ARCHITECTURAL SUPPORT FOR EFFICIENT COMMUNICATION IN FUTURE MICROPROCESSORS

A Dissertation

by

YU HO JIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2009

Major Subject: Computer Engineering

ARCHITECTURAL SUPPORT FOR EFFICIENT COMMUNICATION IN FUTURE MICROPROCESSORS

A Dissertation

by

YU HO JIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Eun Jung Kim
Committee Members,	Duncan M. H. Walker
	Rabi N. Mahapatra
	A. L. Narasimha Reddy
Head of Department,	Valerie E. Taylor

May 2009

Major Subject: Computer Engineering

ABSTRACT

Architectural Support for Efficient Communication
in Future Microprocessors. (May 2009)
Yu Ho Jin, B.S., Korea Advanced Institute of Science and Technology;
M.S., Korea Advanced Institute of Science and Technology
Chair of Advisory Committee: Dr. Eun Jung Kim

Traditionally, the microprocessor design has focused on the computational aspects of the problem at hand. However, as the number of components on a single chip continues to increase, the design of communication architecture has become a crucial and dominating factor in defining performance models of the overall system. On-chip networks, also known as Networks-on-Chip (NoC), emerged recently as a promising architecture to coordinate chip-wide communication.

Although there are numerous interconnection network studies in an inter-chip environment, an intra-chip network design poses a number of substantial challenges to this well-established interconnection network field. This research investigates designs and applications of on-chip interconnection network in next-generation microprocessors for optimizing performance, power consumption, and area cost. First, we present domain-specific NoC designs targeted to large-scale and wire-delay dominated L2 cache systems. The domain-specifically designed interconnect shows 38% performance improvement and uses only 12% of the mesh-based interconnect. Then, we present a methodology of communication characterization in parallel programs and application of characterization results to long-channel reconfiguration. Reconfigured long channels suited to communication patterns enhance the latency of the mesh network by 16% and 14% in 16-core and 64-core systems, respectively. Finally, we discuss an adaptive data compression technique that builds a network-wide fre-

quent value pattern map and reduces the packet size. In two examined multi-core systems, cache traffic has 69% compressibility and shows high value sharing among flows. Compression-enabled NoC improves the latency by up to 63% and saves energy consumption by up to 12%.

To My family

ACKNOWLEDGMENTS

This dissertation is dedicated to my parents. They were always with me when I started the PhD journey in Texas. I am deeply honored by their patience, understanding, and commitment throughout my life. I am in debt for their tremendous moral and financial support.

I would like to thank my two sisters and non-Texas A&M friends. They have provided many experiences and perspectives to last a lifetime. They always supported my endeavors, never doubted my abilities, and deserve much thanks.

My education at Texas A&M University, College Station has been a life-changing experience. I would like to thank my advisor Eun Jung Kim for her many years of patience, wisdom, and encouragement. I sincerely appreciate all her advice and anecdotes, and she has certainly made my graduate experience an enjoyable one. I thank Duncan Walker, Rabi Mahapatra, and Narasimha Reddy. I appreciate their time and advice on all aspects of my research, and for providing insightful feedback on my research and presentations. I also thank Ki Hwan Yum for providing excellent technical advice and teaching me essential skills for good writing.

I thank the numerous students in High-Performance Computing Lab, where a multitude of people with lots of talent come together for a common purpose. I am grateful to have met a wonderful group of people, including Hogil Kim, for listening to my various rants and raves over the years, Manhee Lee, for encouraging me in various ways, Heungki Lee, for always showing confidence in me, Minseon Ahn, for helping me jump into a new simulation tool, and, Baiksong An, for arranging everything in a perfect way during hectic examination times. I also thank other students, Inchoon Yeo, Lei Wang, Babatunde Azeez, Gopinath Vageesan, Varrian Hall, Jay Iyer, Sungho

Park, Chih-Chun Liu, and Poornachandran Kumar. They have been always good colleagues in the lab.

