Time-Division-Multiplexing Based Hybrid-Switched NoC for Heterogeneous Multicore Systems

A DISSERTATION

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

Jieming Yin

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

Doctor of Philosophy

Antonia Zhai

November, 2015

Acknowledgements

Acknowledgments.

Abstract

Heterogeneous multicore systems have become prevalent and provided feasibility for balancing single threaded performance and high throughput requirements. Integrating multiple energy-efficient GPU accelerator cores with a traditional superscalar CPU cores onto the same die emerged as way to achieve the desired performance goal within a stringent power budget. As future heterogeneous systems scale both in number of cores and variety of computation resources, the on-chip interconnection networks (NoCs) must continuously support high-throughput low-latency on-chip data communication. As an integral part of a heterogeneous multicore system, the NoC must be able to perform on-chip data movement in an efficient manner. Meanwhile, future heterogeneous systems allow different types of compute units to have a unified address space, therefore, optimizing data sharing is crucial for improving system performance. Since the NoC plays an vital role in supporting data sharing, it must be designed correspondingly to improve the overall performance. This dissertation addresses the above two challenges, i.e., optimizing data movement and data sharing, in designing NoCs for heterogeneous multicore systems.

Enabling efficient data movement in terms of both performance and energy is critical in heterogeneous multicore systems, in which multiple applications are running simultaneously. In particular, NoCs must be designed to satisfy the communication requirements for both latency-sensitive CPU traffic and throughput-intensive GPU traffic. Traditional packet-switched NoCs, which have the flexibility of connecting diverse computation and storage devices, are facing great challenges to meet the performance requirements within the energy budget due to latency and energy consumption associated with buffering and routing at each router. In the first part of this dissertation, we take advantage of the diversity in performance requirements of on-chip heterogeneous computing devices by designing, implementing, and evaluating a hybrid-switched NoC that allows the packet-switched and

circuit-switched messages to share the same communication fabric by partitioning the network bandwidth through time-division multiplexing.

The second part of the dissertation focuses on maintaining global memory access order using the proposed hybrid-switched NoC, allowing memory operations to perform in parallel while a stronger memory consistency model can still be satisfied. The memory consistency model specifies the order how reads and writes from one thread are visible to other threads. Choosing of memory consistency models can largely impact performance, programmability, as well as hardware implementation. Enforcing a programmer-friendly strong memory consistency model while maximizing memory-level parallelism is challenging, especially in heterogeneous systems where data-parallel cores generate significant amount of outstanding memory requests. End-point ordering at cores is expensive since it prohibits a number of architecture optimizations. However, if correct order is provided in the interconnection network during request transmission, we can potentially improve the performance by parallelizing memory requests. The circuit-switched data path in a hybrid-switched NoC, guarantees message transmission ordering, therefore can be used as an infrastructure to preserve program order. Based on this observation, this dissertation proposes a hybrid-switched NoC attached with a light-weight token ring network to guarantee global memory access order.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modern computer systems have evolved from relatively simple microprocessors to complex heterogeneous architectures. Heterogeneous systems are widely adopted by a variety of areas including data center, PC, tablet, and smart phone. While the nowadays computing industry makes tremendous effort negotiating the trade-offs between power/energy and performance, the integration of CPUs, GPUs, on-chip storages, and memory controllers in current heterogeneous systems has shown its potential for improving both performance and energy efficiency [1, 2]. Recent published work point toward utilizing heterogeneous computing, in particular, the excessive GPU compute power to achieve high-performance exascale capability [3, 4]. With the support of heterogeneous computing, a logically tight integration of different types of computation units is enabled. Such integration is more than allowing various computation resources sharing the same die physically; it further allows various types of cores, for example, CPUs, GPUs, DSPs, and video encoders, to have a unified address space, which delivers simpler memory and programming model as well as better performance.

While shared-memory heterogeneous architectures can potentially eliminate the programming barriers and reduce communication latencies among different computation elements, two problems must be addressed: data movement and data sharing.

Data movement is a fundamental issue in designing a multicore system. It addresses how components are connected and how messages are transferred between components. Looking forward, system sizes in terms of CPU and/or GPU cores are likely to scale continuously to support increasingly complex processing for emerging applications such as virtual reality [5], "Big Data" and "Big Compute" [6], deep neural network [7], and so on. Efficient data movement mechanisms must be proposed to meet the requirements for various on-chip communication patterns under a stringent energy budget, as well as to support large scale systems. Since the on-chip interconnection network (NoC) is critical resource shared by all applications running on the system, it has the most exposure to data movement challenges. In other words, optimizing the NoC can potentially improve the overall performance for the heterogeneous system. Therefore, it is critical to design and optimize the NoC to provide sufficient bandwidth for different applications simultaneously and guarantees performance requirement.

From architecture's perspective, enabling data sharing involves defining and implementing the cache coherence protocol and the memory consistency model. Cache coherence protocol assures updates to memory locations are propagated in a system with private caches, which can largely impact the amount of on-chip traffic as well as the latency of a data read/write. Memory consistency model addresses when a thread sees an update to memory, the selection of which affects the programming model and limits the amount of optimizations allowed to improve the system performance. In a heterogeneous multicore system where computation, on-chip storage, and communication components are tightly integrated, designing coherence protocol or consistency model alone without considering the effect of the shared on-chip interconnect will result in suboptimal system performance. The on-chip interconnection network not only plays an important role in transmitting traffic related to cache coherency and memory consistency, it is also crucial in optimizing data sharing in the context of on-chip heterogeneity. A co-design of data sharing mechanisms and the NoC is a viable way of improving system performance.

1.1 Challenges Addressed in Dissertation

This dissertation addresses data movement and data sharing for heterogeneous multicore systems through designing an energy-efficient NoC.

1.1.1 Enabling Efficient Data Movement through Hybrid Switching

In heterogeneous multicore systems containing both CPUs and GPUs, on-chip communication exhibits variant traffic patterns. CPU-like superscalar cores generate moderate coherence and core-to-core data sharing traffic. In a modern CPU, the processor stalls when requested data are not returned. Therefore, traffic originated from or designated to a CPU core is latency-sensitive. While GPU-like data parallel cores, on the contrary, consists of a collection of data-parallel compute cores. Applications running on GPUs usually have high throughput requirement, hence GPUs generate throughput-intensive streaming traffic. Consequently, the NoCs in a heterogeneous system must be designed to handle both types of traffic efficiently in terms of performance and energy.

A packet-switched network offers the flexibility and scalability for connecting a large number of diverse devices; however, the buffering and routing necessary for each message introduces considerable delay and energy overhead. Prior works showed significant energy consumption associated with buffering at the routers [8, 9]. On the other hand, a circuit-switched network is cost-efficient for throughput-intensive traffic; however, infrequent use of the circuit-switched paths leads to under-utilization of on-chip resources.

A hybrid-switched NoC that supports both packet and circuit switching in the same fabric can potentially facilitate the design of NoCs that are both energy-efficient and flexible. Figure 1.1 gives an example of hybrid-switched NoC, which is built upon a mesh-based packet-switched NoC. In hybrid-switched NoCs, circuit-switched paths act as dedicated express channels between nodes. For example, in Figure 1.1, circuit-switched paths are set up from $R0$ to $R11$, from $R14$ to $R1$, and from $R5$ to $R12$. Packets taking advantage of these express channels can avoid performance and energy overhead associated with routing.

Thus, such networks can be potentially suited for heterogeneous multicore systems that consolidate diverse computation and caching components on a single die.

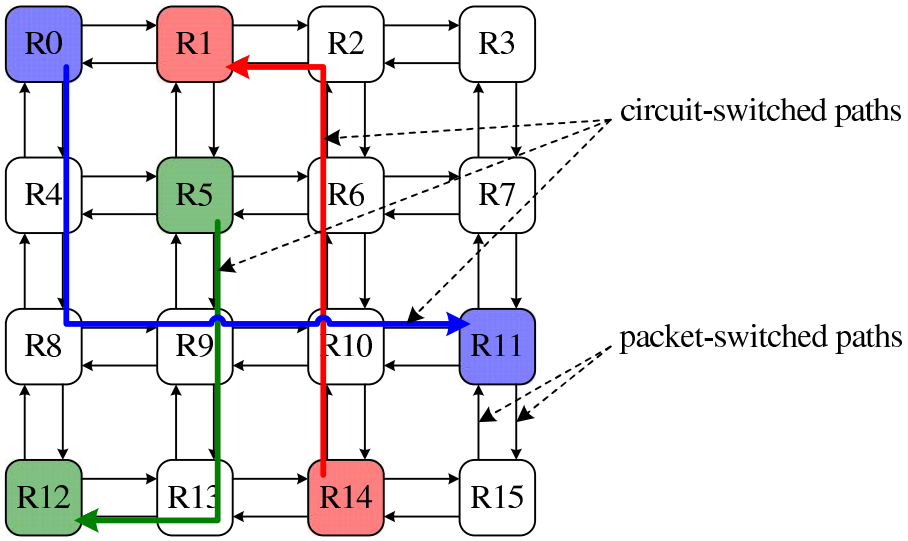


Figure 1.1: Hybrid-switched NoC example.

In this dissertation, we study the feasibility of applying hybrid-switched network to heterogeneous multicore systems; and design a hybrid-switched NoC, in which available bandwidth is divided through time-division multiplexing (TDM). In the TDM-based hybrid-switched NoC, link bandwidth is shared through the use of time-slots. In a given time-slot, the link is either allocated to the packet-switched network, or is exclusively dedicated to a given circuit-switched path. With minimal additional hardware, a TDM-based NoC can support a large number of circuit-switched paths. Utilizing these paths can improve system performance by reducing communication latency and alleviating network congestion. Furthermore, better energy efficiency is achieved by reducing buffering in routers and in turn enabling aggressive power gating.

1.1.2 Optimizing Data Sharing with In-Network Memory Access Ordering

In heterogeneous systems, choosing of memory consistency model is critical since a consistency model defines how memory accesses in a program will appear to the programmer. Different memory consistency models make different trade-offs among performance, programmability, and hardware complexity. Enforcing a stronger consistency model while maximizing memory-level parallelism is difficult; this difficulty is exacerbated by the presence of heterogeneous multicore systems, especially when some cores generate a large amount of outstanding memory accesses.

In a multicore system with a traditional packet-switched mesh NoC, memory access order is maintained at the core end. This is because the packet transmission delay in the interconnection network is non-determined, hence the access order cannot be guaranteed within the network. Fundamentally, memory accesses can be performed out of order as long as the program order is not violated. However, enforcing a particular order at the endpoint (core side) limits the opportunity of reordering and parallelizing memory accesses. If the NoC is designed to preserve memory access order during packet transmission, we can potentially improve the performance by parallelizing memory requests.

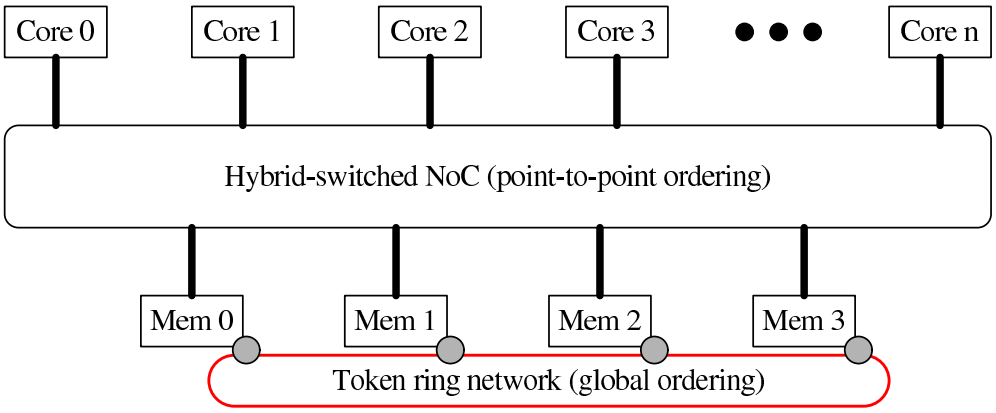


Figure 1.2: Global ordering NoC example.

The hybrid-switched NoC inherently preserves point-to-point order. In this dissertation, we attach the hybrid-switched NoC with a light-weight token ring network to guarantee global memory access order. As shown in Figure 1.2, the hybrid-switched NoC serves as the underlying communication infrastructure while the token ring network is used to preserve memory access order among multiple ordering points. Such design allows memory accesses to be performed in parallel, while still maintains their order globally.

1.2 Dissertation Contributions

This dissertation makes the following key contributions in designing energy-efficient NoCs for heterogeneous multicore systems.

1. This dissertation characterizes the performance requirements for heterogeneous on-chip traffic, and exploits the opportunities of utilizing the on-chip communication characteristics to achieve energy-efficiency.

2. This dissertation designs, implements, and evaluates a hybrid-switched NoC that supports both packet-switched and circuit-switched messages by partitioning the network through time-division multiplexing. We demonstrate the feasibility of implementing TDM-based hybrid switching by extending an existing router design with hybrid switching capability.

3. Taking advantage of hybrid switching, this dissertation proposes an NoC design which provides in-network memory access ordering for directory-based heterogeneous multicore systems. The proposed NoC aims to provide a stronger memory consistency model while maximizing memory-level parallelism through in-network message ordering.

4. This dissertation investigates and compares the scalability of TDM-based hybrid-switched NoC against traditional NoC designs including ring, hierarchical ring, and

packet-switched network. We demonstrate hybrid switching in mesh-based NoCs as energy efficient alternative for large-scale many-core processors with high-throughput data-parallel accelerators.

1.3 Dissertation Outline

The organization of the rest of this dissertation is outlined as follows:

- Chapter 2 presents the TDM-based hybrid-switched NoC. It also defines the concept of packet slack, based on which the proposed hybrid-switched NoC is able to achieve better energy efficiency.
- Chapter 3 optimizes the TDM-based hybrid-switched NoC for more efficient data movement, and conducts a detailed study of the proposed network in the context of heterogeneous multicore system.
- Chapter 4 addresses the data sharing optimization for heterogeneous multicore systems by designing an NoC that supports in-network memory access ordering.
- Chapter 5 compares multiple NoC design choices including ring-based, mesh-based packet-switched and hybrid-switched network, and studies their scalability using both synthetic and realistic workloads.
- Chapter 6 presents the conclusion and points out the future research directions.

Chapter 2

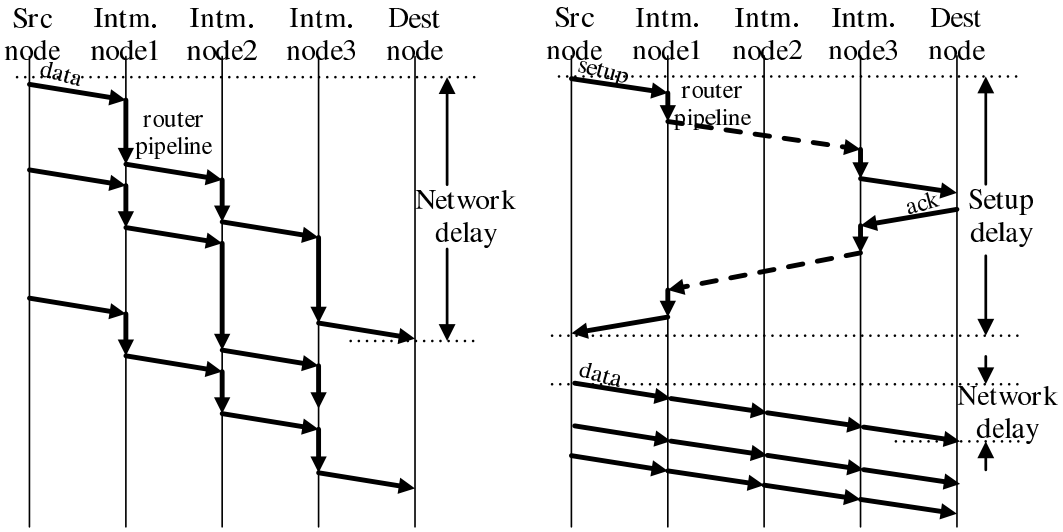
TDM-based Hybrid-Switched NoC

A hybrid-switched NoC that supports both packet and circuit switching can potentially facilitate the design of NoCs with both energy-efficiency and flexibility. In hybrids-witched NoCs, circuit-switched paths act as dedicated express channels between nodes, and packets taking advantage of these channels can avoid performance and energy overhead associated with routing. Thus, such networks can be potentially suited for heterogeneous multicore systems that consolidate diverse computation and caching components on a single die. How can these two switching mechanism be consolidated onto a single network? This Chapter will find the answer to this question.

2.1 Hybird-switched NoC Overview

A hybrid-switched network allows both circuit-switched messages and packet-switched messages to traverse through the network concurrently. These two types of traversals have different timing and energy characteristics.

Figure 2.1(a) shows the timing diagram of a packet-switched message: each message is buffered, routed and forwarded through the link separately at each router along the path from source to destination. Messages incur significant delay and energy consumption



(a) Packet-switched flow (b) Circuit-switched flow

Figure 2.1: An overview of hybrid-switched network.

at each router due to repeated buffering and routing. For irregular traffic patterns, this mechanism can be efficient. However, for a large number of messages utilizing the same route through the network, this repetition can be wasteful. Figure 2.1(b) shows the timing diagram of a circuit-switched message: a separate configuration message is sent through packet switching to reserve a dedicated connection between the source and the destination, and then data traverses through the network as circuit-switched messages without buffering or routing at individual routers. Therefore, circuit switching a message results in lower delay and energy consumption.

2.2 Dividing On-chip Bandwidth through TDM

Jerger et al. [10] propose a space-division multiplexing (SDM) based hybrid-switched NoC. In this network, links are physically partitioned into planes. Each individual plane is allocated to a given circuit-switched connection. The SDM-based hybrid-switched NoC demonstrates performance improvement for coherence-based traffic where most network

messages are short and the number of circuit-switched paths is essentially small. However, an SDM network serializes packets as they are forced to use a single plane even though the other planes are idle, resulting in packet serialization delay and intra-router contentions [11]. These limitations become significant performance bottlenecks when handling throughput-intensive traffic in heterogeneous systems. Furthermore, the number of circuit-switched paths is fundamentally limited by the number of planes, which cannot be increased arbitrarily. As NoCs scale up in size, or as many-to-few-to-many traffic pattern increases with the integration of data-parallel many-core accelerators [12], the number of planes becomes insufficient for reserving circuit-switched paths. Thus, we must seek an alternative sharing mechanism for hybrid-switched NoCs in the presence of heterogeneous many-core systems.

To address the limitations of SDM-based NoC, we propose a hybrid-switched NoC in which packet- and circuit-switched messages share the communication fabric through time-division multiplexing (TDM). In a TDM-based hybrid-switched network, time is discretized into recurrent time-slots. In a given time-slot, the available link bandwidth is either allocated to the packet-switched network, or it is exclusively dedicated to a given circuit-switched path. Compared to an SDM-based NoC, TDM is able to avoid serialization observed in SDM-based NoC because each flit can utilize the full link bandwidth when transmitting. Consequently, each message results in fewer flits, and fewer flits per message can reduce the risk for congestion. Furthermore, by partitioning the network bandwidth in time domain, the number of possible circuit-switched paths is theoretically unlimited. In reality, with minimal additional hardware, a TDM-based NoC can support a large number of circuit-switched paths.

The differences between packet- and circuit-switched interconnection architecture provide us with opportunities to optimize NoC performance as well as reduce energy consumption. In heterogeneous systems, data parallel cores generate throughput intensive traffic. Circuit switching this traffic can significantly reduce energy overhead. Superscalar cores,

on the contrary, have moderate throughput requirements. Thus, circuit switching this traffic has only a moderate energy benefit compared to that of data parallel cores. However, superscalar cores can benefit from the latency reduction associated with circuit switching.

2.3 TDM-based Hybrid-Switched Network

In a TDM-based network, links are shared through the use of time-slots. The assignment of time-slot is kept in slot tables that are maintained at each router. Each slot table entry contains a valid bit and an output port id. For each incoming flit, the router looks up the slot table and determines whether the flit should be packet- or circuit-switched. Slot table entries are updated by explicit path configuration messages. Figure 2.2 demonstrates the path configuration process as seen by a single router. For simplicity, only two input ports are shown in this example. Configuration messages arrive at either of these two ports.

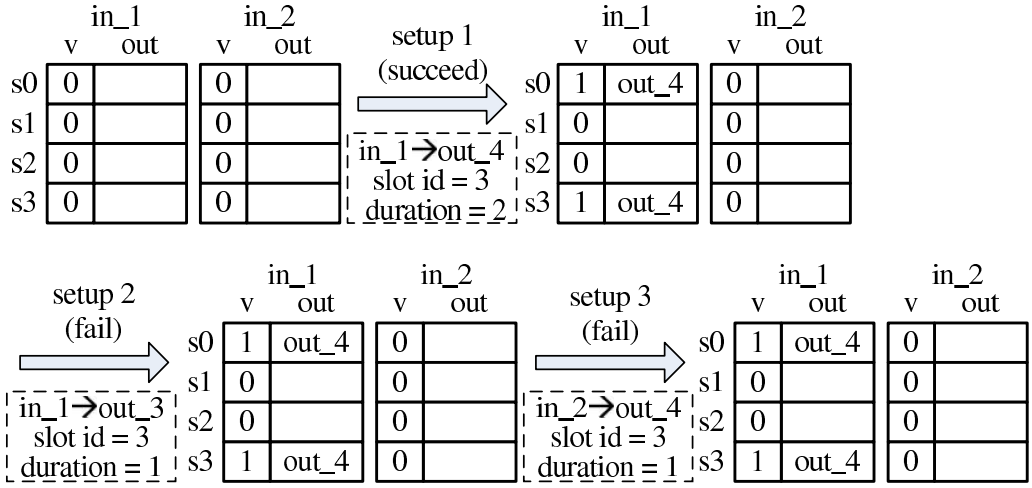


Figure 2.2: Slot table state transition of a single router, in response to 3 setup messages. In a setup message, slot_id specifies the initial time slot when a reservation begins; duration specifies the number of consecutive slots the reservation requires.

1. Initially, no path is reserved and all slot table entries are invalid.

2. A setup message $setup1$ arrives at input port in_1 and requests output port out_4, reserving two consecutive cycles starting from time-slot $s3$. Since all of the tables are empty, $setup1$ is successful and the corresponding slot table is updated. Notice that slot reservation is performed in modulo S fashion (S is the size of the slot table), so $s3$ and $s0$ are reserved for $setup1$.

3. $Setup2$ reserves time slot $s3$ from in_1 to out_3. This setup would fail because the slot has already been allocated. The slot tables remain unchanged and a setup failure acknowledgement is sent back.

4. $Setup3$ reserves time slot $s3$ from in_2 to out_4. This setup would fail because out_4 is already reserved for in_1 at slot $s3$. In this case, setup failed due to a conflict at the output port.

5. Teardown messages are used to remove circuit-switched paths when they are no longer needed. When a teardown message arrives, the valid bits of the corresponding entries are reset so that the slots can be reused by other paths.

No circuit-switched messages should arrive if the time slot is not reserved. However, if a time-slot is reserved for circuit-switched flits and no circuit-switched flit arrived in this cycle, a packet-switched flit will steal this slot. We refer to this technique as *time-slot stealing*. Detailed discussion can be found in Section 2.3.3. In the following sections, we elaborate four design choices to improve network utilization.

2.3.1 Path Configuration

Once the source node has decided to construct a dedicated connection, circuit switching path configuration is performed. In the proposed hybrid-switched network, circuit-switched paths are set up and torn down through explicit configuration messages, which are sent through the packet-switched network. There are three types of configuration messages:

setup_msg, which creates a circuit-switched connection between the source and the destination; *teardown_msg*, which destroys an existing connection; and *ack_msg*, which indicates a setup success/failure.

