266.