I thank the professors that influenced me to pursue the field of Computer Science in KAIST and Texas A&M University. Jin Hyung Kim, my master degree advisor, offered my first opportunity for conducting research. Lastly, I thank Jian Li for his mentoring and stimulating my work when I had internship in IBM Research.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

The advancement of microprocessors is driven by Moore's Law, which predicts that the number of transistors on microprocessors doubles every two years. The substantial performance improvement for the past decade has been mined from increasing clock frequency, which currently almost stopped due to power consumption constraint. In these scaling trends, current microprocessor designs attain performance from parallel processing by integrating a large number of cores in a single die.

Achieving performance for multi-core or many-core has embarked on a design paradigm shift from computation-centric to communication-centric aspects. Future chip multiprocessors (CMPs) that have tens or hundreds of cores and multi-megabyte on-chip caches will require a communication substrate interconnecting components and coordinating data exchange among them. Without well-designed interconnect, the multi-core system will lose the design merits and deteriorate performance advantage.

A. Networks-on-Chip

Traditional interconnects such as shared buses and dedicated wiring suffer from scalability as well as increase complexity of chip designs [1, 2]. Networks-on-Chip (NoC) is a proposal to use a switched network by abstracting a communication unit as a packet. NoC is used for a communication architecture in next-generation chips such as Intel Teraflop 80-core [3], Tilera 64-core [4], RAW [5], and TRIPS [6].

Though NoC is conceptually similar to the chip-to-chip network, NoC has dif-

---

This dissertation follows the style of *IEEE Transactions on Computers.*

ferent characteristics for chip-wide communication support. First, the end-to-end latency is ultra-low, taking a few clock cycles per hop. Routers and links share a high frequency clock with processors. Next, abundant metal resources enable wide-channel links and thus achieve high-bandwidth. Increasing metal layers in future technology may provide another opportunity for high-bandwidth networks. Finally, the cost of NoC is constrained in terms of power and area. In fact, NoC power consumption is significant at 28% of the tile power in Teraflop [3] and 36% of the total chip power in 16-tile RAW chip [5]. In the operand network of TRIPS, the router takes up 10% of the tile area mostly due to FIFO buffers [7].

The design of a low-latency, high-bandwidth, low-power, and area-efficient NoC can be extremely challenging, because different objectives conflict with each other. Generally, high-bandwidth networks achieve low latency by using much resource but tend to consume more power and take more area. In other words, NoC design must be carefully considered for better performance with power and area envelopes of a chip. NoC must be co-designed with other chip components and its design must be evaluated in a total system perspective. Application-driven workload is essential to compare different designs and show the NoC effect on the system. Variability such as injection rates and traffic patterns in real applications also provides excellent opportunities for implementing adaptive hardware in NoC.

B. Dissertation Contributions

This section describes the research contributions of the dissertation. Although each contribution targets a differently configured system, the developed techniques easily adapt to other designs.

1. NoC Designs for Large Cache Systems

Chapter III develops a domain-specific NoC design in a large-scale cache systems. Taking a network-oriented approach to a last-level cache design reduces a growing wire delay in a cache access time [8]. The monolithic cache is broken into multiple banks that can be accessed at different latencies through an NoC. NoC plays a key role in managing cache operations as well as determining the cache access latency. However, using a general-purpose NoC to a specific problem domain may cause large network delay and under-utilization of network resources, since the network is not optimized for the domain. Reducing both latency and resource of the network requires re-examination of NoC design, particularly how NoC interacts with the rest of the multi-bank cache architecture.

The primary contribution includes: (i) a single-cycle router architecture with multicast support as the basic building block of the interconnection networks; (ii) Fast-LRU replacement that can reduce the network latency; (iii) appropriate deadlock-free XYX routing algorithm that requires no horizontal links in a mesh except the first row to save area and power; (iv) a new network topology, called a *halo* network, where the MRU banks are of the same distance from the core; and (v) a halo network with non-uniform sized banks, thus reducing the wasted area on the processor die.