A *setup_msg* contains the source and destination node id, and a slot id indicating the time slot to reserve. Every router along the path of the *setup_msg* checks the availability of the output port at given slots. If it is free, the output port is reserved, and the *setup_msg* is forwarded with slot id incremented by 2 since the circuit-switched network is two-stage-pipelined (the increment is in modulo S fashion, where S is the number of entries in a slot table). Otherwise, the setup is aborted and an acknowledgement is sent back to the source node, indicating a setup failure. If the *setup_msg* successfully reaches the destination, an acknowledgement will also be generated with an *ack_msg* sent back to the source node, indicating a path setup success. After *ack_msg* reaches the source node, depending on the path setup success/failure, either: (a) the source-destination connection is registered at the source node, and packets can be sent in circuit-switched fashion; or (b) a *teardown_msg* is generated to destroy the corresponding connections.

A *teardown_msg* contains the same information as a *setup_msg*. It traverses through the same path as the corresponding *setup_msg* by referring to the slot tables. Therefore, the *teardown_msg* will eventually arrive at the node where the setup failed, and invalidate all reserved slots. Acknowledgement is not required for a *teardown_msg*. Since there is no guarantee that all source-destination connections can be set up successfully in a single try, we allow re-sending the failed setup with a different slot id. However, if the slot tables are too small to hold all possible source-destination reservations, source nodes will repeatedly and unsuccessfully try to set up a connection. We do not implement complex path setup algorithms since the re-send mechanism introduces minimal performance penalty. By selecting the slot table size properly, the overhead can be minimal. In our experiment, configuration messages correspond to less than 1% of total traffic.

A reserved connection can be repeatedly used by the same source-destination pair.

Once a connection has been idled for a long period, it becomes the candidate to be destroyed when new setup requests come in. If all entries inside a slot table are reserved by circuit-switched connections, packet-switched messages might suffer starvation. Although unlikely to happen, in order to prevent starvation, slot allocations are prohibited when the percentage of reserved entries exceeds a threshold. In this work, we arbitrarily set the threshold to 90%. It is worth pointing out that in the proposed network, packet transmission does not wait for a successful circuit-switched path setup, which means a message can be sent through packet-switched network WHILE its path setup is performed in parallel. Therefore, the performance overhead caused by path setup is negligible.

Reserving consecutive slots: When reserving slot table entries, we can either reserve a single slot, or multiple slots consecutively. In this work, consecutive slot reservation is preferred because the entire message can reach its destination within a few cycles. The reservation duration depends on the data length, which is 4 slots in our setup since a cache line is 64-byte long and the flit width is 16 bytes. A duration field is added into the construction messages indicating the number of slots each reservation requires.

Path selection: During circuit-switched path setup, judiciously selecting paths can potentially reduce congestions and balance workload across all routers. To achieve the goal, we apply adaptive routing algorithm to configuration messages. Details of the algorithm can be found in [13].

2.3.2 Time-division Granularity

Time-division granularity corresponds to the percentage of bandwidth reserved for each connection, and in turn determines how often a circuit-switched message for a particular reservation can travel on the reserved path. It also determines the number of slot table entries. Smaller slot tables correspond to coarser granularity and a smaller interval between consecutive circuit-switched messages on the same path. However, smaller slot tables also imply fewer circuit-switched paths. Larger slot tables, on the other hand, provide

finer granularity and can hold more reservations. Nevertheless, messages might stall for more cycles before transmission. Furthermore, larger slot tables also lead to more energy consumption.

In a hybrid-switched NoC, slot table size is an important design parameter and can be workload dependent. We propose to dynamically determine the slot table size. In particular, we start with activating only a small portion of the slot tables and power-gating the rest, and then double the number of active entries as needed until all entries are activated. In other words, the slot table size is a function of the network size as well as the number of circuit-switched paths. While the former can be statically determined, the latter is workload dependent. More slot table entries are activated when path allocation continuously fails. Once the capacity of the slot table is increased, all slot tables are reset, and the path setup procedure restarts. Since program behaviors are relatively stable, we expect slot table reconfiguration to rarely occur.

2.3.3 Hybrid-Switched Router Architecture

We extend the virtual-channeled wormhole router to support hybrid switching. The router architecture is shown in Figure 4.2. The following components are added to a classical router architecture [14]: slot tables, circuit-switched latches, and de-multiplexors which forward incoming messages to either the packet- or the circuit-switched pipeline.

At time T, an arriving flit is forwarded to either the packet- or the circuit-switched pipeline, depending on the entry in the slot table corresponding to time T. Before a circuit-switched flit's arrival, the crossbar is configured in advance by retrieving the output port information from the slot table, and the reserved output is guaranteed to be available in T. Thus, the flit simply proceeds through the router in a single cycle without buffering. It reaches the downstream router in time $T + 2$ after the link transmission stage, which takes another cycle. Packet-switched flits, on the other hand, traverse through the router pipeline.

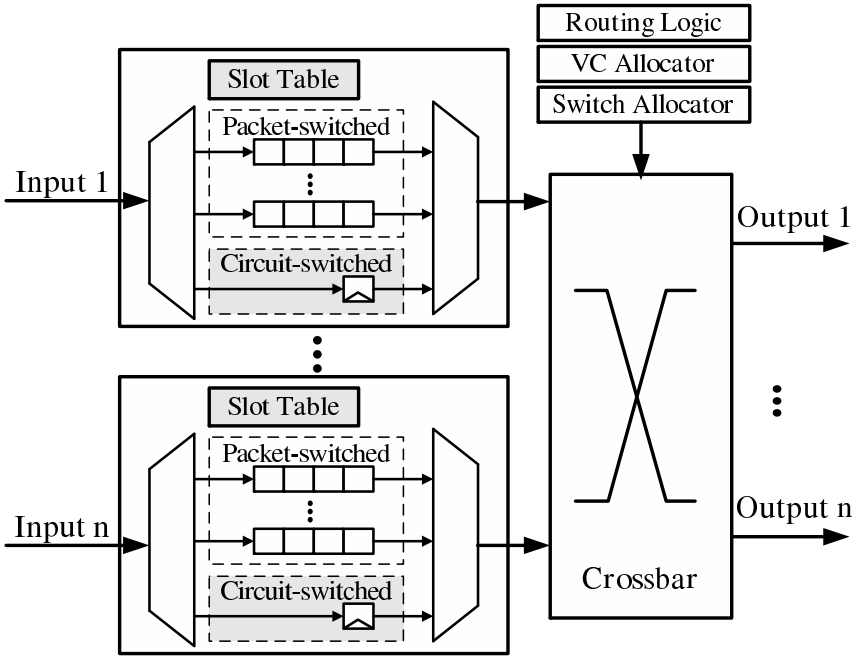


Figure 2.3: Hybrid-switched router structure.

In a reserved time-slot, it is possible that no circuit-switched flit is presented, leaving the crossbar idle. When this happens, we allow a packet-switched flit to utilize the crossbar, referred to as *time-slot stealing*. To enable time-slot stealing in time T, the router must be informed at time $T-1$ whether a circuit-switched flit will arrive in the next cycle. In the proposed design, a designated one-bit CS_flit_enable signal is used to propagate this information from the upstream router to the downstream router. If a circuit-switched flit is arriving, the signal will be enabled when the crossbar from the upstream router finishes its connection allocation. Otherwise, time-slot stealing can be performed in the downstream router.

Reducing Dynamic Energy Overhead

The circuit-switched pipeline is similar to the highly optimized single-cycle packet-switched router, but operating at a much higher frequency. Since it is a single-cycle router pipeline,

the crossbar configuration must be finished by the time a circuit-switched flit arrives at the input port. In a hybrid router, the crossbar is configured based on the switch allocator and the slot. As we mentioned earlier, the content of slot table has to be read out before the arrival of a circuit-switched message. There are two ways to guarantee the timing: 1) access the slot tables every cycle, even if there is no incoming circuit-switched messages; or 2) the upstream router notifies downstream router if a circuit-switched message is sent out, then slot table is accessed ahead of the message's arrival. We adopt the later approach because it is more energy-efficient. And the same CS_flit_enable signal is used to achieve energy efficiency.

However, when traffic load is sufficiently high, frequent slot table access can still cause significant dynamic energy overhead. Therefore, we propose a simple yet efficient technique to further reduce the slot table accesses energy. Notice that consecutive slots are reserved for data transmission in the proposed hybrid router, of which the content stored in the slot table are identical. We can thus latch the line address given to the slot table for 4 consecutive cycles, avoiding the flipping of wires saves dynamic energy from the decoder and table reads.

As shown in Figure 2.4, a modulo-4 up counter with additional control logic is used to prevent the slot table input from changing. The counter consists of two JK flip-flops; the CS_flit_enable signal is given as the input to the counter. External clock is connected to the clock input. When CS_flit_enable is 1, the counter begins to count up at the rising edge of each clock pulse; when CS_flit_enable is 0, the counter holds the output value when the clock pulse arrives. A binary 0 can quickly be loaded into the counter by applying a momentary low at the CLR input. The CLR is used in particular for resetting the counter when CS_flit_enable becomes 0, and it always overrides the other inputs. The outputs of the counter, namely, Q1 and Q0, are passed to a NOR gate as inputs. The output of the NOR gate, together with the CS_flit_enable, are used as an enable signal to drive the latch. The line address is generated by a modulo-S counter which is not shown in the

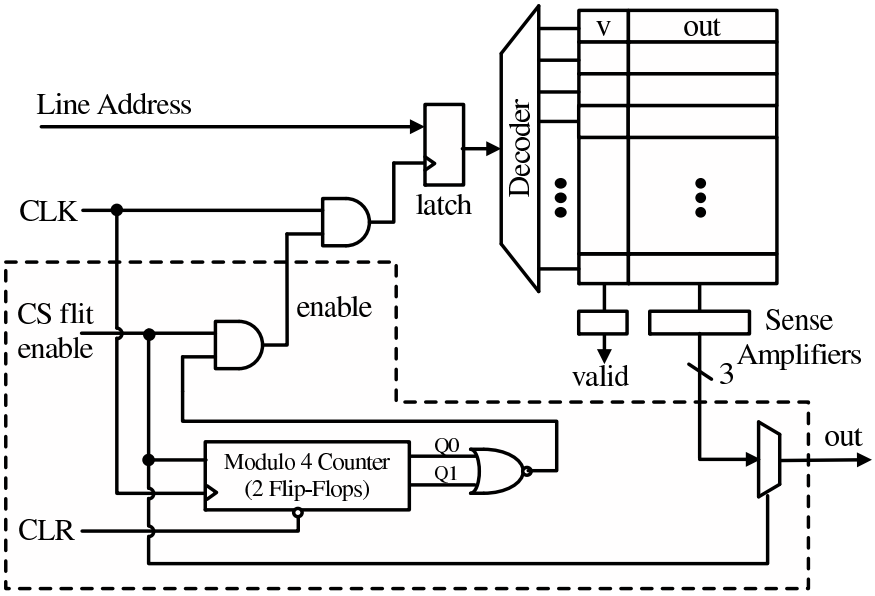


Figure 2.4: Energy efficient slot table access

figure, where S is the slot table size; the line address simply increases every cycle to consult the table entries one after another.

Initially the modulo-4 counter is set to 00, so the output of the NOR gate is a logic 1. Therefore, when the CS_flit_enable signal from the upstream router arrives, the access to the slot table is granted. Moreover, CS_flit_enable is given to validate the *out* value acquired from the slot table. This value is then sent to the switch allocator for the crossbar configuration. In the next cycle, the CS_flit_enable signal is still 1 since a message takes 4 cycles to transmit, and the counter begins to count up, resulting in a logic 0 coming from the NOR gate. Hence, although the line address increases, its first value is latched. In this way, no dynamic energy dissipation is required since the input to the slot table does not flip. The same *out* value is used to configure the crossbar. After 4 cycles, the two-bit counter becomes 00 again, and the access to the slot table is granted. If the CS_flit_enable is 1, the same process repeats; however, if CS_flit_enable is 0, counter value holds and slot table access will not occur.

2.4 Switching Decision

To improve network performance and reduce network energy consumption, deciding whether a message should be forwarded with packet or circuit switching is multifaceted. From energy perspective, as more messages traverse the network through circuit switching, less energy is consumed for buffering and routing; on the other hand, maintaining the slot tables incurs energy overhead. From performance perspective, circuit-switched messages spend less time traversing the network; however, messages might stall and wait for a circuit-switched time slot before transmitting. In the proposed design, a circuit-switched path is only reserved for source-destination pairs communicate frequently to ensure a high utilization of the path. For example, when a core generates a large number of memory accesses, the NoC will reserve circuit-switched paths between the core and the memory components. Once a path is established, not all messages are circuit-switched. Consider the case in which a circuit-switched message is stalled, waiting for its time slots. Such phenomenon leads to increase in network latency when messages only traverse short distances within the network. This issue can be mitigated by allowing a message to be packet-switched if the established path corresponds to a time slot that requires stalling.

To achieve optimal utilization of the network and minimize network energy consumption, we packet switch all CPU traffic while hybrid switch only GPU messages, since GPU traffic has a higher throughput requirement compared to CPU traffic. However, not all GPU messages are circuit-switched. A GPU message is considered to be circuit-switched only when no performance penalty is caused. To predict whether circuit switching a GPU message causes performance degradation, we introduce a metric called *packet slack*.

2.4.1 Exploiting Slack in GPU

A GPU consists of a collection of data-parallel compute cores referred to as streaming multiprocessors (SMs). A large number of lightweight threads are allocated to each SM and are batch-scheduled on the single-instruction-multiple-data (SIMD) pipeline in a fixed

size group called a warp. In other words, warps are the basic execution units in a GPU. All the threads within a warp execute the same instruction but operate with different data values. When a thread in a particular warp encounters a load miss, the entire warp is context-switched out, and another available warp is scheduled.

Since all outstanding memory requests are sent via the interconnection network in the form of packets, we first define the *packet slack* as the number of cycles a packet can be delayed without affecting the overall execution time.

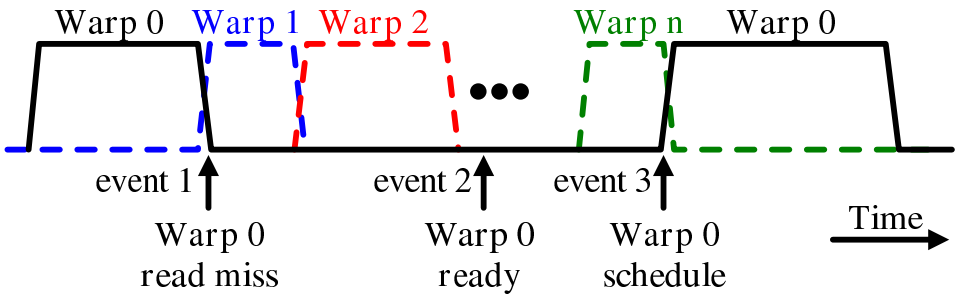


Figure 2.5: Execution of a GPU warp and packet slack

Figure 2.5 demonstrates the execution of a GPU warp (Warp 0) and its packet slack. In an SM, a warp is available if all operands are ready. Assume we have n available warps in an SM that can be scheduled at a given time; Warp 0 is scheduled and executed first. After a few cycles of execution, a read miss occurs, which is denoted as Event 1 in the figure. Then a hardware context switch is performed and Warp 1 gets scheduled and executed. Similarly, when Warp 1 encounters a read miss, the next ready warp is scheduled accordingly. Event 2 represents the cycle when the memory request of Warp 0 is sent back, which means Warp 0 is ready to schedule. However, depending on the scheduling algorithm, Warp 0 may not be scheduled instantly since there are other ready warps waiting in the scheduling queue in front of Warp 0. After a read miss of the last available warp, Warp 0 is rescheduled at Event 3. Packet slack for Warp 0 is the time between Event 2 and 3. It is evident that if we delay the network transmission of the packet for a number of cycles smaller than the slack, the overall execution time will remain unaffected.

Intuitively, packet slack is related to the number of available warps ready to schedule in an SM. The more available warps an SM has during a period of time, the longer the slack it can exploit for each packet. However, the number of available warps within each SM varies from application to application. To further understand the relation between packet slack and the number of available warps, we manually delay the scheduling of each warp for a fixed number of cycles after it is ready. For example, in a 10-cycle-delay experiment, if a warp is ready to be scheduled at cycle t_0, we arbitrarily prevent it from scheduling until cycle t_1=t_0+10 even if there is no warp available to schedule in between. In the worst case, execution is stalled and system performance is degraded due to the arbitrary delay.

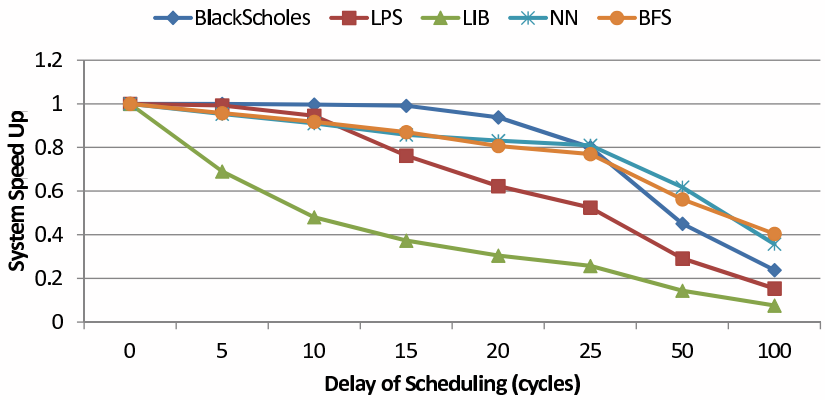


Figure 2.6: System performance degradation due to delay of warp scheduling.

Figure 2.6 shows the performance degradation due to the arbitrary delay of warp scheduling. We notice that for some of the GPU applications such as BlackScholes, large schedule delay can be tolerated. By contrast, for applications such as Lib, even a few cycles of delay will result in significant performance degradation. Our hypothesis is that delay tolerable applications have more warps allocated in each SM. To verify our hypothesis, we collect the average number of available warps for each application during the whole execution as shown in Figure 2.7. Here we set a 95% system speed up as the acceptable performance degradation level. The *tolerable delay cycles* is the maximum number

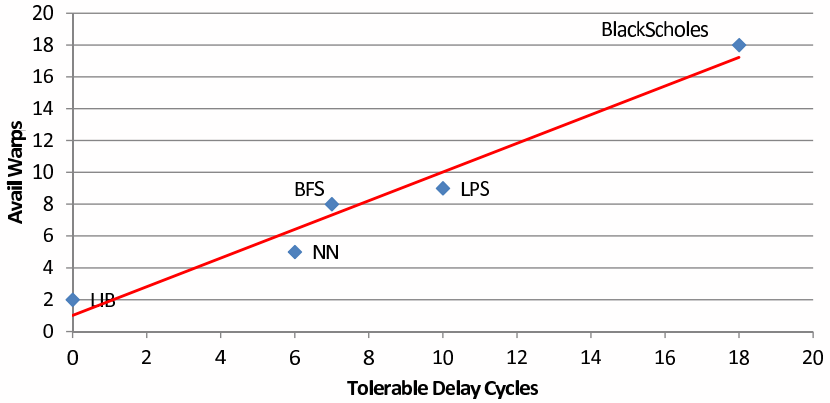


Figure 2.7: Linear relationship between number of available warps and tolerable delay cycles.

of cycles a particular warp scheduling can be delayed without exceeding the 95% speed up threshold. From the figure, we observe that there is a linear relationship between the average number of available warps and tolerable delay cycles for all the GPU applications.

Based on the observation, we predict the approximate slack of a packet as the number of available warps in the SM in a given cycle. An additional register ($Avail\_warp\_reg$) which keeps track of the number of available warps for each SM is added in our design. When a packet is sent from/to a particular SM core, the $Avail\_warp\_reg$ is referred and the packet slack is set according to the value register.

Once packet slack is determined, we can easily predict whether circuit switching a packet results in performance degradation. Since the waiting time before a circuit-switched packet can be injected to network is known (by referring to the slot reservation), we circuit switching a packet only when the waiting time is smaller than the packet slack. This decision does not consider network transmission delay hence is conservative, as the network delay of the circuit-switched path is shorter than its packet-switched counterpart.

2.5 Related work

Energy in NoCs has been shown to be a significant contributor to the on-chip energy consumption [8, 15, 16]. A variety of techniques have been deployed to reduce NoC energy dissipation. Dynamic Voltage and Frequency Scaling (DVFS) is an effective technique for reducing energy consumption [17, 18, 19]. However, DVFS becomes a performance bottleneck in heterogeneous systems with significant throughput requirement since reducing network operation frequency increases packet delay and degrades throughput [13]. Although flexible-pipeline routers [20, 21] can be deployed to reduce the latency penalty by dynamically combining the router pipeline stages, degradation in throughput remains an issue which harms the performance of throughput-intensive applications.

In virtual-channeled packet-switched network, buffers account for a significant portion of energy consumption [8, 15, 16]. Reducing buffer read/write operations or even removing buffers from the routers can largely eliminate the network energy consumption. Express virtual channels (EVC) is proposed to avoid the need for packets to stop and be buffered at intermediate nodes, therefore saves buffer energy and improves throughput [22]. While sharing the motivations, EVC incurs significant hardware overhead due to the complexity for credit management to ensure buffer availability. This complexity limits the length of the express links, and limits the benefit EVC can exploit. In SCARAB [23] and BLESS [24], energy reduction is achieved by removing input/output buffers. However, the performance of bufferless designs degrades under heavy NoC workloads, thus these designs are not suitable for heterogeneous systems with throughput-intensive cores. Dynamic buffer management strategy such as ViChaR [25] improves buffering efficiency by adjusting the depth and number of virtual channels based on network traffic, but they can not eliminate buffer energy consumption as on-chip workload varies.

TDM circuit switching is proposed to provide bandwidth and latency guarantees in Æthereal NoC [26], in which time slots are reserved for each guaranteed flow along the path. However, circuit switching transmission starts after acknowledgement is sent back,

and guaranteed flows are not allowed to use excess bandwidth when the network is under-utilized. MANGO NoC [27] reserves virtual channels for guaranteed flows, therefore large number of buffers as well as costly switching modules are required, leading to significant energy dissipation. Our work utilizes TDM to facilitate resource-sharing between packet- and circuit-switched traffic to reduce NoC energy consumption, which requires different implementations and optimizations such as time-slot stealing, circuit-switched path sharing, dynamic time-division granularity adjusting, and aggressive power gating. Reconfigurable circuit-switched NoCs, such as [28], take advantage of the deterministic traffic patterns of certain applications and create circuit-switched paths for these applications. These paths only handle circuit-switched traffic for fixed sources and destinations, thus are inadequate for heterogeneous multicore systems that have non-deterministic traffic patterns. SDM hybrid-switched NoCs have been proposed to provide hard QoS support in SoC [29, 30, 31, 32]. It is possible for SDM to work together with the proposed TDM mechanism. However, the hardware and performance overhead of such hybrid systems must be carefully evaluated.

Jerger et al. proposed a SDM-based hybrid-switched NoC design with a prediction-based coherence protocol, enabling significant latency reduction [10]. However, their work introduces packet serialization delay, which affects the performance when handling high throughput traffic. SMART selectively circuit-switch packets using asynchronous repeaters [33]. However, SMART is unable to optimize multi-flit packets and only allows for short circuit-switched paths. Instead of using TDM, Kilo-NoC [34] uses both VC routers and elastic buffer routers in certain nodes of the network, to reduce energy consumption and provide scalability. Kilo-NoC mainly focus on enabling QoS and ensure fairness inside NoC in large-scale systems.

2.6 Summary

In this Chapter, we design and implement a hybrid-switched network that supports both packet-switched and circuit-switched messages by partitioning the network through time-division multiplexing (TDM). We demonstrate the feasibility of implementing TDM-based hybrid switching by extending an existing router design with hybrid switching capability. To alleviate performance impact brought by hybrid switching, we introduce the concept of GPU packet slack and explore this opportunity with traffic generated by GPU cores that tolerate memory latencies through context switching between threads.

Chapter 3

Hybrid-Switched NoC Optimization

In this Chapter, we describe three optimizations to enhance the resource utilization and energy-efficiency of the hybrid-switched network.