2. Communication Characterization Towards Reconfigurable NoCs

For NoCs towards CMPs, characterizing communication in multi-threaded applications can bring many new optimization opportunities. Assessment of real application behavior in NoC is still in nascent stage. There are no standard benchmarks for NoC study, and even previous work depends on results of synthetic traffic from outdated applications. Therefore, understanding communication in applications is a first

step to design efficient interconnects. Scientific and commercial workloads are good candidates for CMP applications. Chapter IV presents a characterization method in OpenMP-style parallel applications. The OpenMP programming model achieves performance enhancement mainly by exploring parallelism in a loop. Furthermore, every loop iteration exhibits quite consistent behavior. Communication behavior is understood as multiple phases, where each phase represents a unique spatio-temporal property. This method can capture time-varying traffic variation of one loop iteration in the same application. We explore the recurring behavior of loop execution and use this property to reconfigure an interconnect.

We build a traffic model for the fixed interval of executed instructions. We construct a traffic matrix, where each element has the amount of traffic for one flow, and each row and column specifies source and destination address, respectively. The matrix is regarded as a feature vector for grouping similar intervals as a phase. We use a machine learning technique that clusters similar intervals and classifies each interval as a phase. After this step, one loop iteration behavior is summarized as a sequence of phases.

In each phase, we identify hot flows or hot sources that require a large volume of communications. Additionally, we identify long-distance communications based on a physical distance between a source and a destination. We use this information in a hybrid network, which has a well-designed topology with multiple long channels to overcome limitation of the physical topology. These long channels are dynamically reconfigured to reduce the packet latency by allocating each of them to a router and hence providing an express path.

3. Adaptive Data Compression in NoCs

In cache traffic for cache-coherent multi-core systems, a packet data that includes a long cache line exhibits frequent bit patterns [9, 10, 11, 12]. Exploiting this property enables a long packet payload to be compressed (encoded) as a short index, hence reducing a packet size. Packet compression increases the designed network bandwidth at runtime, and reduces power consumption by resulting in less switching activity in routers and links.

Chapter V presents the main contributions as co-designing compression architecture with NoC architecture to enhance performance and power consumption with a small cost of the compression hardware. First, we propose the shared table scheme, which stores identical values into a single entry from different sources or destinations and removes the network-size dependence. We also present the efficient table management protocol for consistency. Next, we propose performance improvement schemes. Streamlined encoding reduces the encoding latency by overlapping encoding with flit injection. Finally, we present the dynamic compression management that compresses packets on-demand in varying workloads to maximize the performance.

We identify 69% compressibility of cache traffic in a suite of scientific and commercial benchmarks for two CMP designs; 8-core with NUCA and 16-core tiled. Additionally, we observe that a large portion of communication data is shared among different flows. Finally, the detailed simulation results show that data compression improves the latency by 55% and saves the energy by 12%. Furthermore, dynamic compression management achieves 63% latency improvement with 7% energy saving.

C. Dissertation Organization

This dissertation is organized as follows. Chapter II presents architectural components for NoC and examines exploratory NoC-enabled chips. Chapter III presents NoC design consideration in wire-delay dominated cache systems. Chapter IV presents traffic characterization methods with its use for long-link reconfiguration. Chapter V develops data compression for two CMP systems, followed by the conclusion of the research in Chapter VI.

CHAPTER II

BACKGROUND: NETWORKS-ON-CHIP

A. NoC Building Blocks

1. Topology

The network topology determines the various characteristics of the network, such as the average hop count and bisection bandwidth. Because every topology needs to be laid out on 2D die, regular networks such as mesh, torus, or ring are mostly adopted for NoCs. Ring is a simple topology for moderately-sized networks, but the linear hop count dependency and poor bandwidth support make it undesirable for large networks. Torus provides the higher bisection bandwidth and the smaller average hop count than mesh, but power dissipation goes up to almost two times of mesh due to the long links [13]. High radix topology networks which have express channels, such as butterfly [14] or fat trees, require high wiring resource and increase router complexity for a large number of ports. Moreover, it is shown that no single network provides optimal performance across different traffic patterns [15].