3.1 Circuit-switched Path Sharing

In the hybrid router described in Section 2.3, each slot table entry is reserved for a particular source-destination pair. When the communication path of a source-destination pair partially overlaps an existing path, sharing the slot table entries of the existing path, rather than reserve a new path, can potentially improve the utilization of the slot table. In this section, we propose hitchhiker-sharing that allows intermediate nodes along a circuit-switched path to share the connection if the message is sent towards the same destination as the reserved slots; and vicinity-sharing that allows path sharing if the destination of a message is in the vicinity of the destination node of some reserved path. Compared to time-slot stealing, which enables packet-switched traffic to utilize an unused time-slot reserved for circuit-switched traffic, the proposed path sharing mechanisms allows multiple

circuit-switched paths to share entries in the slot tables.

3.1.1 Hitchhiker-Sharing

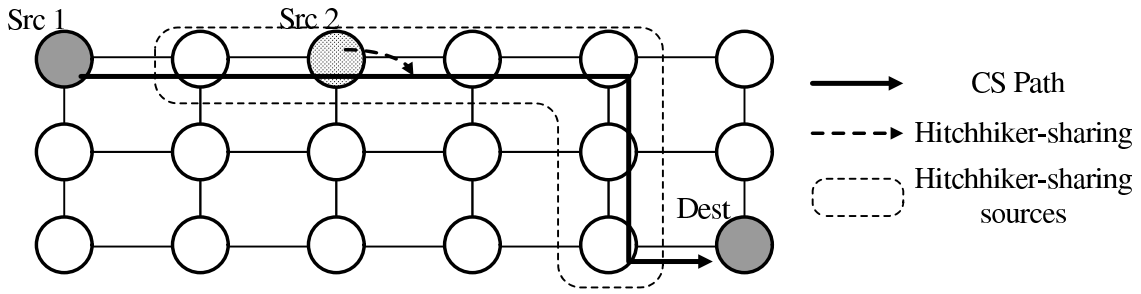


Figure 3.1: Hitchhiker-sharing scheme.

As shown in Figure 3.1, a circuit-switched path is already constructed from source node $Src1$ to destination node $Dest$. If another source node, for example $Src2$, resides along the circuit-switched path, it is allowed to share the circuit-switched connection with $Src1$ when the corresponding time slots are not occupied by messages from $Src1$. To make decision of whether hitchhiker-sharing is feasible, $Src2$ has to know the destinations of circuit-switched connection presented in its slot tables, and the time slots reserved for these connection. This information is stored in a *Destination Lookup Table* (DLT) in $Src2$, and is updated when a new connection is setup in the router of $Src2$. When sending a message, if the circuit-switched path originated from $Src2$ does not exist but hitchhiker-sharing can be performed, the message is still considered to be circuit-switched. If the path sharing fails due to contention, this message will be sent in packet-switched manner. If path sharing towards a particular destination fail continuously, a circuit-switched path setup request is thereby generated from $Src2$. In Figure 3.1, nodes circled by dashed line are candidates that can potentially share the circuit-switched path with $Src1$.

Hitchhiker-sharing scheme does not require modifications to the slot table. Instead, the DLT is required for each node to store the destination and its corresponding time-slot information. The size of the DLT is dependent on the network size and slot table size.

For example, in a k-by-k network, $2\lceil log_2 k \rceil$ bits are required to represent the destinations. Meanwhile, for slot tables with S entries, $\lceil log_2 S \rceil$ bits are needed to indicate the time-slot for each destination. Moreover, we introduce a 2-bit saturation counter to keep track of the failure of circuit-switched path sharing. If the counter becomes '10', a circuit-switched path setup is generated and the entry is removed from the table. The size of an 8-entry DLT in the evaluated system described in Section 3.4.1 is less than 16 bytes.

3.1.2 Vicinity-Sharing

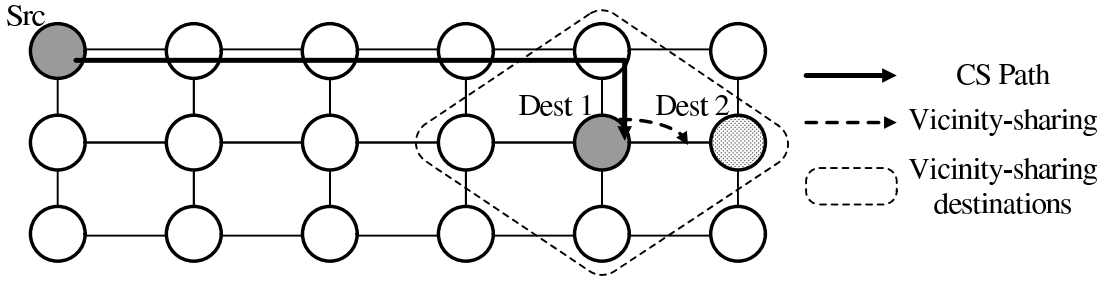


Figure 3.2: Vicinity-sharing scheme.

Figure 3.2 is an example of vicinity-sharing scheme. A circuit-switched path is set up from Src to $Dest1$. If a message is sent from Src to another destination, for example $Dest2$, but dedicated circuit-switched path has not been constructed. Instead of setting up a new connection, this message is allowed to use the existing circuit-switched path between Src and $Dest1$, as long as $Dest2$ and $Dest1$ are adjacent. After reaching $Dest1$, the message will be sent towards it destination through packet-switched network. Similar to hitchhiker-sharing scheme, when contention occurs at the source node, path sharing fails and the message is again packet-switched. A circuit-switched path setup requirement is generated if path sharing fail continuously. In Figure 3.2, vicinity-sharing destination candidates are circled by dashed line.

To support vicinity-sharing, a 2-bit saturation counter is required for each reservation to keep track of the circuit-switched path sharing failure. Moreover, comparison logics are

used to compute the vicinity-sharing candidate nodes. A header flit is required in addition for vicinity-sharing, because messages go through packet-switched network after hop-off. Therefore, when reserving circuit-switched paths, one additional time slot is required.

Notice that hitchhiker-sharing and vicinity-sharing can be combined and applied to the same circuit-switched path. In other words, messages can hop-on at intermediate nodes and get off at nodes close to their destination. With circuit-switched path sharing deployed, most of the frequent connections can still be satisfied even with smaller slot tables.

3.2 Aggressive VC Power Gating

Since the circuit-switched network alleviates the burden on packet-switched counterpart, we can turn off under-utilized router buffers to save static energy while still satisfying the performance requirement. Increasing the number of VCs allows the network to support a higher injection ratio without significant increase in network latency. However, when the buffer pressure is alleviated by circuit switching a portion of the traffic, fewer VCs are needed. Thus, we propose a dynamic VC tuning policy where the number of activate VCs is periodically adjusted based on VC utilization (μ) in comparison to two thresholds, $Threshold_{High}$ and $Threshold_{Low}$. If μ exceeds $Threshold_{High}$, one set of virtual channel will be activated; if μ is below $Threshold_{Low}$, one set of virtual channel will be turned off. The VC must be evacuated before adjusting, and the downstream routers are updated with the new VC count.

3.3 Evaluation on Synthetic Workload

Compared to real applications, synthetic traffic makes it possible to study the NoC behavior under a wider range of on-chip traffic. In this section, we evaluate the performance and energy consumption impact of the proposed NoC with three traffic patterns: 1) uniform random(UR): destinations are randomly selected; 2) tornado(TOR): messages from (x, y)

are sent to $(x + \frac{k}{2} - 1, y)$, where k is the number of nodes in both x and y dimensions; and 3) transpose(TR): messages from (x, y) are sent to (y, x) [35]. In addition, we also compare the performance of our proposed work against [10].

3.3.1 Evaluation Infrastructure

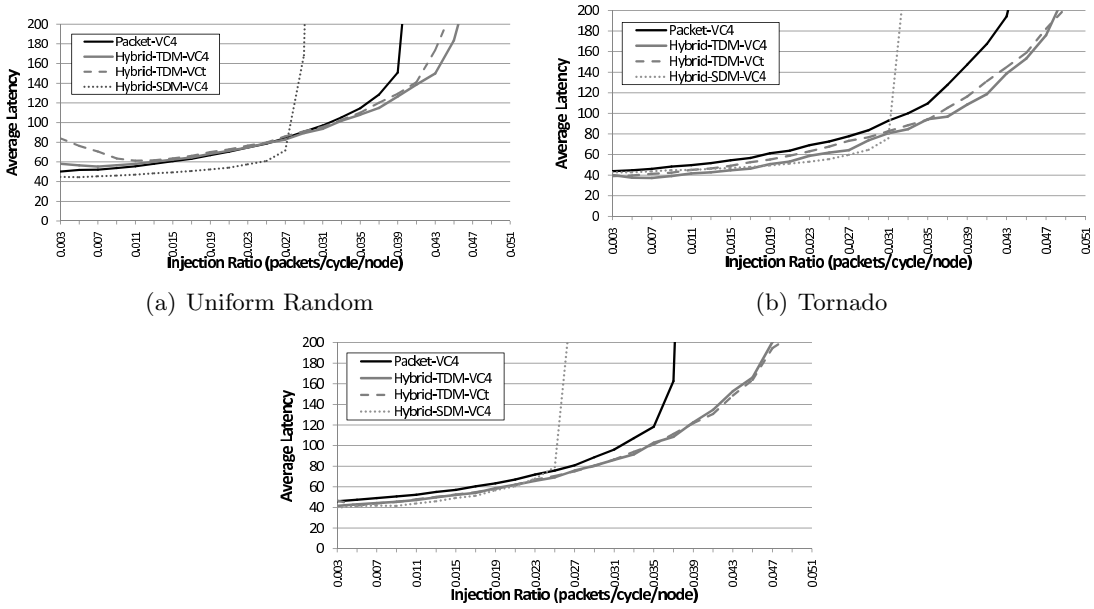
The interconnection network and its power model are based on Garnet [14] and Orion 2.0 [16], respectively. We update the router power model with the hybrid-switched architecture. We consider a 36-node mesh with router parameters shown in Table 3.1. The network is warmed up with 1000 packets and simulated for 100,000 packets.

To compensate for overestimation in router area and inaccuracy in router power in Orion 2.0, as described in [36, 37], we revise the technology parameter in Orion 2.0 and augment our router with an RTL implementation. The SRAM bit cell spacing has been updated based on [37]. To avoid the inaccuracy in the multiplexer-based crossbar model, we assume a matrix crossbar in our evaluation. Furthermore, we use an RTL model to adjust the router area [38]. Both packet- and hybrid-switched routers are synthesized using Nangate Open Cell Library for 45nm technology. The total area of a packet-switched router is $0.177mm^2$; and $0.188mm^2$ for a hybrid-switched router, leading to 6.2% area overhead.

Topology	36-node, 2D-Mesh
Technology	45nm technology at 1.0V, 1.5GHz
Routing	Minimal Adaptive Routing (configuration packet) X-Y Routing (other packet)
Channel Width	16 Bytes
Packet Size	1 flit (configuration message) 4 flits (circuit-switched packet) 5 flits (packet-switched packet, circuit-switched packet when vicinity-sharing applied)
Slot Tables	128 entries
Virtual Channels	4/port
Buffer size per VC	5 in depth

Table 3.1: Router parameters for synthetic workload evaluation.

3.3.2 Impact on Performance



(a) Uniform Random

(b) Tornado

(c) Transpose

Figure 3.3: Average latency as a function of injection rate with synthetic traffic patterns.

Figure 3.3 shows the load-latency curves with different traffic patterns for baseline packet switching with 4VCs(Packet-VC4), SDM-based hybrid switching with 4VCs(Hybrid-SDM-VC4) [10], TDM-based hybrid switching with 4VCs(Hybrid-TDM-VC4), and TDM-based hybrid switching with aggressive power gating(Hybrid-TDM-VCt). For the evaluated traffic patterns, TDM-based hybrid-switched routers improve the throughput by 14.7%, 9.3%, and 27.0%, respectively. Forwarding messages through circuit switching alleviates the buffer pressure and router congestion, which in turn improves the throughput. Exception is found in UR traffic, where TDM-based NoC suffers longer latency. This is because the TDM network required large slot tables to capture as many communication pairs as possible. Therefore, a circuit-switched packet has to wait for long time before its time-slot arrives, resulting in longer delay.

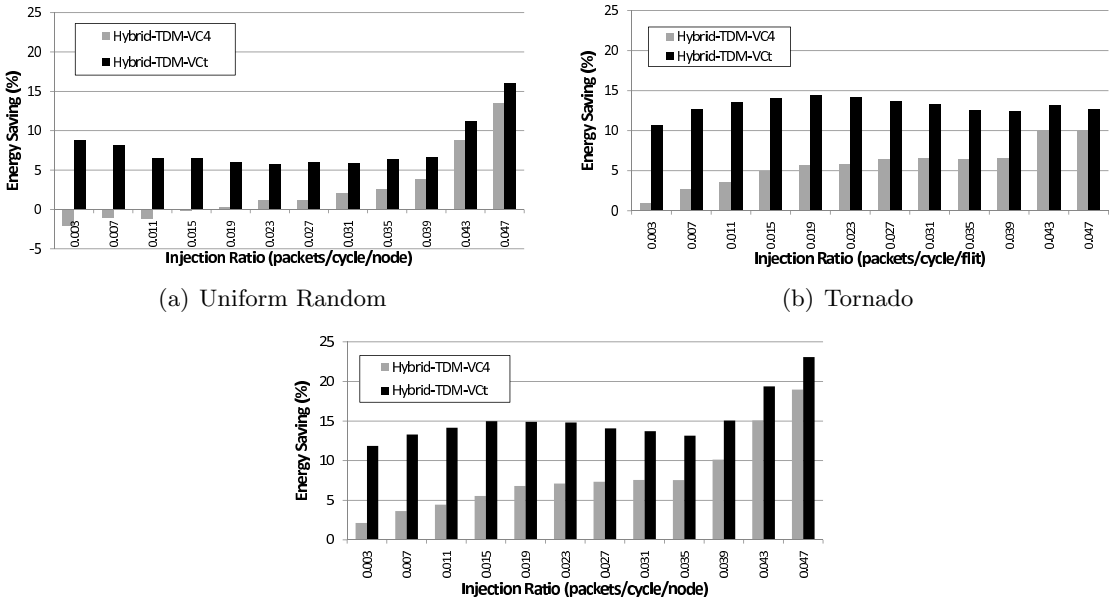
Compared to the TDM-based NoC, the SDM-based NoC reduces latency for all traffic

patterns under low/moderate injection rates, while the proposed TDM-based NoC is able to achieve a higher level of throughput. The TDM-based NoC is competitive in latency reduction in TOR and the TR traffic, but not in UR traffic for reasons explained above. However, the SDM-based NoC is unable to scale as the injection rate increases. This is because in wormhole switching, the SDM-based NoC must serialize packets as they pass through a single plane. This serialization increases the number of flits per packet and increases the risk for congestion and intra-router contention. Eventually, this serialization manifests as a penalty in throughput at high injection rates.

It is worth noticing that the performance of the SDM-based NoC can improve with increased link bandwidth, because packet serialization becomes less serious with wider links for a fixed packet size. On the other hand, as the network increases in size, the performance gap between the TDM- and the SMD-based NoC will widen. This is because more circuit-switched paths are expected in larger networks, but SDM-based NoC switching is unable to support a large number of paths. Overall, we believe that TDM-based hybrid switching is more desirable for many-core systems in the presence of throughput-intensive traffic.

3.3.3 Impact on Energy

Figure 3.4 shows the network energy saving compared to the baseline *Packet-VC4*. SDM-based NoC is not shown because it increases the network energy consumption. The energy saving under uniform random traffic is relatively small. Negative energy saving is observed under low injection rate. This is again because large slot tables are required to capture all possible communication pairs in uniform random traffic. Under low injection rate, the energy saving from circuit switching messages does not offset the overhead caused by large slot tables. In other words, when the number of source-destination pairs is reasonable so that most circuit-switched reservations can be held in slot tables, the hybrid-switched network can efficiently reduce the network energy consumption.



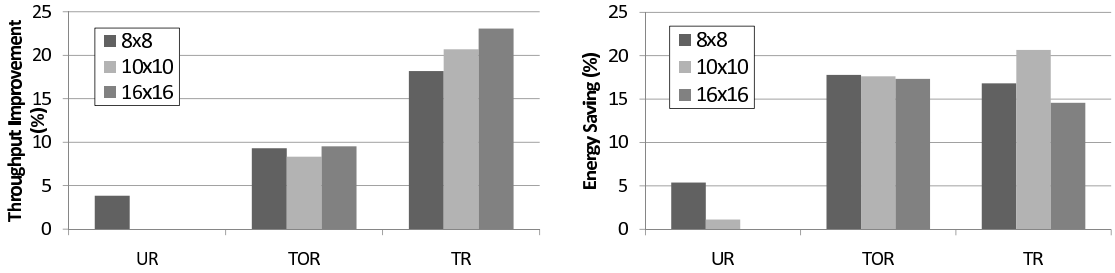
(a) Uniform Random

(b) Tornado

(c) Transpose

Figure 3.4: Network energy saving as a function of injection rate with synthetic traffic. Results are compared to network with 4-VC packet-switched routers.

VC tuning can reduce energy consumption without affecting the throughput significantly. By selecting $Threshold_{High}$ and $Threshold_{Low}$ properly based on network throughput characteristics, sufficient VCs can be guaranteed to prevent saturation. *Hybrid-TDM-VCt* outperforms *Hybrid-TDM-VC4* in terms of energy saving. However, as injection ratio increases, the gap between the two reduces. This is due to the fact that when more flits are injected, VC buffers become busier. More VCs have to be activated to satisfy the throughput requirement, resulting in fewer opportunities for aggressive VC power gating. Overall, *Hybrid-TDM-VCt* achieved additional energy saving over *Hybrid-TDM-VC4* by 2.4%-10.9% under uniform random traffic; 2.6%-10.0% under tornado traffic; and 4.1%-9.7% under transpose traffic.



(a) Max Throughput Improvement

(b) Network Energy Saving

Figure 3.5: Throughput improvement and network energy saving of *Hybrid-TDM-VCt* in larger networks. Energy result is sampled at 75% capacity before *Packet-VC4* saturates.

3.3.4 Scalability

To study the scalability of the proposed NoC, we scale the network size from 64 nodes (8-by-8) to 256 nodes (16-by-16), and evaluate the maximum throughput improvement and energy savings of *Hybrid-TDM-VCt* compared to *Packet-VC4*. We also increase the slot table size to 256 for the larger network because there are more source-destination pairs. Figure 3.5(a) and Figure 3.5(b) show the throughput improvement and energy saving, respectively. For most workloads, TDM-based hybrid-switched network can achieve the same throughput improvement and energy saving as the network scales in size. However, for the uniform random workload, the performance and energy benefit from hybrid switching is relatively low for a small network, and the benefit become negligible as network size increases. This is because the number of frequent communication pairs increases quadratically as the network size grows. Moreover, communication pairs in uniform random traffic are distributed across the network with equal packet injection possibilities. Therefore, only a small fraction of the traffic benefit from circuit switching given limited size slot tables. However, we expect realistic workload to exhibits some form of regularity and can benefit from hybrid switching.

3.4 Evaluation on Realistic Workload

In this section, we present a detailed evaluation of the proposed hybrid-switched network using realistic workloads in two heterogeneous systems, i.e., interleaving system (Section 3.4.2) and clustered system (Section 3.4.3). These systems exhibit different heterogeneities in terms of traffic pattern: in the interleaving system, heterogeneous traffic come from different nodes; while in the clustered system, heterogeneous traffic is generated by each individual node. We aim to show the flexibility of the proposed network in handling different heterogeneous traffic patterns.

Realistic workloads are essential for providing insight into the behavior of the proposed hybrid-switched NoC since they exhibit dramatically different traffic patterns compared to synthetic traffic. Furthermore, realistic workloads are *self-throttling*: if the buffers of a processor are full, the processor slows down packet injection until the congestion is mitigated [24]. In other words, the network almost always operate below saturation with realistic workloads.

3.4.1 Evaluation Infrastructure

We build a heterogeneous multicore simulator that integrates both superscalar-based CPU cores and data-parallel accelerators. CPU cores and the memory hierarchy is modeled after the Simics-based [39] GEMS [40] simulator. The accelerators are modeled with GPGPU-Sim [41]. The system configuration can be found in Table 3.2.

Application Workload

We use applications from the SPEC OMP 2001 [42] benchmark suites as CPU workloads, and applications from [41] and Rodinia [43] benchmark suites to simulate accelerator workloads. We consider 8 correctly compiled CPU benchmarks and 7 GPU benchmarks, where CPU benchmarks include: Ammp, Applu, Art, Equake, Gafort, Mgrid, Swim, and Wupwise; and GPU benchmarks include: Blackscholes, Lps, Lib, Nn, Hotspot,

CPU Configuration	
Processor	Four-way out-of-order, 6 integer FUs, 4 floating point FUs, 128-entry ROB
L1 Cache	Split private I/D caches, each 64KB, 2-way set associative, 64B block size, 1-cycle access latency
L2 Cache	16M banked, shared distributed, 4-way set associative, 64B block size, 8-cycle access latency
Accelerator Configuration	
Accelerator	32-wide SIMD pipeline, 1024 threads, 32KB shared memory
Memory Configuration	
Memory	4GB DRAM, 200 cycle access latency, 4 memory controllers

Table 3.2: Baseline System Configuration for realistic workload evaluation.

PATHFINDER and STO. Heterogeneous CPU-GPU workload is created by executing one copy of the multi-threaded CPU benchmark on multiple CPU cores and one GPU kernel across all accelerator cores. In each simulation, we sample a period of execution that corresponds to 500 million CPU instructions. We enumerate all possible combinations of the CPU and GPU workloads and evaluate 56 workload mixes.

Circuit Switching Decisions

To achieve optimal utilization of the network and minimize network energy consumption, we packet switch all CPU traffic while hybrid switch only GPU messages, since GPU traffic has a higher throughput requirement compared to CPU traffic. However, not all GPU messages are circuit-switched. A GPU message is considered to be circuit-switched only when no performance penalty is caused. Circuit switching a message can potentially increase or decrease transmission latency. While in GPUs, a delayed message may not necessarily cause performance degradation. The number of available warps in an SM can be used as an indicator to imply whether circuit switching a message causes performance penalty. In our work, we estimate the GPU message slack by referring to the number of

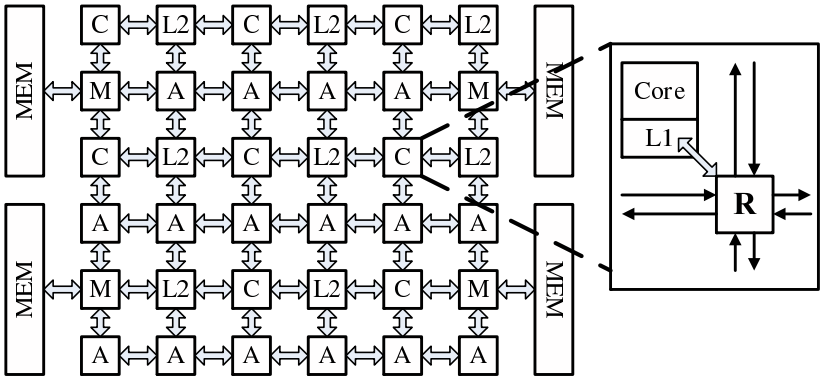


Figure 3.6: Interleaving multicore system.

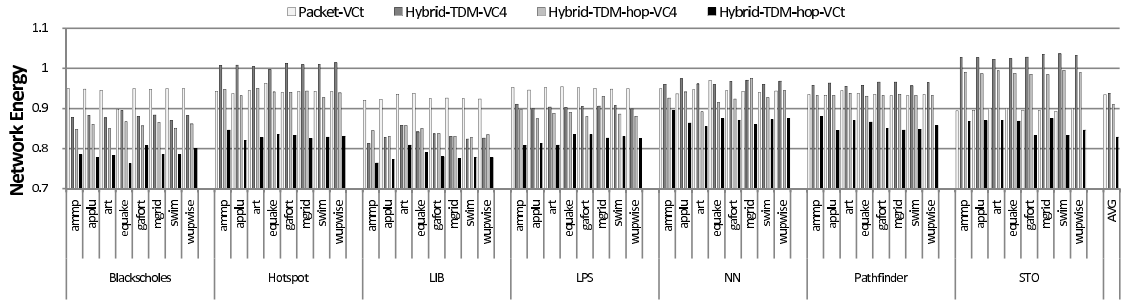
available warps. If the slack is greater than the overall circuit-switched transmission latency, we deliver the message through circuit-switched network. More sophisticated decision process can lead to better performance, however, a detailed investigation of the policy it is beyond the scope of this paper.

3.4.2 Interleaving System

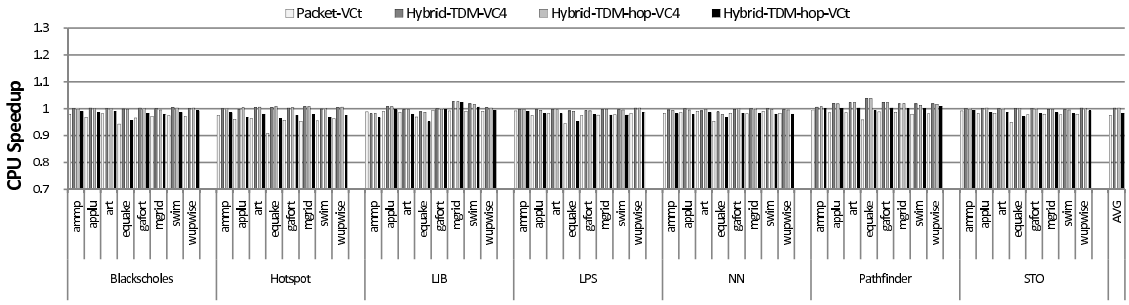
Figure 3.6 shows a 36-tile system connected with routers to a 6-by-6 mesh network, referred to as interleaving topology. A tile consisting of a CPU core as well as an L1 cache is denoted by *C*. A tile that consists of a bank of shared L2 cache is denoted by *L2*, and a tile in which an accelerator resides is denoted by *A*. Off-chip memories are connected via memory controllers labeled with *M*. Each tile is equipped with a hybrid-switched router *R*. Heterogeneous traffic is generated from different nodes, and such structure can be easily extended to a larger system with more tiles.

Overall Energy Efficiency

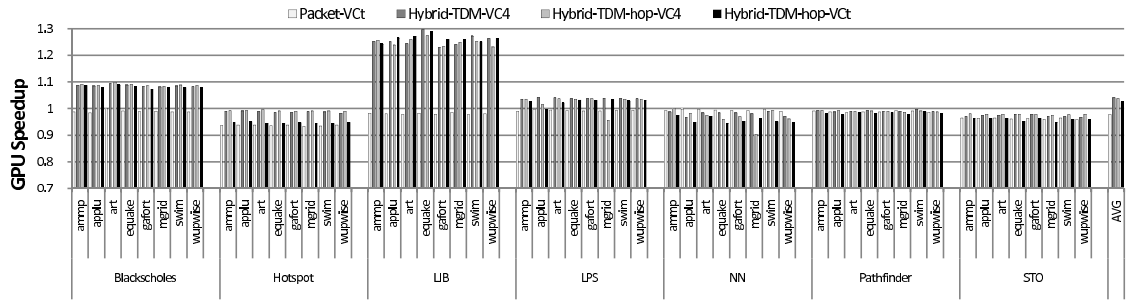
Figure 3.7 show the network energy saving, CPU application speedup, and GPU application speedup, normalized to the baseline 4-VC packet-switched network. X-axis is the workload mix, grouped by GPU benchmarks. The last set of bars *AVG* is the geometric mean across



(a) Network Energy



(b) CPU Performance



(c) GPU Performance

Figure 3.7: Network energy and performance results of hybrid-switched network. All results are normalized to network with 4-VC packet-switched network, which is not shown.

all workload mixes.

Observing from Figure 3.7(a), the proposed hybrid-switched NoC reduces the network energy consumption significantly. Different CPU workloads have minimal impact on network energy saving, because CPU packets only take up a small portion of the entire on-chip

traffic, while the variation in energy saving corresponds to GPU workload. On average, the basic hybrid-switched scheme reduces network energy consumption by 6.3%. With path sharing and aggressive VC power-gating techniques deployed, 9.0% and 17.1% of energy saving is achieved, respectively. In particular, up to 23.8% of energy saving is achieved for BLACKSCHOLES. However, for STO, more energy dissipation is required for *Hybrid-TDM-VC4*. This is because STO has relatively low throughput requirement with only a small amount of packets sent through circuit-switched network, thus dynamic energy saving is insignificant. Moreover, adding slot tables into routers introduces extra static energy consumption. Therefore, applications benefit from hybrid-switched router only when the reduction of dynamic energy offsets the increase of static energy overhead. Although the basic scheme is not efficient for applications with low injection rate, circuit-switched path sharing mechanism allows more packets to be sent through circuit-switched network with smaller slot tables; furthermore, aggressive VC power-gating technique turns off inactive buffers so that considerable amount of static energy consumption is reduced. Overall, by applying aggressive optimizations, the energy saving in STO is still comparable with other applications.

CPU application performance is presented in Figure 3.7(b). CPU performance is hardly affected for most applications. This is mainly because CPU messages are sent through packet-switched network, and are merely delayed slightly when competing with circuit-switched flits in the router. Since not all CPU messages are critical [44], such delay affects the performance trivially. GPU performance varies as shown in Figure 3.7(c). The worse case comes from STO, where all circuit-switched network schemes suffer from performance degradation by over 2%.

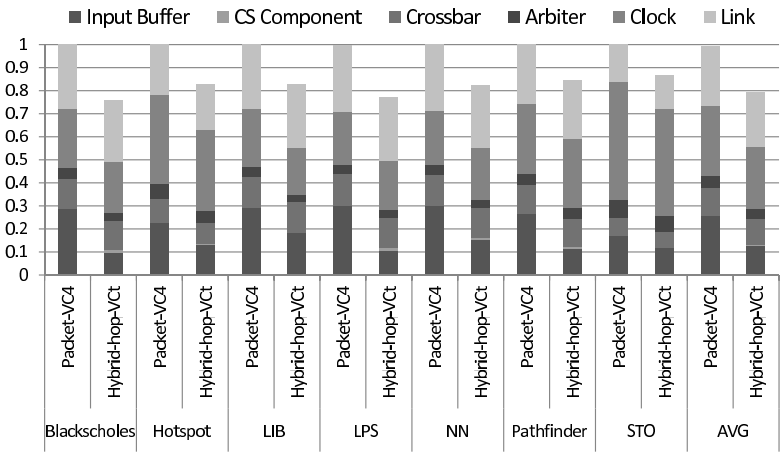
Overall, the proposed hybrid-switched network is able to reduce the network energy consumption by 17.1% with only 1.6% of CPU performance degradation and 2.6% of GPU performance improvement, on average. Therefore, the hybrid-switched network is energy efficient in the evaluated heterogeneous system.

Dynamic energy saving: Figure 3.8(a) demonstrates the breakdown of dynamic energy dissipation, including input buffers, circuit-switching (CS) components, crossbars, VC/SW arbiters, clock, and link energy. All hardware introduced for circuit-switching are referred to as CS Component in this figure. Due to the avoidance of buffer read/write, hybrid-switched network reduces the buffer energy by 51.3%, on average. The overhead caused by circuit switching is 0.6%, on average (up to 1.2% for LPS). Both buffer energy saving and circuit-switching overhead are dependent on the percentage of traffic that are sent through the circuit-switched network, as shown in Table 3.3. Higher circuit-switched percent correspond to more energy saving while larger circuit-switching overhead. Savings from crossbars, links and arbiters are negligible. This is because both circuit- and packet-switched flits have to pass through crossbars and link wires; and arbiters only correspond to a small portion of dynamic energy consumption. Overall, 20.8% of dynamic energy reduction is achieved by deploying hybrid-switched routers, with BLACKSCHOLES benefits the most (24.1%).

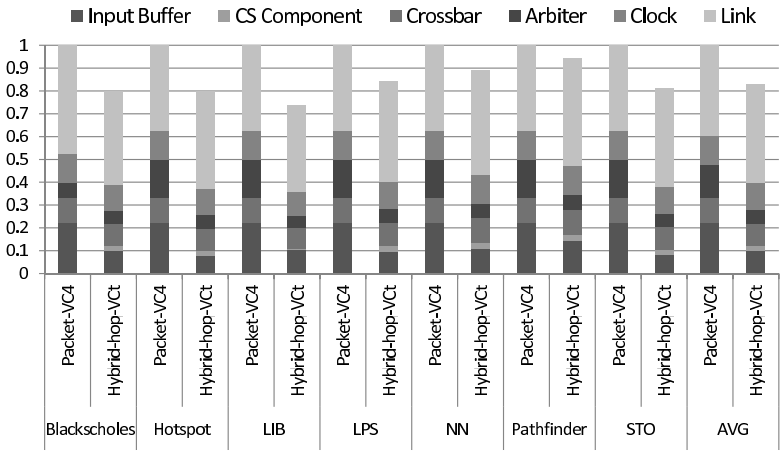
GPU benchmarks	Injection ratio (flits/node/cycle)	Circuit-switched flits percent (%)
BLACKSCHOLES	0.18	55.7
HOTSPOT	0.09	29.1
LIB	0.20	34.4
LPS	0.20	55.0
NN	0.18	38.9
PATHFINDER	0.13	49.1
STO	0.05	18.5

Table 3.3: GPU traffic injection rate and percentage of flits that are circuit-switched for *Hybrid-TDM-VC4*.

Static energy saving: Dynamic energy is expected to reduce as technology advances due to smaller device size, but static energy is expected to increase. Our proposed router



(a) Dynamic energy



(b) Static energy

Figure 3.8: Detailed network energy breakdown. Results are grouped by GPU benchmarks. Each bar is an average over all CPU applications.

exploits the opportunities of power-gating under utilized components as discussed in Section 3.2, so that static energy dissipation can be reduced as well. Dynamic voltage-and-frequency scaling (DVFS) can be applied orthogonally to our technique to mitigate clock energy largely, but is beyond the scope of this paper. Figure 3.8(b) shows static energy breakdown. On average, 17.3% of static energy saving is achieved with 2.1% overhead

caused by circuit-switched components. As expected, all the static energy saving comes from input buffers. Introducing slot tables enables bandwidth sharing, but brings in additional overhead. The only exception happens in LIB, where insignificant static energy overhead is observed. LIB has fewer communication pairs compared to other GPU applications, hence smaller slot tables are sufficient to hold all circuit-switched paths. It is worth pointing out that compared to packet-switched network with VC power-gating (not shown), 6.8% static energy saving is achieved, proving that hybrid-switched network can alleviate buffer pressure and enable further VC power-gating.

Performance Analysis

Hybrid-switched network brings in minimal performance penalty even with aggressive energy saving techniques deployed. As shown in Figure 3.7(b), overall CPU performance impact is negligible. Reducing the number of VCs degrades the network throughput, which slightly affects the CPU application performance by 1.6%. For GPU performance in Figure 3.7(c), NN, PATHFINDER, and STO suffer from slight performance degradation while BLACKSCHOLES and LIB achieve over 9% of speedup. Performance penalty is mainly due to the delay of critical messages. Circuit-switched network might affect message delay in two ways: Firstly, buffering circuit-switched messages at the source increases the queueing delay; Secondly, when contentions occur in routers, packet-switched messages which have a lower priority always give way to circuit-switched messages, therefore the network latency of packet-switched messages increases. If the transmission of critical message is delayed, system performance will be harmed. Recall that in Section 3.4.1 only simple policy is used for deciding which messages should be circuit-switched, accurate performance monitors can be referred in order to avoid performance penalty.

Hybrid-switched network also improves the performance for applications such as BLACKSCHOLES and LIB. This is mainly because memory requests reach their destination faster in

a hybrid-switched network. For example, in BLACKSCHOLES, large number of data write-backs can be delayed without affecting the overall performance. Therefore, sending these write-back data through circuit-switched network alleviates the pressure on packet-switched network, which in turn reduces the contention in packet-switched network. Execution stall in GPU can therefore be prevented since requested data are sent back faster.

Effectiveness of Circuit-switched Path Sharing

Compared to *Hybrid-TDM-VC4*, network energy is further reduced by 2.8% in *Hybrid-TDM-hop-VC4*, on average. While the performance impact on both CPU and GPU applications is negligible. By allowing circuit-switched path sharing, a packet can be sent through circuit-switched network without reserving dedicated connections, given that its route is partially overlapped with existing circuits. Better energy saving can therefore be achieved. Especially when traffic injection rate is not high, most of the circuit-switched connections are not fully utilized. Path sharing enables smaller slot tables being used. Moreover, when slot tables become smaller, the waiting time before a circuit-switched flit can be sent is reduced. As a result, more flits can be considered to be circuit-switched as performance constraint is easier to satisfy. Meanwhile, aggressive VC power-gating can potentially benefit from circuit-switched path sharing as VC buffers becomes less busy. Therefore, path sharing promotes aggressive network energy saving and is a critical design in the hybrid-switched network.

Effectiveness of Aggressive VC Power-gating

The aggressive VC power-gating policy described in Section 3.2 can be applied to both packet- and hybrid-switched NoC. In a heterogeneous system with various on-chip traffic patterns, sufficient VCs are required to satisfy the throughput requirement while guaranteeing the desired latency. Energy and performance trade-off can be accomplished by reducing the number of VCs in a router. Comparing between the first and last set of

bars in Figure 3.7, i.e., *Packet-VCt* and *Hybrid-TDM-hop-VCt*, the hybrid-switched network further reduces the network energy consumption by 10% on average, while providing better speedup. The energy savings come from 1) dynamic energy reduction due to circuit-switching large number of flits, and 2) static energy reduction due to the fact that input buffer pressure is alleviated by circuit-switched network, more buffers can be power-gated. For some applications, performance degradation is seen for both packet- and hybrid-switched network, because the network fails to provide sufficient bandwidth after reducing VC numbers.

Overall, aggressive VC power-gating policy reduces energy consumption significantly in applications with moderate on-chip communication requirement. We believe activating and deactivating VCs based on more accurate metrics, for example, packet latency, will ensure better performance.

3.4.3 Clustered System

To demonstrate the applicability of the hybrid-switching on other network, we evaluate a clustered system, where a CPU core, an L2 cache and two data-parallel accelerators are integrated in a cluster with a router, as shown in Figure 3.9. The system has 16 clusters forming a 4x4 mesh. Heterogeneous traffic is generated by each individual cluster. Figure 3.10 shows the network energy reduction and system performance improvement achieved by the hybrid-switched network compared to a packet-switched network.

In short, the clustered system is able to demonstrate the same level of improvement in energy efficiency as the interleaving system. The clustered system generates considerably more traffic compared to the interleave systems. Consequently, the clustered system is more congested, and thus opportunities for dynamic VC tuning is reduced. However, the hybrid-switched routers are able to capture more frequent communication pairs and establish more circuit-switched paths. Overall, in this clustered system, hybrid-switched network is able to reduce the network energy consumption by 14.5% (up to 20%). Moreover,

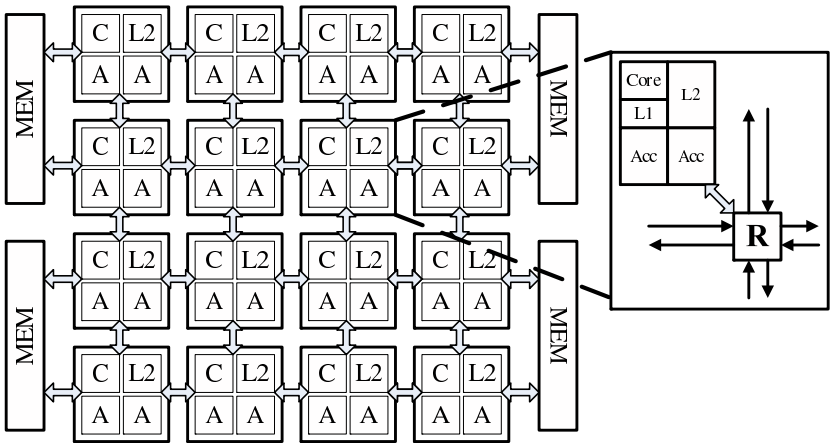
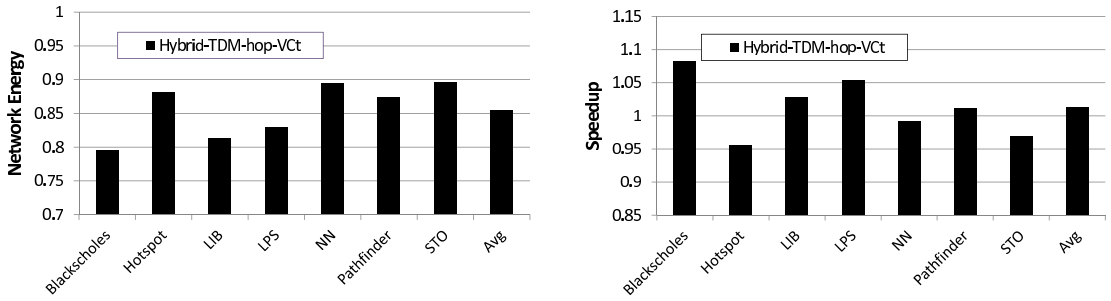


Figure 3.9: Clustered multicore system.



(a) Network Energy

(b) Speedup

Figure 3.10: Network energy and performance for clustered system.

compared to packet-switched network, the overall performance impact is minimal because hybrid-switched network does not incur significant message transmission delay.

To summarize, we believe that hybrid-switching is able to improve energy efficiency for diverse NoC systems with adequate switching policies that are capable of aggressively exploiting NoC traffic regularity while judiciously trading-off energy-saving opportunities with performance impact.

3.5 Summary

In this chapter, we optimize the TDM-based hybrid-switched NoC for better energy efficiency, and evaluated the proposed design with both synthetic and realistic workloads. Evaluation results show that, hybrid-switched NoC enables efficient data movement in heterogeneous multicore systems. In particular, with hybrid-switched NoC deployed, the network energy consumption is reduced by as much as 24% and system performance is improved by as much as 12%, compared to the network using canonical packet-switched routers. With an adequate policy for setting up circuit-switched paths, as well as selecting and forwarding messages that can benefit from circuit switching, TDM-based hybrid-switched networks can efficiently reduce the network energy consumption, and even improve overall system performance. Moreover, optimizations such as circuit-switched path sharing and VC power gating are essential to achieve further energy savings. In conclusion, in heterogeneous multicore and manycore systems, we believe that TDM-based hybrid-switched NoC provides a viable alternative for achieving low-latency and high-throughput with a stringent power budget.

Chapter 4

In-Network Memory Access Ordering

The integration of CPUs and GPU-like data parallel accelerators onto the same die has shown its potential for improving performance and energy efficiency. With the support of heterogeneous computing, a logically tight integration of different types of computation units is enabled, which allows heterogeneous cores to have a unified address space. In the context of heterogeneous systems, choosing of memory consistency model is critical since a consistency model defines how memory accesses in a program will appear to the programmer [45]. Different memory consistency models make different trade-offs among performance, programmability, and hardware complexity.

Consider the example shown in Figure 4.1 involving two cores (C0 and C1) and two shared variables (x and y) belonging to different memory divisions. C0 first writes to x, and then writes to y; while C1 first reads from y, and then from x. All the variables have an initial value of 0. Due to the uncertainty of the Network-on-Chip (NoC) delay, these read and write requests might arrive at the memory controllers in any order, even if they are issued in program order. With the order suggested by left side of Figure 4.1, the value of (x, y) read by C1 is (0, 1). Depending on the memory consistency model the underlying

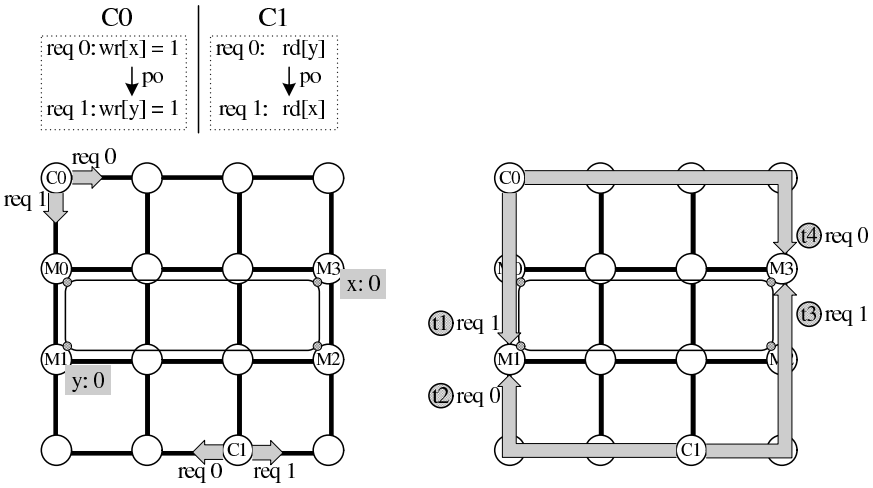


Figure 4.1: Request transmission in unordered mesh network.

machine assumes, this result can be illegal. In order to guarantee a correct result, the core has to maintain the order of read/write requests. Such end-point ordering can be very expensive and lead to significant performance penalty. Trade-offs can be made between performance and programmability–more relaxed consistency models allow reordering of read/write requests while programmers are responsible for adding fence instructions when applying order. However, if a program is not properly written, the memory accesses may be performed in an order different from what the programmer expects, resulting in unpredictable program behavior. In the era of heterogeneous multicore architecture, this will lead to significant debugging effort given the mass number of threads supported by the systems.

Enforcing a programmer-friendly strong memory consistency model while maximizing memory-level parallelism is challenging, especially in heterogeneous systems where data-parallel cores generate significant amount of outstanding memory requests. End-point ordering at cores is expensive since it prohibits a number of architecture optimizations. However, if correct order is provided in the interconnection network during request transmission, we can potentially improve the performance by parallelizing memory requests. In

multicore and many-core systems, packet switching mesh-based NoC is the de facto choice due to its flexibility and scalability [46, 8, 47]. Packet switching networks do not inherently guarantee message ordering, and could result in uncertainties during message transmission. Circuit switching networks, on the other hand, provide both energy efficiency and message transmission ordering [48, 49], therefore can be used as an infrastructure to preserve order. The combination of both packet and circuit switching provides us with an efficient on-chip communication platform which also enforces memory access order within the network.

In this chapter, we propose a memory access ordering network for directory-based heterogeneous multicore systems to maximize memory-level parallelism while still maintain a stronger memory consistency model such as total store order. The underlying hybrid switching network, which supports both packet and circuit switching in the same fabric, handles heterogeneous traffic efficiently; the global order of memory requests is maintained with the help of a light-weight token ring network.

4.1 Memory Access Ordering Network

In this section, we first give an overview of the proposed network, and then describe the microarchitecure design in detail.

4.1.1 Network Overview

In high level, the ordering network consists of three components: 1) a mesh-based hybrid switching data network that supports both packet and circuit switching; 2) a light-weight token ring network passing around a token which guarantees memory access ordering; and 3) network interface (NI) controllers that examine and update the token, as well as store and manage memory access requests.

Hybrid switching data network. A hybrid switching NoC, built upon a packet

switching NoC, allows traffic be sent in both packet and circuit switching manners. Dedicated circuit switching paths are established between communicating pairs. Once a connection is set up, messages from the same source to the same destination can repeatedly reuse the path without destroying the connection. Compared to packet switching, circuit switching data paths have two major advantages. First and foremost, the communication latency is predictable, because the length of each path and message traversal latency are determined. Secondly, buffering and routing are not required, therefore circuit switching has lower per-hop delay as well as lower energy overhead. Time-division multiplexed hybrid switching NoC allows fine-grained sharing between packet and circuit switching data paths [48], which is potentially a good candidate for ordering network messages. In our proposed NoC, we enforce messages' order by routing them through circuit switching, while the rest of the traffic is routed through packet switching without preserving any order.