2. Router

The router design plays a vital role in switched networks. The pipelined router uses wormhole switching for small buffer requirement, virtual-channels (VCs) to alleviate Head-Of-Line (HOL) blocking, and credit-based flow control. The pipeline stages of a conventional router consist of route computation (RC), virtual channel allocation (VA), switch allocation (SA), and switch traversal (ST) [16]. First, RC stage directs a packet to the proper output port of the router by looking up the packet destination address. Next, VA stage allocates one available VC of the downstream router deter-

mined by RC. SA stage arbitrates input and output ports of the crossbar, and then successfully granted flits traverse the crossbar (ST). After ejecting from the router, the flit crosses over a link to the next router (LT). Therefore, for each hop, head flits require four cycles (RC+VA+SA+ST), while middle/tail flits require two cycles (SA+ST).

Even though an aggressive design can merge adjacent operations into a single-cycle operation, this dependency still exists. Moreover, recent results show that the operating clock cycle for the router is 12 fanout-of-four (FO4) delays [17], which is close to the optimal pipeline delay of the modern superscalar processor [18]. To minimize and break the serial dependency, we use the following techniques.

- **Lookahead routing:** The routing decision is made one hop ahead of the current router. It eliminates the routing delay by removing the serial dependency between routing and VC allocation like the SGI SPIDER chip [19]. The routing outcome is stored in a flit and hence used to allocate VC or the switch output port in the next router.

- **Buffer bypassing:** If the input buffer designated for the VC is empty, a flit directly goes to the crossbar or the VC allocator (if it is a head flit) through a bypass path without being stored at the tail entry of the input buffer. It can remove the delay for the read and write operations of buffer.

- **Speculative switch allocation:** Switch allocation is performed speculatively at the same instance of VC allocation so that a head flit enters into the crossbar right after VC allocation [20]. This speculative switch allocator is only applied to head flits and cannot be granted if the normal switch allocator reserves all the available ports. In this way both VC and switch allocations can be performed at the same cycle.

- **Arbitration precomputation:** Arbitration of competing requests for a VC or a switch port is precomputed and stored for the next arbitration after one output VC in VC allocator or input/output port of the crossbar in switch allocator is granted [17]. The grant signals of the arbiter are generated as the product of the precomputed grant enable signals and incoming arbitration requests. Hence arbitration outcomes are prepared one cycle ahead and latched for the next cycle.

All these techniques except lookahead routing work well in a lightly loaded network where the possible cases for each scheme occur frequently. Our router design maximizes these chances and eventually delivers a flit in a single clock cycle by reducing the critical path in a traditional pipelined router.

Router buffers are instrumental in the NoC design, because the buffer space takes up silicon area and dissipates the significant portion of the router power. Though wormhole switching makes a buffer size less than a packet size, it lowers down buffer utilization. Deadlock avoidance routing algorithms that separate flows into different VCs also incur imbalance of VC utilization for some traffic patterns. Sharing buffer space [21] and controlling the number of VCs [22] reduce expensive cost of the buffer.

3. Link

Links that connect neighbor routers use parallel global wires. Compared with small pin bandwidth limitation in off-chip networks, multi-layer fabrication processes enable the use of wide links in on-chip networks. However, high wiring density will give less space for logic such as cores or caches. Because long wires are implemented as buffered wires with repeaters and latches, the global long wires are difficult to be routed over logic [23]. Over-providing metals to the links for an on-chip network may

easily exhaust top-level metal resources, hence making it hard to route and place power/ground interconnects.

The number (h_{opt}) and sizing (k_{opt}) of repeaters and one segment delay (τ_{opt}) are determined as following RC equations [24].

$$h_{opt} = \sqrt{\frac{2r_s(c_0 + c_p)}{r_w c_w}} \qquad k_{opt} = \sqrt{\frac{r_s c_w}{r_w c_0}} \qquad \tau_{opt} = 2\sqrt{r_s c_0 r_w c_w}(1+\sqrt{\frac{1}{2}(1 + \frac{c_p}{c_0})}) \quad (2.1)$$

where c_0, c_p, and r_s are the input capacitance, the output capacitance, and the output resistance of the minimum size repeater, and c_w and r_w are the unit length capacitance and resistance of the wire. Then, the delay of uniformly buffered wire for a given length L is $\log 2 \times \tau_{opt} \times \frac{L}{h_{opt}}$. Also power consumption of a buffered wire comes from charging and discharging both wire and repeater capacitances. For V_{dd} supply voltage, unit length for a buffered wire consumes the following power (E_w).

$$E_w = 0.5V_{dd}^2(\frac{k_{opt}}{h_{opt}}(c_0 + c_p) + c_s + c_i) \quad (2.2)$$

We model the wire capacitance (c_w) into two parts: wire-substrate capacitance (c_s) and inter-wire coupling capacitance (c_i).

We obtain the wire property from ITRS [2] and PTM model [25], and the repeater property from [26]. Table I shows interconnect property from 65nm to 22nm. Table II presents delay, power, and area of global wire for each technology generation.

B. NoC Applications

Several forms of NoC designs have been proposed to interconnect processing components on a die. *Tile networks* have high scalability to many-core designs because an identical tile having a core and a SRAM memory can be replicated easily [3, 4]. *Cache networks* are another trend in a large-scale L2 cache design to mitigate increased wire

Table I. Global Interconnect Characteristic

Year	2007	2010	2013	2016
Technology	65nm	45nm	32nm	22nm
supply voltage, Vdd (V)	1.1	1.0	0.9	0.8
clock freq. (GHz)	6.73	11.51	19.3	28.8
pitch (nm)	210	135	96	66
aspect ratio	2.3	2.4	2.5	2.6
width (nm)	105	68	48	33
spacing (nm)	105	68	48	33
thickness (nm)	241.5	163.2	120	85.8
height(ILD) (nm)	241.5	163.2	120	85.8
dielectric permittivity	2.85	2.65	2.25	2.15
Rw (Ohm/mm)	867.593	1982.41	3819.444	7770.007
c_s (fF/mm)	21.955	19.569	15.948	14.647
c_i (fF/mm)	81.354	79.297	70.425	70.251
$c_w = 2c_s + 2c_i$ (fF/mm)	206.618	197.732	172.746	169.796

delay and share a large capacity effectively [27, 28]. *Bypass networks* are intended to connect multiple ALUs to overcome increased complexity and delay of bypass path in microprocessors [5, 6]. Communication in tile networks or cache networks are cache traffic that transports cache lines and manages coherence in shared data, while short operand traffic moves in bypass networks.

Table III shows the main NoC features in recent designs. RAW is the proposal to schedule operands in ALU networks and overcome the wire-delay problem for superscalar processors [5]. A mesh network connecting 16 tiles has two physical networks (static and dynamic). The static network controls operands among distributed ALUs, and the dynamic network transports all other traffic such as memory, interrupts, and user-level message passing codes. TRIPS is a distributed processor consisting of multiple tiles connected through two networks [6]. The L2 cache consists of 24 network tiles and 16 memory tiles in on-chip network (OCN), while a processor has 16 execution tiles and other 9 tiles in operand network (OPN). TRIPS prototype shows that an OPN router takes up a significant chip area (10% of the execution tile and 14% of