Token ring network. This light-weight token network connects ordering points, defined as the nodes in which memory access order is maintained, into a ring. Although memory requests arrive at the ordering points in-order, they are not guaranteed to be processed in-order globally. Token is used for maintaining message order globally, and it is passed around the token network from one ordering point to another. The token indicates the ids of the requests that are being served. Each core generates its own request id independently. Therefore, the token contains a collection of ids, each corresponding to an individual core. The network interface controllers at the ordering points modify the token when all requests with the same id from the same core are completed.

Network interface. The NI is responsible for receiving and examining the token at each ordering point, making sure that the memory component only gets requests with ids smaller than the id indicated by the token. Once the memory component finishes processing the requests, it notifies the NI and the later will set the corresponding bits inside the token. Also by examining the token, the NI increments the request id in the token when it detects that all memory controllers have finished their work. The updated

token is then injected back to the token network. Notice that a single memory instruction could result in multiple memory requests sending to multiple destinations, it is possible that more than one memory controllers receive requests from the same core with identical request id. To guarantee memory access requests are issued to the memory controllers in order, the NIs are also responsible for storing and managing the requests. Requests from the same core are grouped and stored in a cache-like array structure, from which requests with smaller ids are issued prior to those with larger ids. Details of the NI microarchitecture are presented in Section 4.1.4.

4.1.2 Data Network Microarchitecure

In a heterogeneous multicore system, the NoC must be able to handle both latency-sensitive traffic as well as throughput-intensive streaming traffic. Hybrid switching network offers the flexibility of handing both types of traffic efficiently [49], hence we apply the hybrid switching design in our baseline network.

TDM-based hybrid switching router. Figure 4.2 shows the microarchitecture of a hybrid switching router. With the support of *slot tables*, the hybrid switching router allows both packet and circuit switching traffic to share the same communication fabric through time-division multiplexing (TDM) [48]. A slot table has limited number of entries; and it is looked up by the allocation logics every cycle in modulo-fashion. A slot table entry indicates whether a particular cycle is reserved for a circuit switching data path (when V bit is set), or packet switching (when V bit is 0); it also keeps track of the input-output port mapping for a particular circuit switching path (indicated by OUT field). A flit is injected to either the packet or the circuit switching data path from the NI, depending on whether a circuit switching connection is already established from its source to its destination. Once the flit is on circuit switching path, it can be forwarded towards its destination without buffering and routing at each router.

For example, in Figure 4.2, slot table entry 1 for input port 1 is valid, meaning that a

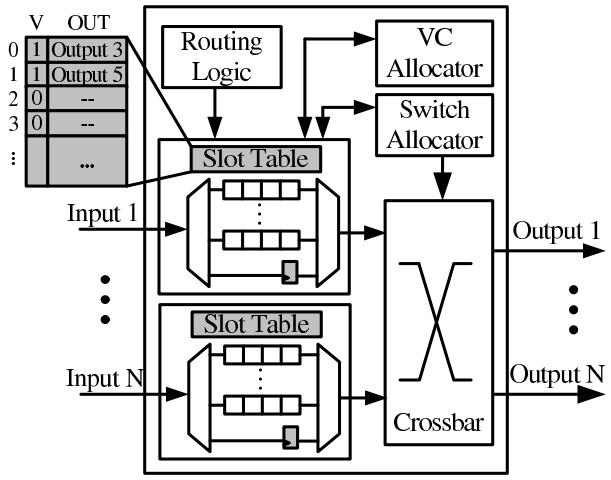


Figure 4.2: Data network router architecture.

circuit switching connection is established for this cycle. According to *OUT* field, output port 5 is reserved for this connection. Therefore, if a circuit switching flit arrives in cycle $1+n\times S$ (S is the size of the slot table, and n is some arbitrary number), it is then directly forwarded to output port 5 in the next cycle without performing route computation and arbitration. Assume link traversal takes 1 cycle, the flit will arrive at the downstream router in 2 cycles. Since there is no buffering for circuit switching flits, slot table entry 3 in the downstream router must be configured properly beforehand to accept and forward this flit. In TDM-based hybrid switching router, if a time-slot is reserved for circuit switching but circuit switching flit does not present in that cycle, a packet switching flit is allowed to use this time slot. Such design greatly improves link utilization.

Slot table configuration. Slot tables are a critical structure to ensure the correctness of flit forwarding, and they can be configured either dynamically in run-time or statically in design-time. Dynamic configuration captures run-time traffic patterns and adopts to traffic behavior changes. However, a dynamic approach usually requires a separate setup network which adds additional hardware overhead to the NoC. Furthermore, the lack of global decision makes dynamic slot table configuration less efficient–conflict happens when

two circuit switching paths try to reserve the same slot table entry, which results in a setup failure and possibly infinite retries. Therefore, circuit switching path reservation in a dynamic approach is not guaranteed to be successful. In contrast, a static approach assigns slot table entries in design-time without introducing additional hardware overhead. However, the slot allocation is usually fixed therefore lacks flexibility. In out proposed network, we use static approach because: 1) The main purpose of using circuit switching network is to guarantee end-to-end message ordering, flexibility and adaptiveness is less important for our design; 2) Our proposed NoC requires circuit switching connections for ALL source to destination communication pairs, which cannot be satisfied by dynamic approaches; and 3) A static approach avoids setup latency, which avoids unnecessary delays for messages that preserve ordering.

We present a slot allocation algorithm in Algorithm 1. The first input parameter $comm\_pairs$ is the number of communication pairs for which circuit switching connections are required. For each communication pair, the path (essentially a list of links that messages traverse through) from the source to the destination, is stored in the second input parameter $path[comm\_pairs]$. The output $starting\_slot\_id[comm\_pairs]$ is the starting time slot entry assigned to each communication pair at the source router.

Line 1 initializes a global slot table, keeping track of time slot assignment for all communication pairs. The g_slot_table is a 2D array, which has as many rows as there are in a router's slot table. The key idea behind this algorithm is the following. For each $path$, place its $link_id$s one after another to the corresponding rows inside g_slot_table. Globally, placing link id m in row n means that link m is assigned to a circuit switching connection in cycle n. Slot n should never be assigned to the same link m for another path, because having the same link id appear in the same cycle indicates a conflict in circuit switching set up. If a conflict occurs between two paths, the latter path has to revert all its slot allocations and choose a new row from g_slot_table to start again, until all its links are placed without conflict. Every path starts placing its first link from the first row, as shown

Algorithm 1 Static slot allocation algorithm

Input: comm_pairs, path[comm_pairs]
Output: starting_slot_id[comm_pairs]

```
1:  g_slot_table[][] ← {}
2:  starting_slot_id[comm_pairs] ← {0}
3:  for i = 1 to comm_pairs do
4:     starting_slot = 0;
5:     succeed = false;
6:     while (!succeed) do
7:        slot_id = starting_slot;
8:        for each link_id in path[i] do
9:           if (g_slot_table[slot_id].find(link_id)) then
10:             //undo all changes for this path
11:             roll_back();
12:             succeed = false;
13:             starting_slot++;
14:             break;
15:          else
16:             slot_table[slot_id].push_back(link_id);
17:             succeed = true;
18:             slot_id++;
19:          end if
20:       end for
21:    end while
22:    starting_slot_id[i] = starting_slot;
23: end for
```

in Line 4. Line 8-20 attempt to place all links from a path into *g_slot_table*. When a conflict happens, the failed path roll back its previous allocation and start re-try from a new *starting_slot*, as shown in Line 9-14. If no conflict happens, the *link_id* is simply pushed back to the corresponding row in *g_slot_table*, as stated in Line 15-18. In Line 22, the *starting_slot* is stored when slot allocation is completed for a communication pair.

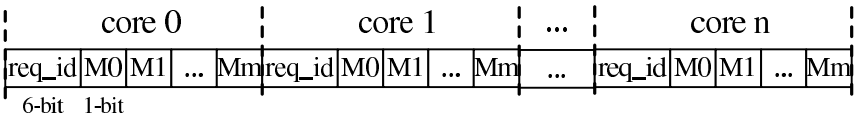
Given the starting time slot entry at the source, slot table allocation in each individual router for a circuit switching path can be easily calculated.

4.1.3 Token Network Microarchitecure

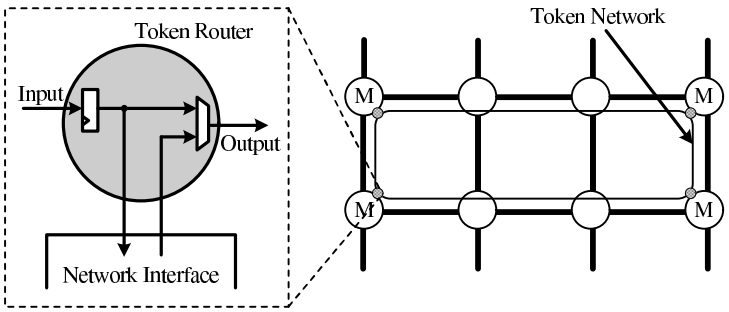
Token network is used to preserve global memory access order among multiple ordering points. Ordering points are shared but distributed structures where memory accesses are handles, for example, directories or memory controllers are considers as ordering points. Only a token is transmitted in this dedicated network.

Token format. The token format is shown in Figure 4.3(a). For a system with n cores and m ordering points, the token contains n fields. Each field corresponds to an individual core. Inside each field, *req_id* indicates the id of the memory request that is currently being processed by the ordering point. Followed by m valid bits, i.e. M_0, M_1, ..., M_m, each represents an ordering points. A valid bit is set when an ordering point finishes serving the request with the same *req_id* as indicated by the token. When a core executes a load/store instruction and generates new memory access request(s), it increments its local request id by 1 and embeds this id with the request packet(s), which is(are) then sent to the ordering point(s). All memory access requests will be buffered at the destination network interfaces. Then, the network interface at each ordering point examines the token, and forwards requests to memory components when certain constrains are satisfied (details in Section 4.1.4).

Token network. The token network is a unidirectional ring with extremely simple routers at each ordering point. As shown in Figure 4.3(b), a token router receives the token from upstream router, forwards the token either to the network interface or to the downstream router, depending on whether the network interface requires to examine the token. If the network interface decides to accept the token, the token is removed from the token network. After processing, the network interface updates the token and injects it back into the token network. In any given time, there is at most one token in the token network.



(a) Token format



(b) Token network and router

Figure 4.3: Token network microarchitecure.

4.1.4 Network Interface Microarchitecure

Working in tandem with the token network, network interface plays an important role in maintaining memory access order. The proposed NI contains un-ordered queue, ordered queue, re-order arrays, token buffer, and token examination logic, usages of which are described below.

Sending Process. At the source NI, messages that require to be ordered are buffered in *OD Queue* and sent through circuit switching; while the rest are buffered in *UO Queue* and send through packet switching, as shown in Figure 4.4. Ordered requests are injected into the data network when the reserved time slots are ready while unordered requests are sent when the output port is free.

Receiving Process. Upon receiving, unordered requests are buffered in a first-in-first-out manner at the destination NI. Ordered requests are stored in different *Re-order Arrays*, based on their source nodes, i.e., ordered requests from core n are stored in the re-order array reserved for core n. An ordered request is retrieved from the re-order array

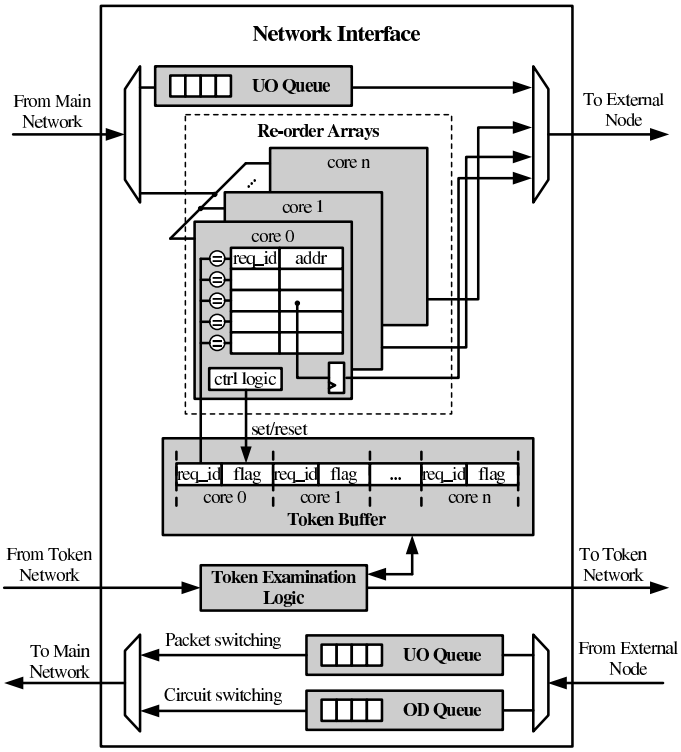


Figure 4.4: Network interface microarchitecture.

by referring to its *req_id*, indicated by the *Token Buffer*. Token buffer is a structure that interacts with the token. Like the token, in an n-core system, a token buffer has n fields; inside each field, *req_id* indicates the memory request that is being processed. When a request with a particular *req_id* is served, $flag$ in the corresponding field in the token buffer is set to 1 by the control logic.

Token Update Process. The req_id inside a token buffer is simple a copy from the token. When all valid bits (M_0 through M_n) of a particular core inside the token are 1s, the req_id in the token for that core is increased by 1, and all valid bits are reset. This update process is performed by *Token Examination Logic*. The token examination logic is also responsible for updating the corresponding fields in the token buffer, i.e., when the logic sees the token is carrying a larger req_id than in the token buffer, it updates the token

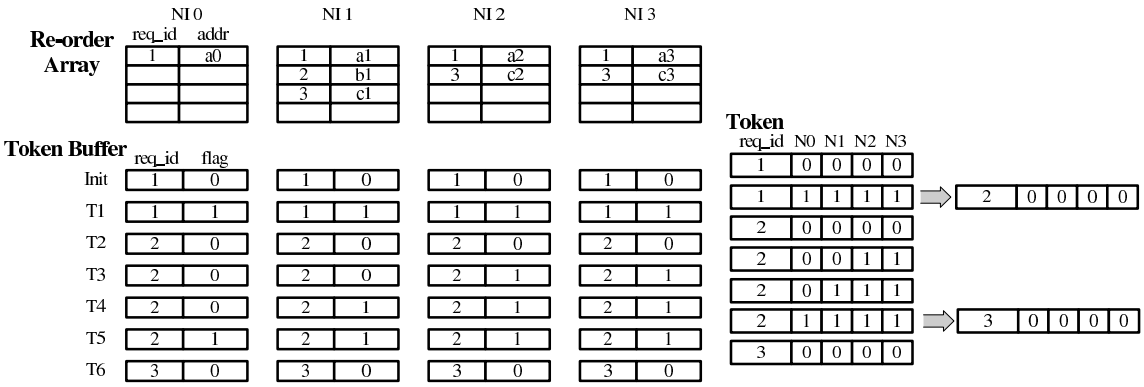


Figure 4.5: Token network walkthrough example.

buffer's req_id and resets the flag. Since the ordered requests are sent through circuit switching at the source, they are guaranteed to arrive in-order at a certain destination node. Therefore, a re-order array should see requests coming with incrementing req_id. This feature is important because it allows an NI to time-out when no further request is coming (more details in the walkthrough example below). If the req_id from the token buffer is smaller than the smallest req_id observed from the re-order array, the NI will simply skip processing this request by setting the flag to 1.

Walkthrough Example

Figure 4.5 demonstrates how the ordered network works. In this example, 4 NIs (NI_0-NI_4) are connected to the token network. For simplicity, we only show one re-order array in each NI, assuming only one core is sending requests. In NI_0, only 1 request is buffered in the re-order array, with req_id equals to 1 ($req1$). In NI_1, 3 requests $req1-3$ are waiting to be served. Both NI_2 and NI_3 have 2 pending requests.

1. Initially, for each token buffer, the req_id and $flag$ are set to 1 and 0, respectively. The req_id inside the token is set to 1, and the 4 valid bits representing 4 NIs are set to 0.

2. In T1, all NIs finish serving $req1$ and set the flag to 1 in their own token buffers. Meanwhile, the token is circling in the token network and examined by the NIs, until all its valid bits are 1s. Then the token's req_id becomes 2 and all the valid bits are reset to 0. This update process can only happen in one of the 4 NIs.

3. After the token has been updated, it continues circling in the token network and notifies the NIs to process $req2$, leading us to the status given in T2.

4. Notice $req2$ is only present in NI_1, meaning that the other NIs might skip this request. Both NI_2 and NI_3 see requests with larger req_id. Since it is guaranteed that requests are received in-order, NI_2 and NI_3 skip processing $req2$ by setting the flag of token buffer to 1, as shown in T3. Once the token reaches these NIs, the valid bits are set by token examine logic.

5. In T4, NI_1 sets its flag to 1 after serving $req2$. However, up till now, flag in NI_0 is still 0. Essentially, NI_0 is still waiting for $req2$ or a request with larger req_id. Remember the arrival of circuit switching messages are deterministic, the NI knows when a message from a particular node is to arrive by referring to the slot table. Therefore, after noticing some of the valid bits from the token are set, the waiting NI starts to time-out.

6. Eventually, in T5, NI_0 sets its flag and the token is updated correspondingly.

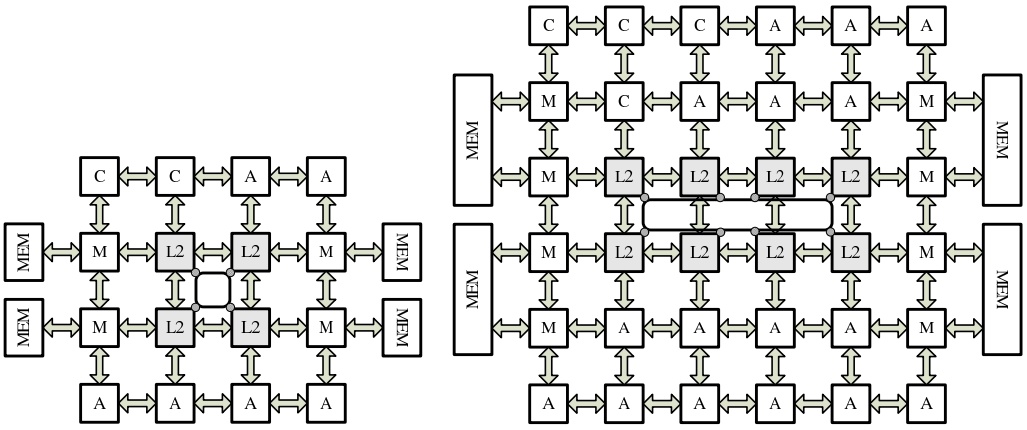
7. In T6, token buffers are updated and the above described process continues.

4.2 Evaluation

In this section we describe the simulation infrastructure and the workloads used to evaluate our ordering network design. Then we present the experimental results, followed by a discussion on the flexibility and limitation of the proposed network.

4.2.1 Evaluation Infrastructure

We use gem5-gpu [50] as our evaluation infrastructure. Simulator configuration can be found in Table 5.2. Figure 4.6 shows the topology of the evaluated systems. The 16-node (4-by-4 mesh) system contains 1 CPU core, 6 GPU accelerator cores, 4 shared L2 cache banks, and 4 memory controllers connected to off-chip memories. CPU and GPU accelerator cores share the same memory address space. MESI protocol is used to keep both CPU and GPU private L1 Data Caches coherent. Directories are located at L2 caches (shaded in figure), whose NIs are connected by the token network. Similarly, the 36-node (6-by-6 mesh) system has 8 directory nodes connected to the token network.



(a) 16-node System (b) 36-node System

Figure 4.6: Evaluated multicore system for evaluating ordered network.

We augment the baseline Garnet [14] network with a TDM-based circuit switching network [48] as well as a token ring network. Token width is dependent on the number of cores as well as directories. In the evaluated 36-node system, limited by the link width of the token network, token must be sent in multiple cycles. Therefore, the NI is responsible of breaking the token into multiple packets when sending and reassemble the token when receiving. Flow control in token network is not required because a token packet is guaranteed to be accepted by either the token router or the NI. The data network and token

network configuration are listed in Table 5.1.

CPU Configuration	
Processor	in-order, non-pipelined
L1 Cache	Split private I/D caches, each 64KB, 4-way set associative, 64B block size, 1-cycle access latency
Accelerator Configuration	
Accelerator	32-wide SIMD pipeline, 1024 threads, 32KB shared memory
L1 Cache	Split private I/D caches, each 16KB, 4-way set associative, 128B block size, 2-cycle access latency
Memory Configuration	
L2 Directory Cache	Shared banked, 2M/bank, 8-way set associative, 128B block size, 8-cycle access latency
Memory	4GB DRAM, 200 cycle access latency

Table 4.1: System configuration for evaluating ordered network.

We use CUDA applications from Rodinia [43] benchmark suites as accelerator workloads. Explicit memory copies are removed from the workloads, and pointers are used instead. The evaluated workloads include: *backprop*, *bfs*, *gaussian*, *hotspot*, *kmeans*, *nn*, *nw*, and *pathfinder*. We execute only one application each time on one CPU core, which then launches the kernel across all GPU cores.

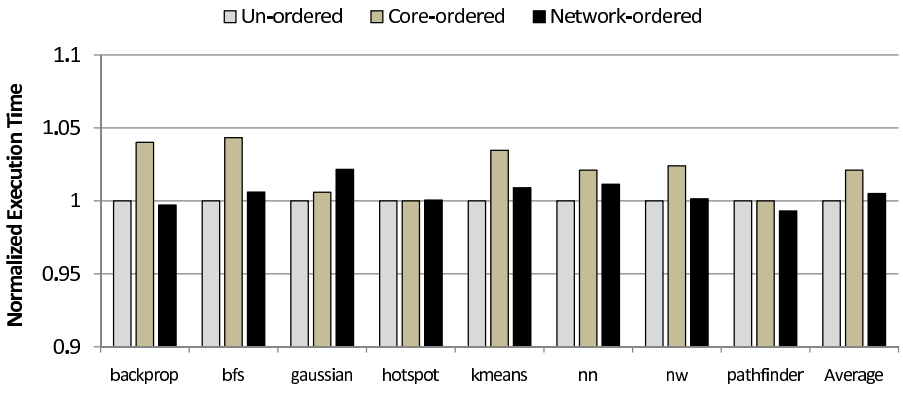
4.2.2 Circuit Switching Decision

Circuit switching all memory requests is unnecessary and is costly in terms of performance. For read/write requests accessing private data, or read requests accessing read-only shared data, sending them with un-ordered packet switching network is safe. However, if a write is accessing shared data, it has to be ordered properly. Prior work have proposed memory management unit and OS page protection mechanisms to classify memory blocks into

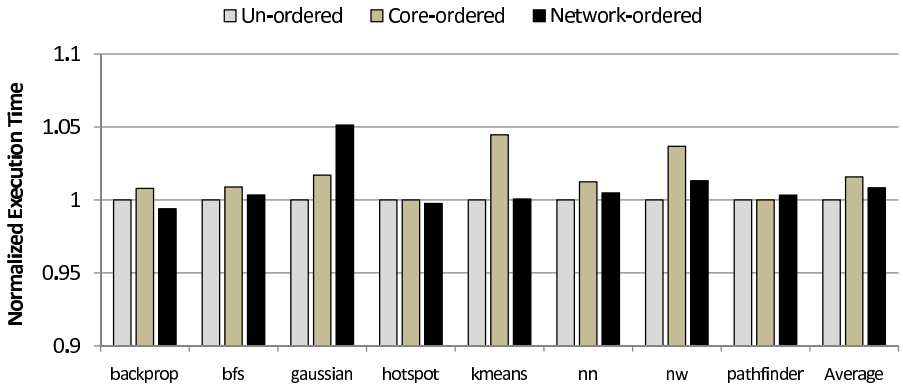
Data Network	
Topology	16/36-node, 2D-Mesh
Routing	Minimal Adaptive Routing
Channel Width	32 Bytes
Packet Size	4 flits (circuit switching packet) 5 flits (packet switching packet)
Slot Tables	16-node: 16-entry slot table 36-node: 32-entry slot table
Virtual Channels	4/port
Buffer size per VC	5 in depth
Token Network	
Token Width	16-node: 80 bits 36-node: 336 bits
Link Width	16 Bytes

Table 4.2: Router parameters for evaluating ordered network.

private and shared [51, 52, 53], so that safe access can be determined. In this work, we use a similar mechanism as proposed in [52] to decide whether to circuit switching a write request or not. In particular, we extend the page table entries with a core identifier (CID) field, a read-only (RO) bit, and a shared (SH) bit. Memory accesses to a page with RO bit unset and SH bit set need to be ordered. TLB entry is extended with a lock (LK) bit. When LK bit is set, all accesses must be sent through circuit switching to maintain their order. Upon a TLB miss, the TLB miss handler refers to the page table and checks the ownership (CID) of the page. If the page is owned by another core with RO bit unset, the TLB miss handler allocates a TLB entry and sets its LK bit. Meanwhile, if the SH bit is also unset, a notification is generated and sent to the owner of this page, which will then set the LK bit in the owner's TLB entry. Such notification is not required when TLB miss handler finds the SH bit set in the page table entry, since the combination of RO bit unset and SH bit set means all the sharers already have their LK bits set in TLB entries. Notice that in GPGPU applications, CPU initializes shared data before a kernel launch, and usually it does not access the data until kernel completion. Therefore, CPU should



(a) 16-node system



(b) 36-node system

Figure 4.7: Performance of ordered network. Results normalized to the baseline Un-ordered system

not update the CID field in page table entries during initialization. Once initialization is finished, a CPU TLB shootdown is required, such that later accesses to the shared data can be captured properly.