Table II. Global Interconnect Delay/Power

Year	2007	2010	2013	2016
Technology	65nm	45nm	32nm	22nm
h_{opt} (mm)	0.491	0.318	0.268	0.190
k_{opt}	47.113	38.248	36.794	36.536
one segment wire delay (ps)	59.175	58.086	66.528	66.528
unit length delay (ps/mm)	20.542	182.750	247.896	350.543
unit length delay (FO4/mm)	5.151	11.281	21.519	44.260
unit length dynamic power (mW/mm)	2.045	1.513	1.119	0.869
unit length leakage power (uW/mm)	1.441	1.625	1.777	2.338
unit length sc power (mW/mm)	0.331	0.257	0.214	0.184
unit length total power (mW/mm)	2.378	1.771	1.336	1.056

Table III. NoC Design Parameters in Existing CMPs

Architectures	Topology	Router	Link (bits)	Routing	Flow control	Clock
RAW	2 4x4 meshes	3/5-stage, no VC	256/32	static	credit-based	425MHz
TRIPS OPN	5x5 mesh	1-stage, no VC	142	static (YX)	on-off	400MHz
TRIPS OCN	4x10 mesh	1-stage, no VC	138	static (YX)	credit-based	400MHz
Tilera chip	5 8x8 meshes	1/2-stage, no VC	32 (per mesh)	static	credit-based	1GHz
Cell EIB	4 rings	-	128 (per ring)	minimal	PCS	1.6GHz
Teraflop	8x10 mesh	5-stage, 2 VCs	39	source routing	on-off	5GHz
UltraSparc T1	crossbar	-	103	-	-	1.2GHz

a processor core) mainly because the router has 2.2 kilobits storage for buffers [7].

Tile processor recently announced by Tilera has 5 physical 2D mesh networks to connect 64 cores [4]. The five networks are user dynamic network (UDN), I/O dynamic network (IDN), static network (STN), memory dynamic network (MDN), and tile dynamic network (TDN). The rationale behind this design comes from the benefit of logical networks from independent physical networks, which allows for privilege isolation of traffic, independent flow control, and traffic priotization. Cheap wiring due to multi-layer metal resources makes this realization cost effectively free, while inexpensive buffering (1.1% of tile area for each network) is achieved by excluding use of virtual channels in a router.

Cell processor uses Cell Element Interconnect Bus (EIB) as an on-chip intercon-

nect [29]. In four 16B ring-networks for data bus, two networks are dedicated for data transport of one direction. Cell has an additional shared command bus structured as a tree and a star-like central data bus arbiter. Analytical characterization of EIB shows that the command phase, which coordinates connection-oriented end-to-end transfers, consumes almost 50% of the total zero-load packet latency and that the sustainable network throughput is reduced by a single element causing hot spot or circuit switching.

The Intel Teraflops Processor architecture contains 80 tiles arranged in 2D array and connected by a mesh network [3]. Mesh network provides two virtual channels for instruction and data to prevent short instruction packets from being mixed with long data packets. Experimental results in 65-nm process technology fabrication shows that a 5-port router requires 17% of the total 3-mm^2 tile and its power dissipation is 28% of the tile power. Because this significant network power consumption does not meet the chip power envelope (less than 10% of the total chip power), they concluded that pipeline reduction techniques such as speculation and bypassing in the router are not desirable and fine-grained power management schemes are necessary.

Sun's 32-way Multithreaded Sparc Processor has a single crossbar for communication among Sparc pipes, L2 cache banks, and I/O subsystem [30]. Crossbar interconnect provides 200GB/s bandwidth and queues up to 96 transactions each way. Global arbitration based on age-based priority scheme enables memory ordering for the machine [1].

[1] In Table III, link width is calculated from a 200GB/s bandwidth crossbar and 13 ports (8 Sparc pipes, 4 L2 cache banks, 1 I/O).

CHAPTER III

NOC DESIGNS FOR LARGE CACHE SYSTEMS

A. Overview

With the current rate of technology advancement, increasing wire delays in modern microprocessor designs [31, 1] leads to various techniques to minimize the impact of slow on-chip communication. Typically, on-chip communication has been conducted via shared buses or dedicated wires. Dedicated wires can provide the best customization to applications. However, these interconnects are influenced by various parasitic capacitance and crosstalk from adjacent wires that cannot be predicted until the actual layout and routing are performed [32]. For shared buses, a single communication exclusively uses the whole bus even if multiple communications could operate simultaneously on different parts of the bus. Using global buses is not a scalable solution because the bus bandwidth may become a major bottleneck as the number of chip components increases.

Another way to design an on-chip communication is with a switched network. All the components are connected to the network that routes packets among them, which has the advantages of structure, performance, and modularity [33, 34, 35, 36]. There has been much research on the architectures of future chip multiprocessor (CMP) designs [37, 6, 38] using switched networks for better scalability and resource sharing. Furthermore, these networks have been adopted to overcome wire delay in specific domains such as large scale cache designs [8, 27].

The regular topologies, such as meshes and tori, have been used in on-chip network designs. However, a general-purpose network with regularly distributed network resources can cause problems in the following two cases: under-provisioning and over-provisioning of network resources. Under-provisioning network resources below com-

munication requirement causes poor performance. On the other hand, when network resources are over-provisioned, underutilization of the network resources occurs and large network delays are caused by the increased network size. Furthermore, over-providing network resources results in wasting the chip area. Therefore, it is critical to design an optimal network for a specific domain by breaking the regularity of the interconnection network. It is also important to exploit the potential parallelism of the interconnection networks in the problem domain. Achieving these two goals requires specific knowledge of both interconnection networks and the application domain.

In some large scale cache designs [8, 27], 2D mesh networks have been adopted to interconnect small cache banks to overcome wire delays. For example, in Non-Uniform Cache Architectures (NUCAs) [8], the cache is broken into multiple banks that can be accessed at different latencies through an on-chip network. D-NUCA (Dynamic NUCA) allows cache blocks to *migrate* among cache banks in such a way that recently accessed cache blocks can move towards the core, which helps reduce the average cache access time. However, the network delay is still a dominant portion of the cache access time. A 16MB D-NUCA using a 16×16 mesh network demonstrated an average access time of 17 cycles without network contention while the bank access time is only 3 cycles. The worst case is when the requested cache block is not found in the L2 cache. In this case, all the cache banks in the bank set [1] must be checked sequentially before memory is accessed. In addition, 20% of the total links in a mesh network are never used, while the network occupies 41% of the total cache area.

The main purpose of this research is to investigate the design in the interconnection framework and, particularly, how it interacts with the rest of the multi-bank cache architecture. The research proceeds as follows: first, we propose a single-cycle

[1] To implement a set associative cache in a networked cache system, a set is distributed across multiple banks. This is named a bank set [8].

wormhole router architecture that supports multicast efficiently. Because multicast can significantly reduce large network delay, it decreases the total cache access time dramatically. Unlike the existing multicast routers proposed before, the proposed router takes only one cycle in each hop and does not require any extra storage. It becomes the basic building block of the interconnection network design for the proposed large scale cache systems.

Next, we present Fast-LRU replacement in which cache replacement occurs concurrently with tag-matching. We investigate detailed operations in the network including tag-matching, replacement, and placement for a cache hit and miss. Fast-LRU can be further improved by parallelizing required operations with multicast support. The proposed networked cache system performs best when it uses Multicast Fast-LRU replacement.