4.2.3 Experimental Results

For evaluation, we compare our proposed mechanism against 1) a baseline system, which uses only packet switching NoC and does not maintain any ordering at the core, referred to

as **Un-ordered**; and 2) a system using only packet switching NoC and maintains total store order (TSO) at each core, named as **Core-ordered**. TSO guarantees that the sequence in which store instructions appear in memory for a given processor is identical to the sequence in which they were issued by the processor [54]. To support TSO, we provide a FIFO write buffer for each core. The write buffer must be drained before each atomic instruction and memory fence instruction [55]. Our proposed work is referred to as **Network-ordered** in the evaluation.

Figure 4.7 shows the normalized execution time for 16-node system and 36-node system. In 16-node system, Network-ordered system degrades the performance by 0.5% on average, compared to Core-ordered system which incurs 2.1% performance penalty. In 36-node system, execution time is increased by 0.8% on average in Network-ordered system, in contrast to 1.6% in Core-ordered system. Overall, enforcing memory access order either at core or at network has insignificant performance impact to heterogeneous systems. This is mainly because the parallelism in GPGPU applications is capable of hiding memory access latency so that long access delay is tolerable. The second reason is that SIMD accelerators are simple in-order cores, which leads to fewer optimization opportunities in terms of re-ordering memory operations to begin with. Additionally, the using of synchronization in GPGPU applications further limits the memory level parallelism. Network-ordered system performs better than Core-ordered system because write requests from a single core are pipelined, in other words, the core does not need to stall and wait for each request to complete.

Throughput intensive applications such as gaussian, nn, and nw suffer performance degradation in Network-ordered system. Figure 4.8 shows the average link utilization as well as average network latency of the evaluated applications in 36-node configuration. The proposed network forces requests to be transferred in order through circuit switching, which under-utilizes the available on-chip bandwidth. When traffic injection rate is low, circuit switching can even reduce packet transmission latency, which in turn improves

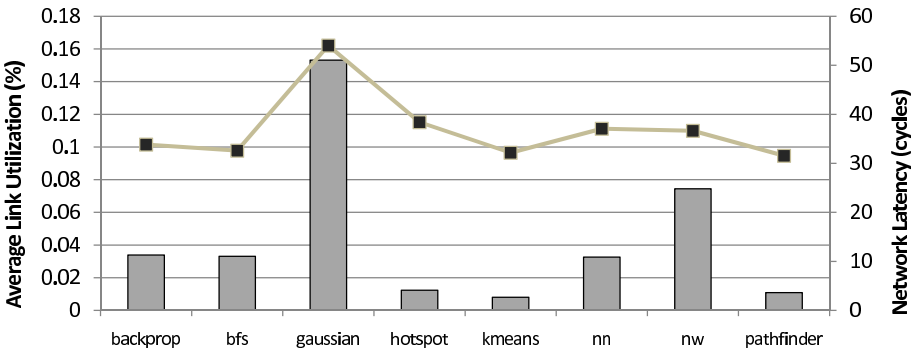


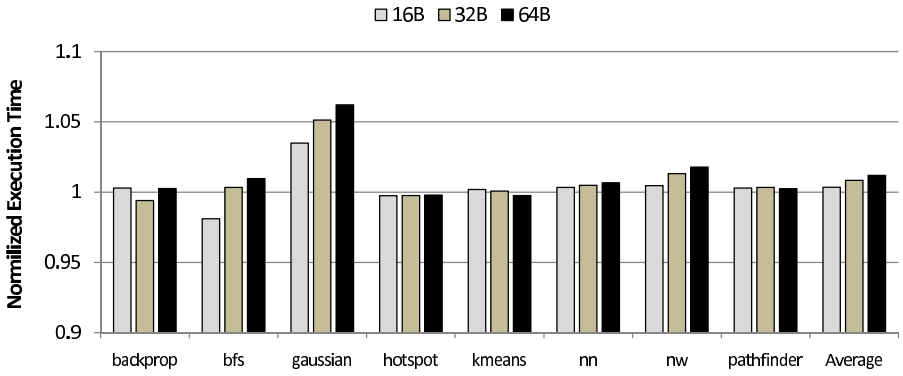
Figure 4.8: Average link utilization and network latency for ordered network.

overall performance for some applications. However, when cores begin to generate considerable amount of outstanding requests, leaving the network under-utilized impacts the performance.

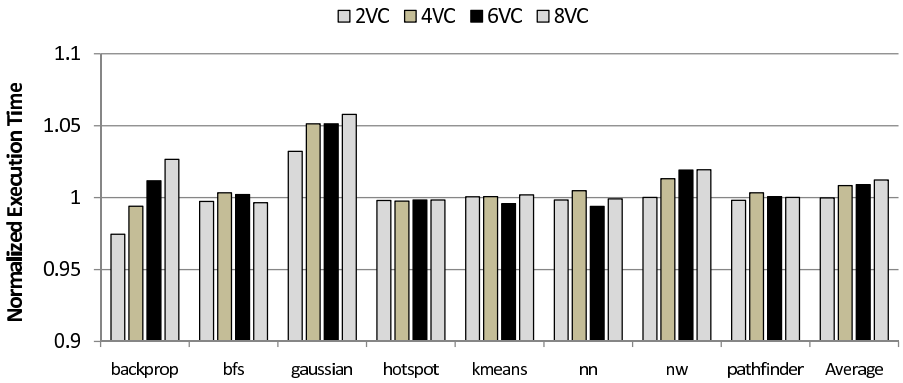
4.2.4 Sensitivity Study

To study how our proposed design is sensitive to network throughput, we vary the channel width as well as the number of VCs for each input port. Channel width influences the number of flits in a data packet, while number of VCs influences network congestion. In Figure 4.9, experimental results are normalized to the baseline un-ordered network with 32-Byte channel width and 4 VCs for each network class.

Channel Width. In most of the evaluated applications, our design is insensitive to channel width. However, although not shown in figure, doubling the channel width do improve overall performance by 0.3%-19.6%. Among all applications, gaussian and nw are more sensitive to varying channel width, while the performance improvement of increasing channel width from 16-Byte to 32-Byte is 19.6% and 13.6%, respectively. Since gaussian and nw are throughput intensive applications as discussed above, increasing channel width will in face reduce the execution time. However, our proposed network is not utilizing the



(a) Channel width



(b) Number of VCs

Figure 4.9: Normalized execution time with varying network parameters in 36-node system.

network bandwidth efficiently in such cases, hence do not achieve the same benefit with wider channels compared to the baseline packet switching network.

Number of VCs. In the baseline network, 2 VCs does not provide enough bandwidth. When increasing the number of VCs from 2 to 4, from 4 to 6, and from 6 to 8, average performance is improved by 7.1%, 0.9%, and 0.4%, respectively, in the baseline network (data not shown in figure). Due to the same reason described above, the performance gap between our network and the baseline is larger when VC number increases from 2 to 4,

compared to the other cases. Beyond 4 VCs, the proposed network is insensitive to varying VC numbers.

4.2.5 Discussion

In this section, we briefly discuss how the proposed network can be possibly applied to systems with different cache write policies and coherence protocols. We also show through a few examples to demonstrate the memory access behavior provided by our proposed network. Finally, we discuss the limitations of the proposed design.

Caches and Coherence Protocol

In the above evaluation, we assume write-back policy for L1 caches. Upon a write miss, an exclusive read request is first issued to load the data at the missed-write location, followed by a write-hit operation. In such write-allocate policy, write data must be buffered before proper coherence permission is granted, causing additional area and complexity in cores and L1 caches. Typically, modern GPU L1 caches are optimized for throughput and assume write-through policy. Although write-through policy simplifies coherence protocol design, the number of write-throughs could result in significant on-chip traffic, especially in applications with massive data sharing. Nevertheless, our proposed network can be easily applied to systems with write-through caches: the sending of actual write data must be ordered correctly, therefore, consecutive time slots must be reserved in slot tables to transfer the entire cache line. Supporting write-through cache requires a larger slot table compared to write-back. Slot table optimizations such as path sharing can potentially reduce the cost of the hardware overhead while improving performance [48], but is beyond the scope of designing an NoC that supports in-network access ordering. In terms of coherence protocol, our network works with directory-based protocols but not snoopy protocols. Network ordering rely on sending token among ordering points such as directories and memory controllers, while snoopy protocols is based on broadcast and do not have ordering points.

Memory Access Behavior

By enforcing the order for both read and write requests, the proposed ordering network guarantees that every node sees the write operations in the same order. If read ordering is relaxed, processor consistency can be guaranteed. In processor consistency, the order in which other processors see the writes from any individual processor is the same as they were issued. The benefit of ordering memory accesses at the network over ordering at the core is that the core can continue issuing instructions and does not stall on writes. We informally show through several examples to demonstrate the memory behavior that can be achieved with the proposed network. These examples are originally demonstrated to study the relaxed behavior of IBM POWER multiprocessors [56].

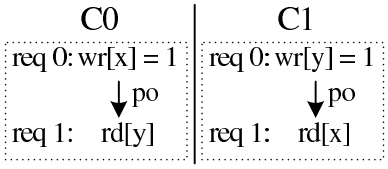


Figure 4.10: Store buffering

Store Buffering. In this example, two cores write to different memory locations, then read from the other's location. Relaxed memory models, in which write-to-read order is relaxed, allows both reads to be 0. This is the key relaxation allowed in TSO model where store buffers are used for optimizing performance. With our proposed network, if all memory requests are sent through circuit switching, correct program order can be maintained (indicated by *po*) and relaxed behavior is prohibited. By circuit switching memory requests and maintaining request order at ordering points, we ensure that the write request from each core is handled prior to the read request.

Message Passing. This is the example given in Section **??**. In this example, Core 0 first writes to x, and then writes to y; while Core 1 first reads from y, and then from x. Our ordering network is able to guarantee the two writes from Core 0, and the two reads from Core 1 are handled in program order, respectively. Therefore, the relaxed behavior

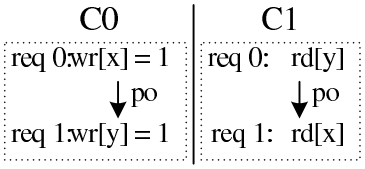


Figure 4.11: Message passing

in which read of y=1 and read of x=0 is prohibited. In contrast, with TSO model, there is no guarantee that $req0$ from Core 1 is processed before $req1$, especially when x and y belong to different memory controllers.

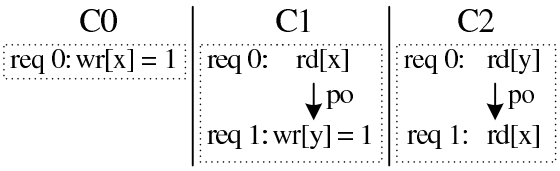


Figure 4.12: Write-to-read causality

Write-to-Read Causality. In this example, Core 1 reads from location x, which is written by Core 0; and Core 2 reads from location y, which is written by Core 1. Our network does not assume any ordering among cores so the demonstrated causality is not maintained. Hence multiple possible results are allowed. However, if the read of x in Core 1 is 1, and the read of y in Core 2 is 1 as well, the read of x in Core 2 can only be 1 with our network applied.

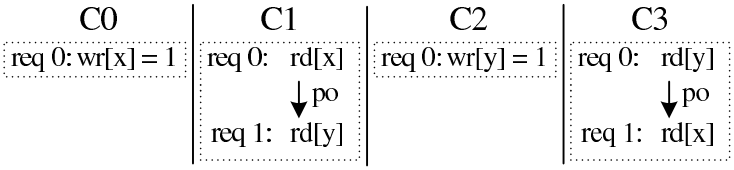


Figure 4.13: Independent reads of independent writes

Independent Reads of Independent Writes. In this example, Core 0 and Core 2 write to different locations x and y, respectively. While Core 1 and Core 3 read from both locations with a different order. If read x in Core 1 returns 1, and read y in Core 3 returns

1, their second read requests cannot both return 0 in our network. Without ordering network, it is possible that the second read request from both Core 1 and 3 reaches their destinations prior to the first request, by which time neither of the write requests from Core 0 and 2 finishes.

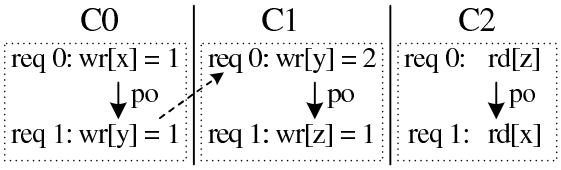


Figure 4.14: Coherence edge

Coherence Edge. In this example, Core 0 writes to location x and y, Core 1 writes to location y and z, while Core 2 reads from z and x. The outcome of Core 2 is undefined with ordering network, meaning that the read value of (z, x) can be any of the four combinations: (0, 0), (0, 1), (1, 0), (1, 1). However, if the execution order between the two writes to y is explicitly enforced, i.e., Core 0 performs the write before Core 1 as shown in figure, then in Core 2, the combination of (1, 0) is prohibited in our network.

Limitations

While the proposed network can potentially provide a strong memory model with minimal performance overhead, it is based on several assumptions. First, we assume that the optimizations performed during compile time and kernel launch time do not violate program order. However, optimizations such as common subexpression elimination and dead code elimination might affect the access to shared data, which in turn violates the program order. Second, we assume the underlying processors are SIMD-like in-order cores and applications exhibit good thread-level parallelism. CPUs are less active during kernel execution. Enforcing memory access order in superscalar out-of-order CPU cores are much more expensive due to lacking parallelism to hide long access latencies [56, 57].

4.3 Related Work

Hybrid switching NoCs have been studies thoroughly in both homogeneous and heterogeneous multicore systems. Spacial-division-multiplexing-based (SDM-based) hybrid switching NoCs have been proposed to provide QoS support in SoCs [29, 31, 30]. Jerger et al. [10] use SDM-based hybrid switching NoCs to improve the performance of coherence-based traffic in CMPs. However, SDM introduces packet serialization delay, due to the fact that a channel is physically divided into multiple planes. For throughput-intensive GPGPU applications, packet serialization increases intra-router contention and can potentially degrade the system performance. TDM-based circuit switching provides bandwidth and latency guarantees in Æthereal [26] and NOSTRUM NoC [58]. More recently, TDM-based hybrid switching NoCs for heterogeneous multicore and accelerator rich architectures have been proposed [48, 49]. These work mainly focus on reducing NoC energy consumption and are only able to provide best-effort circuit switching communication. However, to maintain global memory access order with circuit switching, our proposed design guarantees all computation cores can reach ordering points through circuit switching.

Ring-based interconnects are widely adopted by mainstream commercial multicore processors with data-parallel accelerators including the IBM Cell [59], Intel Larrabee [60], and Intel Sandy Bridge [61]. The simplicity of control logic and datapath enables efficient router implementation in ring-based NoCs. Instead of using ring-based NoC as the data network, we use it as a token network to minimize the cost of token passing.

SCORPIO [62] supports global ordering of requests on a mesh network for snoopy coherence. SCORPIO consists of an un-ordered packet switching data network for broadcasting coherence messages, and a fixed-latency bufferless notification network to achieve distributed global ordering. The data network is heavily optimized for latency so that broadcast requests can traverse through multiple hops in a single cycle. SCORPIO is demonstrated to be efficient in 36 and 64-core CMP systems that contain only in-order CPU cores. However, in heterogeneous multicore systems where streaming traffic is dominant,

snoopy coherence is less efficient than directory-based coherence. Moreover, maintaining global ordering for wide GPU data requires more buffering resource at the NI.

Hechtman and Sorin [55] re-visit hardware memory models in massively-threaded throughput-oriented processors like GPGPUs, and observe that sequential consistency achieves the same performance compared to weak consistency models on a variety of applications. While similar observation is made in our work, we focus on heterogeneous multicore systems that contain both CPU and GPU cores. Moreover, we propose to push the responsibility of enforcing memory consistency to the network so that performance critical hardware modifications at cores can be avoided.

4.4 Conclusion

We design, implement, and evaluate an NoC design which provides in-network memory access ordering for directory-based heterogeneous multicore systems. The proposed network contains a hybrid switching NoC, which serves as the underlying communication infrastructure; as well as a light-weight token ring network, which is used to enforce memory access order among multiple ordering points. Evaluation results show that the proposed NoC brings insignificant impact to overall performance. In comparison to the baseline un-ordered mesh network, the proposed design causes only 0.5% and 0.8% performance degradation in 16-node and 36-node heterogeneous multicore systems, respectively. Overall, we believe the proposed NoC is a viable way of maximizing memory-level parallelism while maintaining memory access order.

Chapter 5

Scalability of Hybrid-Switched NoC

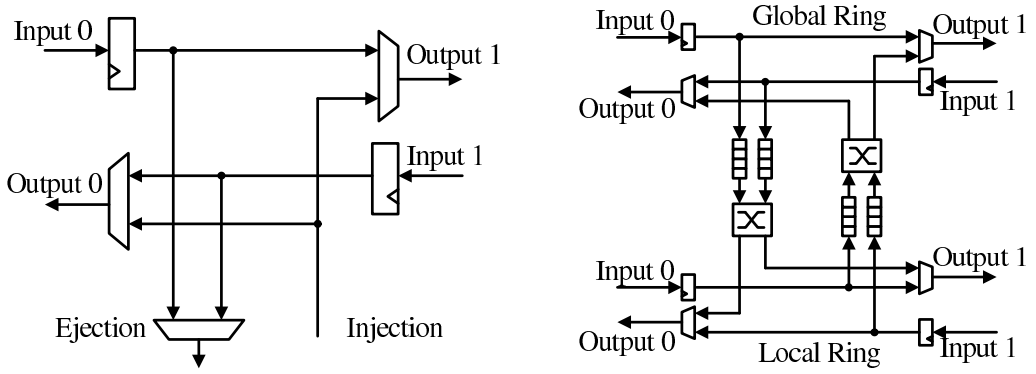
Despite the increase in the number of accelerator cores that are integrated onto the same processor, the on-chip interconnection network must continuously support the high throughput on-chip traffic with a stringent power budget. Therefore, NoCs must be able to scale while still maintain energy efficiency. In this Chapter, we compare the performance and energy efficiency of TDM-based hybrid-switched NoC with the de facto NoC architectures.

5.1 NoC Design Choices

To design an optimal NoC for data-parallel accelerators, trade-offs must be made between energy and performance. A ring-based NoC uses energy efficient routers, but suffers from significant link energy dissipation and potentially larger hop count. In comparison, a hierarchical ring-based NoC has better scalability, at the cost of more complex routers; however, link energy consumption can still be a dominant factor. A packet-switched mesh-based NoC scales well, at the cost of longer her-hop delay and higher router energy consumption.

Finally, a hybrid-switched mesh-based network is more energy efficient than a packet-switched network; however, maintaining the slot tables introduces additional overhead. In this section, we first investigate four NoC design choices, namely, single ring-, hierarchical ring-, packet-switched mesh-, and circuit-switched mesh-based interconnects. Then we introduce hybrid-switched NoC and show how it can achieve better performance and energy efficiency by taking advantage of the GPU traffic pattern.

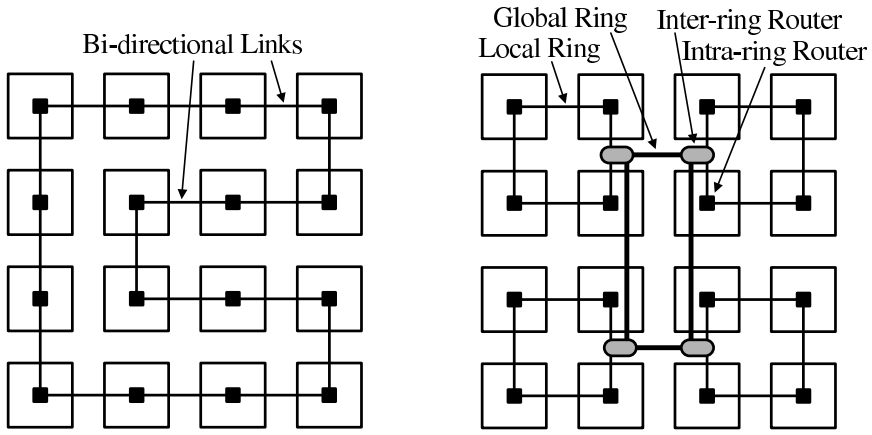
5.1.1 Ring-based NoC



(a) Ring router (b) Inter-ring router

Figure 5.1: Router microarchitecture for ring-based NoCs.

Single Ring. In a single ring NoC, traffic is injected into the network at each router, and removed from the network when reaches its destination. A ring-based interconnect prioritizes the traffic that are already on the network. A packet is guaranteed to moving forward in the next cycle, thus buffering in routers is not required. Figure 5.1(a) illustrates the router microarchitecture in a bidirectional ring network. The corresponding network topology is shown in Figure 5.2(a). We assume the link bandwidth is sufficiently large so that all packets are 1-flit wide. The router uses input registers at both input ports for latching. The multiplexer at each output port either forwards the incoming flit from its neighbor, or the flit from the injection port. New traffic can be injected only when the



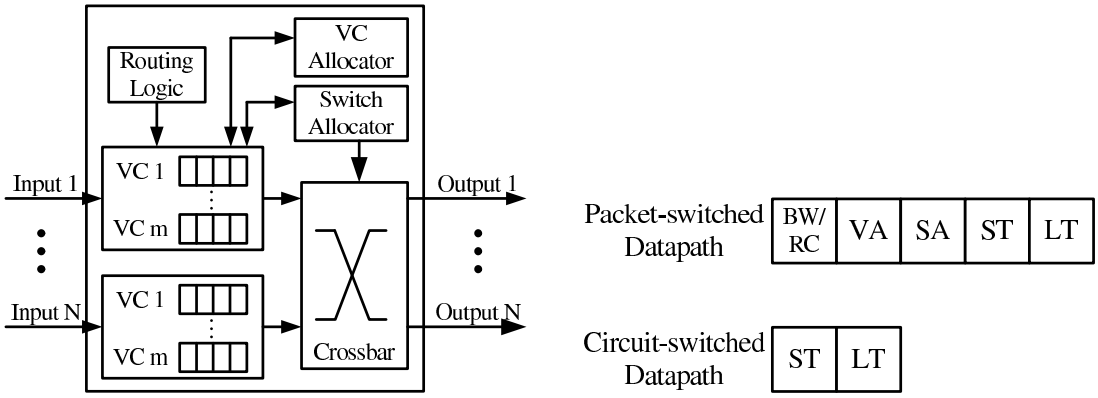
(a) Ring (b) Hierarchical ring

Figure 5.2: Network topology for ring-based NoCs.

output port is free of contention. When a flit reaches its destination, it is pushed to the ejection port. Notice that flits from both input ports can arrive in the same cycle and need to be ejected, ejection contention must be avoided due to the lack of router buffering. We assume each ejection port can accept a flit from one direction on even clocks and from the other direction on odd clocks like [60]. Since the network traversal latency from the source to the destination is determined, the source can therefore inject a flit at an appropriate time and avoids contention at the destination.

Hierarchical Ring. As shown in Figure 5.2(b), in a hierarchical ring network, multiple local rings are connected to a global ring by inter-ring routers [63, 64]. Figure 5.1(b) gives a possible implementation of the inter-ring router, in which buffers are used to temporarily store packets when the output link is busy, and a crossbar from/to the local ring switches the packet to an appropriate output port. Compared to single ring, a hierarchical ring NoC requires more routers as well as wire segments. Regardless of the hardware overhead, heterogeneous router design and global wiring dramatically complicates the circuitry design.

5.1.2 Mesh-based NoC



(a) Packet-switched router (b) Mesh-based router pipeline

Figure 5.3: Router microarchitecture and pipeline for mesh-based NoCs. BW: Buffer Write, RC: Route Compute, VA: Virtual Channel Allocation, SA: Switch Allocation, ST: Switch Traversal, LT: Link Traversal

Packet-switched Mesh. Figure 5.3(a) depicts a virtual-channeled packet-switched router with N input and output ports. Its pipeline stages are shown in Figure 5.3(b). A router in a mesh network typically consists of five pipeline stages including link traversal: input buffer write and route computation, VC allocation, switch allocation, and switch traversal [14]. Input buffers are managed using credit-based flow control to avoid buffer overflow. A packet is broken into flits at the NI. When a flit is injected into the router, the routing logic immediately decides its output port. The VC allocator assigns a free virtual channel on that particular output port. Upon successful allocation of an output VC, the switch allocator configures the crossbar and forwards the flit to the output link.

Circuit-switched Mesh. A two-stage circuit-switched pipeline is shown in Figure 5.3(b). In a circuit-switched network, once a circuit-switched connection is setup, messages are forwarded from the source to the destination without buffering and routing along the path. Thus, for individual messages, both transmission latency and dynamic energy consumption are significantly reduced. These circuits act as dedicated express

channels from one node to the other, thus circuit-switched network trades off flexibility and on-chip resource sharing with much lower transmission delay and energy consumption.

5.1.3 Hybrid-switched NoC

As introduced in Chapter 2, a hybrid-switched NoC, built upon a packet-switched NoC, allows traffic be sent in both packet- and circuit-switched manner. Dedicated circuit-switched paths are established between communicating pairs. Once a connection is set up, messages from the same source to the same destination can repeatedly reuse the path without destroying the connection. With the support of slot tables, the hybrid-switched router allows packet- and circuit-switched traffic to share the same communication fabric through time-division multiplexing. A slot table has limited number of entries; and the table is referred every cycle in modulo-fashion. A slot table entry indicates whether a particular cycle is reserved for a circuit-switched datapath; it also keeps track of the input-output port mapping for a particular circuit-switched path. A flit is injected to either the packet- or the circuit-switched data path from the NI, depending on whether a circuit-switched connection is already established from its source to its destination. Once the flit is on the circuit-switched path, it can be forwarded towards its destination without buffering and routing at each router. Each hop takes only two cycles for a circuit-switched flit, and thus circuit switching suffers lower packet transmission latency than packet switching.

Switching Decision for GPU Traffic

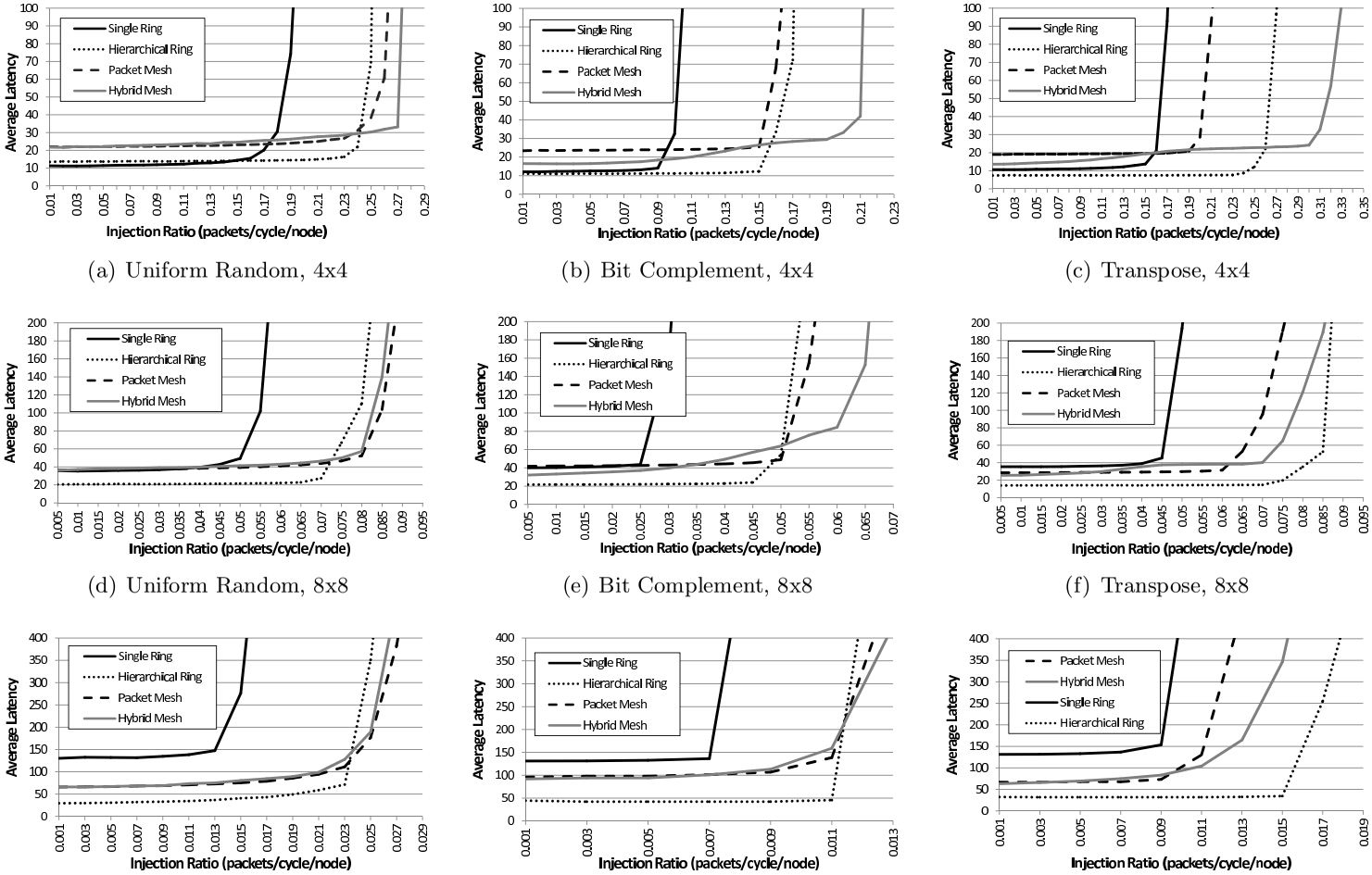
In order to achieve optimal utilization of the network and minimize network energy consumption, deciding whether a message should be forwarded with packet or circuit switching is multifaceted. From energy perspective, reserving more dedicated paths and circuit switching more packets reduces NoC energy consumption, at the cost of larger slot tables. From performance perspective, circuit-switched packets spend less time traversing the network; however, packets might stall and wait for a circuit-switched time slot before

transmitting.

Considering the GPU traffic pattern and design trade-offs, we here propose a two-phase switching decision making mechanism for the hybrid-switched network. The *first phase* identifies whether a circuit-switched connection should be established between communicating pairs. This particular phase contains two steps: 1) identify frequent end-to-end communication pairs, and 2) give preference to long-distance communication. At the start of program execution, each NI keeps track of the number of communications toward each destination. After a period of time T, among the nodes whose communication frequency are above the average packet injection ratio, the NI selects the one with longest distance and establishes a circuit-switched connection. The *second phase* decides whether a candidate packet should be buffered for circuit switching. A packet is considered to be circuit-switched only when no performance penalty is caused. Performance tolerance cycles (or slack) for a packet can be estimated by referring to the number of available threads inside each GPU core [13]. If the tolerance cycle compensates for the NI buffering delay plus the circuit-switched data path traversal delay, a packet is buffered and later on circuit-switched. Otherwise, it is considered to be packet-switched.

5.2 Evaluation on Synthetic Workload

Compared to real applications, synthetic traffic makes it possible to study the NoC behavior under a wider range of on-chip traffic. In this section, we evaluate different NoC designs with three traffic patterns: 1) uniform random (UR), in which destinations are randomly selected; 2) bit complement(BC), in which traffic from (x, y) are sent to (k_y-y-1, k_x-x-1), where k_x and k_y are the number of nodes in x and y dimension, respectively; and 3) transpose (TR), in which messages from (x, y) are sent to (y, x) [35]. Open-loop simulation results are presented in Section 5.2.2, and closed-loop simulation results are presented in Section 5.2.3.



(a) Uniform Random, 4x4 (b) Bit Complement, 4x4 (c) Transpose, 4x4

(d) Uniform Random, 8x8 (e) Bit Complement, 8x8 (f) Transpose, 8x8

(g) Uniform Random, 16x16 (h) Bit Complement, 16x16 (i) Transpose, 16x16

Figure 5.4: Throughput-latency evaluation for 4x4, 8x8, and 16x16 networks with open-loop synthetic traffic.

5.2.1 Evaluation Infrastructure

The interconnection network and its power model are based on Garnet [14] and Orion 2.0 [16], respectively. We update the router power model with the hybrid-switched architecture. Router and link parameters are shown in Table 5.1. For a fair comparison, we assume constant bisection bandwidth for all networks. We assume 45nm technology node. Routers are operated at 1.5GHz with 1.0V supply voltage.

Router Parameter	
Single Ring	1-cycle delay
Hierarchical Ring	4-flit buffer for both intra-ring routers and inter-ring routers
Packet-switched	4-stage pipelined, X-Y routing, 4 VCs/port, data-width-size buffer/VC
Hybrid-switched	**4x4:** 16-entry slot table, **8x8:** 64-entry slot table, **16x16:** 128-entry slot table
Link Parameter	
Single Ring	2mm, 1-cycle delay, 64-byte width
Hierarchical Ring	**local:** 2mm, 1-cycle, 64-byte **global:** 2mm, 2-cycle, 64-byte **third-level global:** 4mm, 4-cycle, 64-byte **fourth-level global:** 4mm, 4-cycle, 64-byte
Packet-switched	2mm, 1-cycle delay, **4x4:** 32-byte width,
Hybrid-switched	**8x8:** 16-byte width, **16x16:** 8-byte width

Table 5.1: Interconnection network configuration for synthetic workload evaluation.

5.2.2 Open-Loop Simulation Results

Open-loop measurements aim to study the network characteristics, for example, throughput and zero-load latency, on a specific traffic pattern. With an infinite source queue at each injection node, the traffic parameters are independent of the network while source queuing delay and head-of-line blocking effects are properly accounted. Since it does not necessarily measure overall system performance, we use open-loop simulation to study the network

characteristics of various design choices.

Figure 5.4 shows the load-latency curves for the evaluated NoCs with different traffic patterns. In 16-node network, ring-based NoCs achieve lower zero-load latency while hybrid-switched mesh network achieves higher throughput. Forwarding messages through circuit-switched data paths makes better utilize of the available link bandwidth. In UR traffic, hybrid switching hardly improves the network throughput over the packet-switched NoC. This is because hybrid-switched NoC cannot capture all possible communication pairs for all-to-all traffic, due to the limited size of slot tables. As system size scales to 64 and 256 nodes, hybrid-switched network achieves higher throughput than packet-switched network in BC and TR traffic. Single ring suffers from high network diameter, therefore a significant increase of zero-load latency is seem. BC traffic are sent through the middle of the network for all topologies, therefore the throughput is limited by the bisection bandwidth. Hierarchical ring performs the best with TR traffic in larger networks, because significant amount of traffic can be handled by local rings and lower-level global rings, the available on-chip bandwidth is largely exploited. While in mesh-based network, traffic is congested at routers residing on the $x = -y$ diagonal, therefore the network saturates at a early point.

5.2.3 Closed-Loop Simulation Results

Closed-loop measurements, in which the network performance influences the traffic pattern, aim to measure the overall system performance rather than the network performance. In our closed-loop simulation, synthetic traffic are used to model the creation of memory requests. Once requests are received, responses are generated from the destination and sent back to the source. We only allow a limit number of outstanding requests per injection node, i.e., a node stops injecting new requests into the network until its outstanding requests get resolved. In the evaluation, each node generates 5000 requests; and simulation terminates after all responses are received. Usually a request message is much shorter than the link

bandwidth of ring networks, which causes bandwidth fragmentation on ring links [65]. In our experiment, we assume request messages are of the same size as response messages. The reason behind using closed-loop simulation is to mimic the impact of miss status holding registers (MSHRs) in a processor. In a GPU, it is anticipated that there are much more outstanding memory requests than in a CPU, due to the fact that a GPU core supports thousands of threads. However, it is unlikely for a GPU core to support unlimited number of outstanding requests without stalling the processor.

We evaluate 16-, 64-, and 256-node networks and vary the number of outstanding requests from 8 to 64. Performance and network energy consumption results are shown in Figure 5.5 and Figure 5.6, respectively. We observe that for UR traffic, the performance difference between hierarchical ring-based network and mesh-based networks is minimal; for BC traffic, hierarchical ring performs slightly better than hybrid-switched network; while for TR traffic, hierarchical ring performs the best, due to the reason discussed above. In terms of network energy consumption, hybrid-switch NoC achieves the best energy efficiency in larger networks under BC and TR traffic. For UR traffic, hybrid-switched NoC is energy inefficient, because the energy saving from circuit switching packets does not offset the cost of maintaining slot tables in the hybrid-switched NoC. However, GPU traffic exhibits regularity hence is expected to benefit from hybrid switching.

5.3 Evaluation on GPGPU Workload

In this section, we present a detailed evaluation of the NoC design choices described in Section 5.1 using realistic workloads.

5.3.1 Evaluation Infrastructure

We use GPGPU-Sim [41] to simulate the data-parallel accelerator architecture. Simulator configuration can be found in Table 5.2. Memory controllers are placed at the left and right boarder of the chip. Again, bisection bandwidth is kept constant for all networks.

We use CUDA applications from Rodinia [43] benchmark suites as accelerator workloads. The evaluated applications include: *bfs*, *mum*, *gaussian*, *kmeans*, *lavaMD*, and *pathfinder*.

Number of Compute Cores	16/36/64
Number of Memory Controllers	4/6/8
MSHRs/Core	32
Warp Size	32
SIMD Pipeline Width	16
Number of Threads/Core	1024
Number of CTAs/Core	8
Number of Registers/Core	32768
L1 Cache/Core	16KB
Shared Memory/Core	32KB

Table 5.2: Simulator configuration for realistic workload evaluation.

5.3.2 Experimental Results

We scale the system size from 16-core (4x4) to 64-core (8x8). Figure 5.7 and Figure 5.8 show the overall system speedup and the network energy consumption, respectively, normalized to the ring network. Each bar under the same application represents a different network configuration.

Performance Analysis

In a 16-node system, hybrid-switched network achieves 7.4% performance improvement over single ring, 2.6% over hierarchical ring, and 3.4% over packet-switched mesh. The improvement becomes 16.2% over single ring, 7.2% over hierarchical ring, 4.9% over packet-switched mesh for 36-node system; and 14.1% over single ring, 6.0% over hierarchical ring, and 6.2% over packet-switched mesh for 64-node system. The performance improvement of hybrid-switched NoC mainly comes form forwarding packets to destination with shorter delay. Table 5.3 presents the traffic injection ratio, measured in flits/node/cycle, and percentage of flits that are circuit-switched for each application. *Pathfinder* has very low

injection ratio and high thread-level parallelism (TLP), thus is insensitive to network latency. Circuit switching traffic for these applications does not lead to a direct performance improvement. On the other hand, applications such as *bfs* and *lavaMD* have lower TLP with moderate throughput requirement, and can potentially benefit from hybrid-switched NoC.

	16-node		36-node		64-node	
GPU benchmarks	Inj. rate	CS per.	Inj. rate	CS per.	Inj. rate	CS per.
bfs	0.32	46.41	0.22	28.71	0.13	20.34
mum	0.20	44.25	0.21	26.20	0.14	17.32
gaussian	0.23	46.51	0.19	27.47	0.12	19.21
kmeans	0.15	61.93	0.20	40.28	0.14	20.68
lavaMD	0.29	35.82	0.09	38.36	0.07	35.93
pathfinder	0.04	62.67	0.06	37.84	0.06	21.45

Table 5.3: Traffic injection ratio (flits/node/cycle) and percentage of flits that are circuit-switched

Network Energy Analysis

As expected, links contribute to a significant portion of network energy dissipation in ring-based networks. Even in hierarchical ring, over 70% of network energy is consumed by link. Packet-switched NoC has the highest router energy consumption among all evaluated networks. Hybrid-switched design demonstrated to be the most energy efficient, especially in larger systems. Compared to the packet-switched network, hybrid-switched design reduces the router energy consumption by 14.6%, 11.4%, and 8.5% in 16-, 36-, and 64-node system, respectively. The amount of energy saving is dependent on the percentage of traffic sent through the circuit-switched data path, as shown in Table 5.3. Better energy saving is achieved when more messages are sent through circuit switching. We see a decrease in circuit-switched traffic percentage as the system size grows. This is due to $slottable fragmentation$. In our experiment setup for 64-node system, a data packet is

broken into 4 flits. Therefore, in order for the packet to transfer without interleaving, 4 consecutive slot table entries are reserved in a row. If the number of slot table entries between two reservations is smaller than 4, these entries cannot be utilized, resulting in fragmentation. An efficient slot reservation strategy can reduce slot table fragmentation, but is beyond the scope of this paper. Although missing evaluation in this work, hybrid-switched design can potentially provide opportunities for using smaller router input buffers to reduces router leakage [48], because the circuit-switched data path alleviates the burden on the packet-switched counterpart, fewer VC buffers are required.

In summary, among the evaluated NoC design choices, hybrid-switched network achieves promising performance improvement and energy efficiency. We believe that future data-parallel accelerators can potentially benefit from hybrid switching.

5.4 Related work

Ring-based NoCs have been extensively studied in the context of many-core processors with data-parallel accelerators. Ainsworth et al. [66] investigated the Cell Broadband Engine Element Interconnect Bus, and Kim et al. [65] proposed a lightweight router architecture to improve the throughput of ring networks in a small scale system. Overall, ring-based NoCs are unable to scale to a large number of cores. A hierarchical organization of rings addresses this scalability issue and was shown to be able to achieve comparable performance characteristics with packet-switched mesh-based networks in larger systems [63, 64, 67]. However, in the context of data-parallel accelerators, we found that hierarchical ring networks are not as energy efficient as hybrid-switched networks.

Various proposals for simpler and more energy efficient NoC routers have emerged. However, many are not suitable for traffic patterns originated from data-parallel accelerators. For example, bufferless routers are energy efficient when the traffic injection rate is low, and thus they were demonstrated as being effective in NoCs that connect many CPU cores [24, 68]. However, data-parallel accelerators often have a much higher throughput

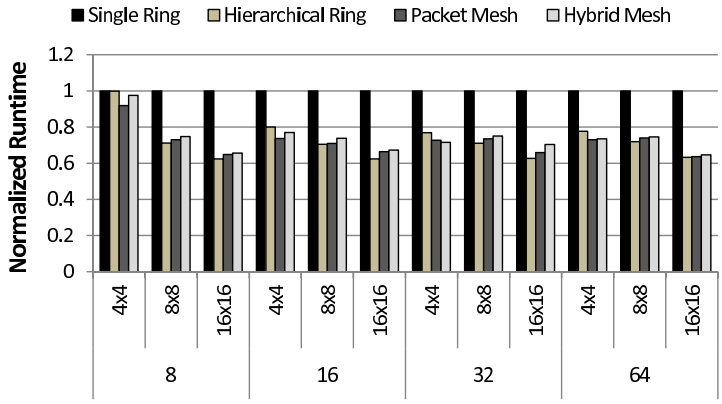
requirement than CPU cores. Therefore, bufferless routers are not suitable for NoCs that connect accelerators. Various routers optimizations for packet-switched NoCs can be incorporated with hybrid-switched network. For example, Bakhoda et al. [12] proposed the throughput-effective checkerboard organization for handling on-chip traffic in GPUs, and Kim [69] proposed a low-cost router microarchitecture that decouples the routing in the x and the y dimensions to reduce the router area and power consumption. These techniques are orthogonal and can be applied to hybrid-switched NoC.

Spacial-division-multiplexing-based (SDM-based) hybrid-switched NoCs have been proposed to provide QoS support in SoCs [29, 31, 30]. Jerger et al. [10] use SDM-based hybrid-switched NoCs to improve the performance of coherence-based traffic in CMPs. However, SDM introduces packet serialization delay, because the link bandwidth is physically divided into planes. In high-throughput networks, packet serialization increases intra-router contention and can potentially degrade the system performance. Previously proposed TDM-based circuit switching provides bandwidth and latency guarantees in Æthereal [26] and NOSTRUM NoC [58]. However, without adequate mechanisms for adapting traffic patterns, these approaches have a significant hardware overhead that results in increased energy dissipation.

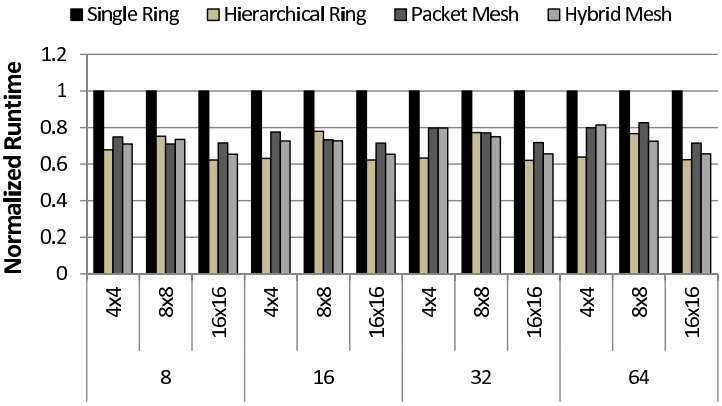
5.5 Summary

In this work, we evaluate four NoC design choices and present a case for hybrid-switching in mesh-based NoCs as energy efficient alternative for large-scale many-core processors with high-throughput data-parallel accelerators. We have demonstrated that the traffic pattern generated by GPU cores are well-suited for hybrid-switched networks. Results show that hybrid-switched network with proposed switching decision making mechanism can effectively handle throughput intensive on-chip traffic. Overall, when the number of cores scales beyond 16, hybrid-switched NoCs are able to approximate the low per-hop delay and per-hop energy consumption of ring-based NoCs while achieving the scalability

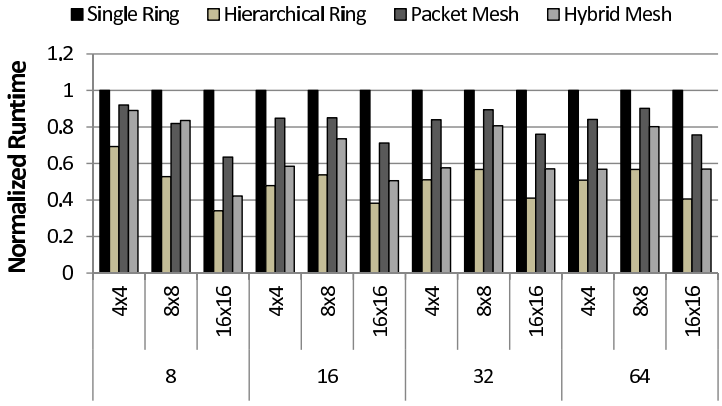
of mesh-based NoCs.



(a) Uniform Random

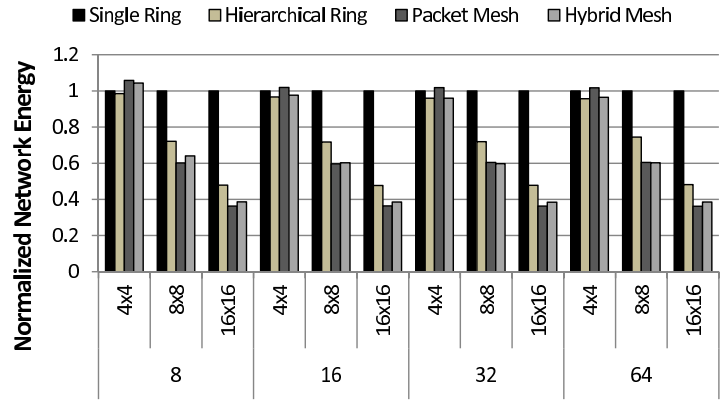


(b) Bit Complement

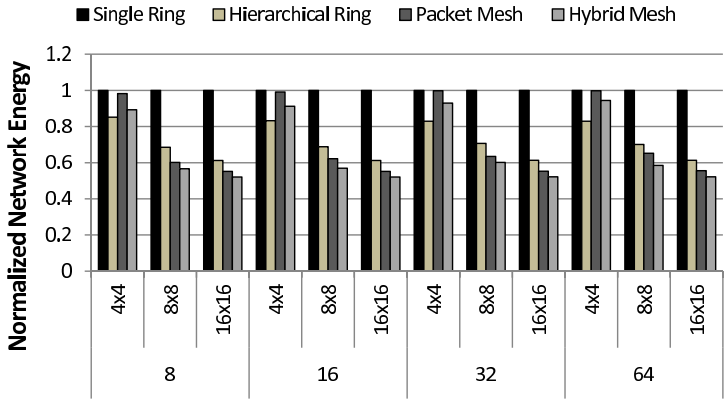


(c) Transpose

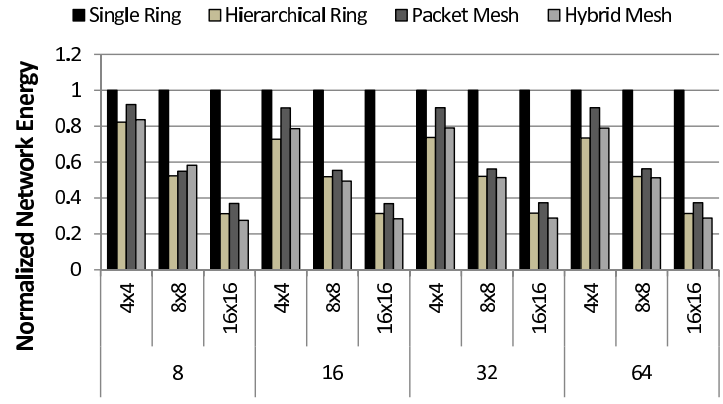
Figure 5.5: Performance evaluation for 4x4, 8x8, and 16x16 network with 8, 16, 32, and 64 outstanding requests under closed-loop synthetic traffic. Results are normalized to Single Ring.



(a) Uniform Random



(b) Bit Complement



(c) Transpose

Figure 5.6: Network energy evaluation for 4x4, 8x8, and 16x16 networks with 8, 16, 32, and 64 outstanding requests under closed-loop synthetic traffic. Results are normalized to Single Ring.

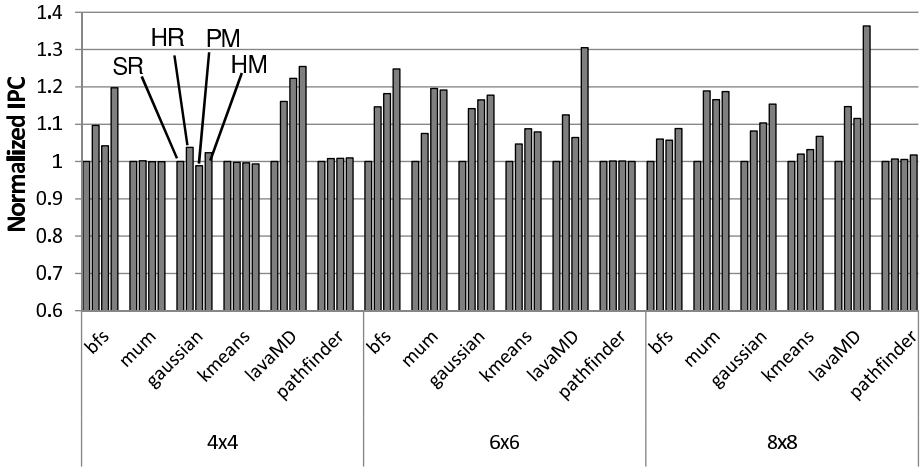


Figure 5.7: System speedup for: single ring (SR), hierarchical ring (HR), packet-switched mesh (PM), and hybrid-switched mesh (HM) interconnect. Results are normalized to SR.

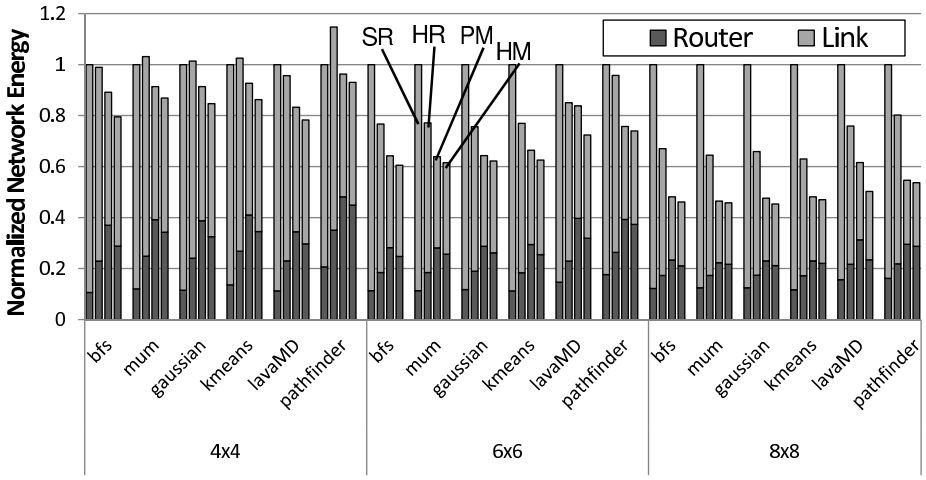


Figure 5.8: Normalized network energy consumption for: single ring (SR), hierarchical ring (HR), packet-switched mesh (PM), and hybrid-switched mesh (HM) interconnect. Results are normalized to SR.

Chapter 6

Conclusions and Future Directions

Concomitant with diminishing performance improvement of complex uniprocessors, integrating multiple energy-efficient accelerator cores with a traditional superscalar cores onto the same die emerged as way to achieve the desired performance goal within a stringent power budget. Heterogeneous system architecture further enables a logically tight integration of on-chip processing units by providing a unified address space [70]. As the system size continues to scale in terms of both number and variety of cores, efficient data movement and data sharing becomes critical in achieving optimal system performance. In this dissertation, we have addressed the data movement and data sharing challenges through designing a TDM-based hybrid-switched NoC.

We first make the observation that on-chip communication exhibits variant traffic patterns and diversity in performance requirements. CPU cores are optimized for single-thread workloads, and generate latency-sensitive coherence messages which benefit from packet switching. While GPUs cores are optimized for high throughput workloads, and generate throughput-intensive streaming traffic which benefit from circuit switching. Taking advantage of the diversity, we propose a hybrids-witched network that allows the packet-switched and circuits-witched messages to share the same communication fabric by partitioning the

network through time-division multiplexing. We demonstrate the feasibility of implementing a hybrid-switched router by extending a canonical packet-switched router. In order to quantify the performance benefit from circuit switching a message, we introduce packet slack as a metric and propose a mechanism to predict the slack.

We further optimize the hybrid-switched NoC for better energy efficiency. In particular, we propose path sharing schemes that allow circuit-switched paths to be fully utilized; and aggressive virtual channel power gating mechanism that leverages the energy saving from hybrid switching traffic. The proposed hybrid-switched NoC as well as its optimization is evaluated with both synthetic and realistic workloads. Evaluation results show that hybrid-switched NoC is able to reduce the network energy consumption by as much as 24% and improve system performance by as much as 12%, compared to traditional packet-switched NoCs. Therefore, hybrid-switched NoC enables efficient data movement in heterogeneous multicore systems.

The data sharing challenge in heterogeneous multicore systems is addressed by co-designing the NoC with on-chip memory hierarchy. In particular, we propose an NoC design which provides in-network memory access ordering while still able to maximize memory-level parallelism in a system with stronger memory consistency model such as TSO. The proposed NoC consists of a hybrid-switched data network which guarantees point-to-point ordering, as well as a light-weight ring network which maintains the global order for memory accesses. Compared to a system with relaxed consistency model and unordered packet-switched NoC, enforcing a stronger memory consistency with the proposed network results in negligible performance impact. Therefore, the ordering network is an effective way of optimizing data sharing in heterogeneous multicore systems.

Finally, we conduct a scalability study of the hybrid-switched network in comparison to other on-chip interconnection design choices including ring, hierarchical ring, and packet-switched NoC. We evaluate these alternatives with both synthetic and realistic GPU workloads. We have demonstrated that the traffic pattern generated by GPU cores

are well-suited for hybrid-switched NoC. In a multicore system with more than 16 cores, the hybrid-switched NoC is able to approximate the low per-hop delay and per-hop energy consumption of ring-based NoCs while achieving the scalability of packet-switched NoCs.

6.1 Future Directions

The NoC is becoming increasingly important in multicore systems. In the future, on-chip components such as CPUs, GPUs, caches, I/O, and various programmable logic components are tightly integrated, the NoC performance plays a critical role in determining the overall system performance. In designing NoCs for large scale heterogeneous systems, we see several potential research directions.

6.1.1 Co-design of computation and communication

In current heterogeneous systems, computation and communication are independent in a sense that computation/scheduling does not consider network congestion while on-chip communication is not aware of the locality of the data accesses. The above situation will result in suboptimal system performance and unbalance in resource utilization. Hybrid-switched NoC has the feature of enabling bulk data transfer without interleaving individual requests. Therefore, it is an ideal candidate of maintaining data locality at both ends (core-end and memory-end). Consider an example in which a processor issues multiple requests accessing consecutive memory addresses. In traditional packet-switched NoC, these requests can be blocked by other requests during router arbitration. When arriving at the destination, their locality might be lost because requests from other processors can arrive at anytime and break their consecutiveness. With hybrid-switched NoC, however, the circuit-switched path guarantees consecutive requests from a processor reach their destination consecutively, which can potentially maintain the access locality. Additionally, network congestion can be indicated by the utilization of circuit switched paths. Without complicated logic, processors can schedule the computation accordingly to avoid generating

more traffic that congests the network further.

6.1.2 Evaluating NoC scalability for large scale systems

On-chip communication demands will continue to increase as system size grows. In this paper, we evaluate the NoC scalability using both synthetic and realistic workloads. However, both approaches have limitations. Synthetic workloads allow fast NoC evaluation and design space exploration, but these traffic patterns do not realistically capture cache behavior hence results might not be representative for real systems. On the other hand, simulating a full-system with realistic workloads provides representative results but requires long turn-around time. More importantly, current academic benchmarks are unlikely to keep all processors busy when the system size reaches a certain point. Therefore, in the context of evaluating NoCs for large future systems, it is increasingly crucial to look for an alternative simulation methodology which is fast, accurate, and scalable.

Recent work proposed the SynFull methodology that performs statistical analysis of a workload's on-chip traffic to create compact traffic generators based on Markov models [71]. While the models generate synthetic traffic, the traffic is statistically similar to the original trace and can be used for fast NoC simulation. However, a key limitation of SynFull is that the trace collection and analysis performed on an N-node system can only create synthetic traffic generators for other N-node systems. To simulate a larger system, extrapolation methodologies can be proposed, which takes traces collected from various smaller systems, and projects the trace to a parametrically different system. The goal is to ensure that the extrapolated trace is representative of the NoC communication behavior of the target system.

References

[1] Guhan Krishnan, Dan Bouvier, Praveen Dongara, and Louis Zhang. Energy efficient graphics and multimedia in 28nm carrizo apu. In *27th Hot Chips*, 2015.

[2] P. Hammarlund, A.J. Martinez, A.A. Bajwa, D.L. Hill, E. Hallnor, Hong Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R.B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton. Haswell: The fourth-generation intel core processor. *Micro, IEEE*, 34(2):6–20, 2014.

[3] Oreste Villa, Daniel R. Johnson, Mike O'Connor, Evgeny Bolotin, David Nellans, Justin Luitjens, Nikolai Sakharnykh, Peng Wang, Paulius Micikevicius, Anthony Scudiero, Stephen W. Keckler, and William J. Dally. Scaling the power wall: A path to exascale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 830–841, 2014.

[4] M.J. Schulte, M. Ignatowski, G.H. Loh, B.M. Beckmann, W.C. Brantley, S. Gurumurthi, N. Jayasena, I. Paul, S.K. Reinhardt, and G. Rodgers. Achieving exascale capabilities through heterogeneous computing. *Micro, IEEE*, 35(4):26–36, 2015.

[5] David Kanter. Graphics processing requirements for enabling immersive vr. In *AMD White Paper*, July 2015.

[6] Chansup Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, J. Kepner, A. McCabe, P. Michaleas, J. Mullen, D. O'Gwynn, A. Prout, A. Reuther, A. Rosa, and C. Yee. Driving big data with big compute. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, pages 1–6, 2012.

[7] Larry Brown. Nvidia. Deep learning with gpus. http://www.nvidia.com/content/events/geoInt2015/LBrown_DL.pdf.

[8] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *Solid-State Circuits, IEEE Journal of*, 43(1), Jan 2008.

[9] A Kumar, P. Kundu, A.P. Singhx, Li-Shiuan Peh, and N.K. Jha. A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator in 65nm cmos. In *Computer Design, 2007. ICCD 2007. 25th International Conference on*, pages 63–70, 2007.

[10] N. Enright Jerger, Li-Shiuan Peh, and M. Lipasti. Circuit-switched coherence. In *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, pages 193–202, 2008.

[11] Reetuparna Das, Satish Narayanasamy, Sudhir K. Satpathy, and Ronald G. Dreslinski. Catnap: Energy proportional multiple network-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 320–331, 2013.

[12] Ali Bakhoda, John Kim, and Tor M. Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, pages 421–432, 2010.

[13] Jieming Yin, Pingqiang Zhou, Anup Holey, Sachin S. Sapatnekar, and Antonia Zhai. Energy-efficient non-minimal path on-chip interconnection network for heterogeneous

systems. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pages 57–62, 2012.

[14] N. Agarwal, T. Krishna, Li-Shiuan Peh, and N.K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33–42, 2009.

[15] Arnab Banerjee, Robert Mullins, and Simon Moore. A power and energy exploration of network-on-chip architectures. In *Proceedings of the First International Symposium on Networks-on-Chip*, NOCS '07, pages 163–172, 2007.

[16] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 423–428, 2009.

[17] Li Shang, Li-Shiuan Peh, and Niraj K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, HPCA '03, 2003.

[18] Seung Eun Lee and Nader Bagherzadeh. A variable frequency link for a power-aware network-on-chip (noc). *Integr. VLSI J.*, 42(4):479–485, September 2009.

[19] Asit K. Mishra, Reetuparna Das, Soumya Eachempati, Ravi Iyer, N. Vijaykrishnan, and Chita R. Das. A case for dynamic frequency tuning in on-chip networks. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 292–303, 2009.

[20] Hiroki Matsutani, Yuto Hirata, Michihiro Koibuchi, Kimiyoshi Usami, Hiroshi Nakamura, and Hideharu Amano. A multi-vdd dynamic variable-pipeline on-chip router

for cmps. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 407–412, 2012.

[21] Pingqiang Zhou, Jieming Yin, Antonia Zhai, and Sachin S. Sapatnekar. Noc frequency scaling with flexible-pipeline routers. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pages 403–408, 2011.

[22] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. Express virtual channels: Towards the ideal interconnection fabric. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 150–161, 2007.

[23] Mitchell Hayenga, Natalie Enright Jerger, and Mikko Lipasti. Scarab: A single cycle adaptive routing and bufferless network. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 244–254, 2009.

[24] Thomas Moscibroda and Onur Mutlu. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 196–207, 2009.

[25] Chrysostomos A. Nicopoulos, Dongkook Park, Jongman Kim, N. Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. Vichar: A dynamic virtual channel regulator for network-on-chip routers. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 333–346, 2006.

[26] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: concepts, architectures, and implementations. *Design Test of Computers, IEEE*, 22(5):414–421, 2005.

[27] Tobias Bjerregaard and Jens Sparso. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, pages 1226–1231, 2005.

[28] Pascal T. Wolkotte, Gerard J. M. Smit, Gerard K. Rauwerda, and Lodewijk T. Smit. An energyefficient reconfigurable circuit switched network-on-chip. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005.

[29] Francesca Palumbo, Danilo Pani, Andrea Congiu, and Luigi Raffo. Concurrent hybrid switching for massively parallel systems-on-chip: The cyber architecture. In *Proceedings of the 9th Conference on Computing Frontiers*, CF '12, pages 173–182, 2012.

[30] Angelo Kuti Lusala and Jean-Didier Legat. A sdm-tdm-based circuit-switched router for on-chip networks. *ACM Trans. Reconfigurable Technol. Syst.*, 5(3):15:1–15:22, October 2012.

[31] M. Modarressi, H. Sarbazi-Azad, and M. Arjomand. A hybrid packet-circuit switched on-chip network based on sdm. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 566–569, 2009.

[32] A.K. Lusala and J. Legat. Combining sdm-based circuit switching with packet switching in a noc for real-time applications. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 2505–2508, 2011.

[33] Tushar Krishna, Chia-Hsin Owen Chen, Woo Cheol Kwon, and Li-Shiuan Peh. Breaking the on-chip latency barrier using smart. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 378–389, 2013.

[34] Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu. Kilo-noc: A heterogeneous network-on-chip architecture for scalability and service guarantees. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, pages 401–412, 2011.

[35] Arjun Singh, William J Dally, Amit K Gupta, and Brian Towles. Goal: A load-balanced adaptive routing algorithm for torus networks. In *International Symposium on Computer Architecture (ISCA*, pages 194–205. ACM Press, 2003.

[36] Andrew B. Kahng, Bill Lin, and Siddhartha Nath. Explicit modeling of control and data for improved noc router estimation. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 392–397, 2012.

[37] Mitchell Hayenga and Lipasti Mikko Johnson, Daniel R. Pitfalls of orion-based simulation. In *Workshop on Duplicating, Deconstructing, and Debunking*, 2012.

[38] Daniel U. Becker. Efficient microarchitecture for network-on-chip routers. *PhD thesis, Stanford University, August 2012*.

[39] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, February 2002.

[40] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, 2005.

[41] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, and T.M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 163–174, 2009.

[42] SPEC OMP2001. Available at `http://www.spec.org/omp/`.

[43] Shuai Che, M. Boyer, Jiayuan Meng, D. Tarjan, J.W. Sheaffer, Sang-Ha Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload*

Characterization, 2009. IISWC 2009. IEEE International Symposium on, pages 44–54, Oct 2009.

[44] Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita Das. Aergia: Exploiting packet latency slack in on-chip networks. *IEEE Micro. Special Issue: Micro's Top Picks from 2010 Computer Architecture Conference*, February 2011.

[45] Ragavendra Natarajan and Antonia Zhai. Leveraging transactional execution for memory consistency model emulation. *ACM Trans. Archit. Code Optim.*, 12(3), August 2015.

[46] M.B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal. Scalar operand networks: on-chip interconnect for ilp in partitioned architectures. In *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*, Feb 2003.

[47] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, September 2007.

[48] Jieming Yin, Pingqiang Zhou, Sachin .S. Sapatnekar, and Antonia Zhai. Energy-efficient time-division multiplexed hybrid-switched noc for heterogeneous multicore systems. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 293–303, 2014.

[49] Jason Cong, Michael Gill, Yuchen Hao, Glenn Reinman, and Bo Yuan. On-chip interconnection network for accelerator-rich architectures. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, 2015.

[50] Jason Power, Joel Hestness, Marc Orr, Mark Hill, and David Wood. gem5-gpu: A heterogeneous cpu-gpu simulator. *Computer Architecture Letters*, 13(1), Jan 2014.

[51] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. Reactive nuca: Near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 184–195, 2009.

[52] Blas A. Cuesta, Alberto Ros, María E. Gómez, Antonio Robles, and José F. Duato. Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 93–104, 2011.

[53] Abhayendra Singh, Satish Narayanasamy, Daniel Marino, Todd Millstein, and Madanlal Musuvathi. End-to-end sequential consistency. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 524–535, 2012.

[54] Scott Owens, Susmit Sarkar, and Peter Sewell. A better x86 memory model: X86-tso. In *Proceedings of the 22Nd International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '09, pages 391–407, 2009.

[55] Blake A. Hechtman and Daniel J. Sorin. Exploring memory consistency for massively-threaded throughput-oriented processors. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 201–212, 2013.

[56] Susmit Sarkar, Peter Sewell, Jade Alglave, Luc Maranget, and Derek Williams. Understanding power multiprocessors. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 175–186, 2011.

[57] Sarita V. Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. *Computer*, 29(12), December 1996.

[58] Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum

network on chip. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, DATE '04, 2004.

[59] D.C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P.M. Harvey, H.P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D.L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *Solid-State Circuits, IEEE Journal of*, 41(1):179–196, 2006.

[60] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: A many-core x86 architecture for visual computing. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, 2008.

[61] Intel. Sandy Bridge. http://software.intel.com/en-us/articles/sandy-bridge.

[62] B.K. Daya, C.-H.O. Chen, S. Subramanian, Woo-Cheol Kwon, Sunghyun Park, T. Krishna, J. Holt, A.P. Chandrakasan, and Li-Shiuan Peh. Scorpio: A 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 25–36, 2014.

[63] Govindan Ravindran and Michael Stumm. A performance comparison of hierarchical ring- and mesh- connected multiprocessor networks. In *Proceedings of the 3rd IEEE Symposium on High-Performance Computer Architecture*, 1997.

[64] V. Carl Hamacher and Hong Jiang. Hierarchical ring network configuration and performance modeling. *IEEE Transactions on Computers*, 50(1), 2001.

[65] John Kim and Hanjoon Kim. Router microarchitecture and scalability of ring topology in on-chip networks. In *Proceedings of the 2Nd International Workshop on Network on Chip Architectures*, pages 5–10, 2009.

[66] Thomas William Ainsworth and Timothy Mark Pinkston. On characterizing performance of the cell broadband engine element interconnect bus. In *Proceedings of the First International Symposium on Networks-on-Chip*, 2007.

[67] Chris Fallin, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, Greg Nazario, Reetuparna Das, and Onur Mutlu. HiRD: A Low-Complexity, Energy-Efficient Hierarchical Ring Interconnect, 2012.

[68] Chris Fallin, Chris Craik, and Onur Mutlu. Chipper: A low-complexity bufferless deflection router. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 144–155, 2011.

[69] John Kim. Low-cost router microarchitecture for on-chip networks. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 255–266, 2009.

[70] Heterogeneous system architecture: A technical review. http://developer.amd.com/wordpress/media/2012/10/hsa10.pdf.

[71] Mario Badr and Natalie Enright Jerger. Synfull: Synthetic traffic models capturing cache coherent behaviour. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, pages 109–120, 2014.