Finally, we propose a deadlock-free XYX routing algorithm and a new *halo* network topology to reduce the cache access time and minimize the number of links in the cache system. We discuss how the L2 cache layout on the processor die can help reduce the cache access time. In the XYX routing algorithm, the normal XY routing is used for delivering cache requests from the core to the banks while the replies from each bank to the core are transfered in the Y direction first. Compared with XY routing, the XYX routing algorithm saves most horizontal links in a mesh network. Removing horizontal links leads to the removal of all the associated input buffers and the simplification of both arbiter and crossbar designs. Incorporating small radix routers also results in short pipeline stage latency as well as small chip area. With the halo topology, all the closest banks of each column in the mesh network are placed in the same one-hop distance from the core.

Simulation results with SPEC2000 benchmarks show that the networked cache system with all of the proposed techniques improves the IPC by 38% and uses only

12% of the interconnection area of the D-NUCA system with Multicast Promotion replacement [8]. Specifically, Multicast Fast-LRU replacement improves the IPC by 20% in the mesh network and the halo network further improves it by 18%. We also present analytical models for different communication network designs, which further verifies the performance improvement shown through simulation.

B. Related Work

Several studies explore the large on-chip cache designs to overcome the wire delay problem. NUCA [8] shows that the traditional large cache based on partitioned subbanking is ineffective because its access latency is determined by the slowest (i.e. farthest from the core) subbank. One of the proposed designs, Static NUCA (S-NUCA), uses dedicated wires to each bank and incurs significant area overhead. In another Dynamic NUCA (D-NUCA) design that organizes banks with a switched network, they exploited cache block migration, partial-tag search for early miss detection, and multicast for fast bank access. NuRAPID [39], which accesses tag and data sequentially, decouples each placement in separate storages. This flexible management increases energy-efficiency in NUCA by filtering most misses from only tag-array accesses. However, it has an overhead to maintain pointer structures in the existing bank architecture. TLC (Transmission Line Caches) [40] uses communication medium for banks as a fast transmission line having short inductive-capacitance (LC) delays. Although transmission line can provide express channels, its application is limited due to the high area requirement. Heterogeneous interconnects are used to improve performance for NUCA such as sending critical address bits on fast L-wires for early bank look-up [41]. Wire and router designs are considered in the cache model used in CACTI 6.0 [42]. Additionally, some of those designs were examined for cache sharing environment in CMPs [27, 43, 28].

An on-chip network that enables low-latency from its high bandwidth is used in partitioned architectures such as CMPs [37, 6] and Networks-on-Chips (NoCs) [33, 34, 35, 36]. A few research showed that the network design must be tailored to the communication behavior of applications running on the network for resource saving and performance improvement. A mesh network was optimized to each parallel application by analyzing its requirement of both links and routers [44]. Instead of assigning buffering resources uniformly in a 2D mesh network, the allocation of different buffering size in each channel reduces the implementation cost and increases performance in the NoC design of media applications [45].

Router design directly impacts the performance of the on-chip network. It is difficult to achieve one-cycle flit delivery in a conventional pipelined router due to dependencies between pipeline stages. Speculative switch allocation breaks the dependency chain between virtual channel allocation and switch allocation [20]. A recently proposed low-latency router delivers a flit in a single cycle [17]. It eliminates control overheads (routing and arbitration logic) from the critical path by input buffer bypassing, lookahead routing, and pre-computation of arbitration decisions.

To support one-to-many communication primitives, multicast routers have been designed especially for multistage networks and 2D mesh/hypercube topologies [46, 47]. Due to the nature of multicast, we need to provide extra storage to hold packets and a deadlock prevention mechanism. In a chip-to-chip domain, the central-buffer-based design showed better performance over the input-buffer-based design, because the queuing capability of the central buffer is superior for unicast packets [46]. However, the additional storage is not desirable in the on-chip domain where area budget is very tight. There are two ways to prevent deadlock: deadlock avoidance/recovery and complete packet buffering. While deadlock avoidance routing causes inefficient resource utilization and deadlock recovery mechanisms are highly complex, complete

buffering requires large buffer storage. Therefore, the main challenge of multicast support in an on-chip network is to prevent deadlock without extra storage requirements.

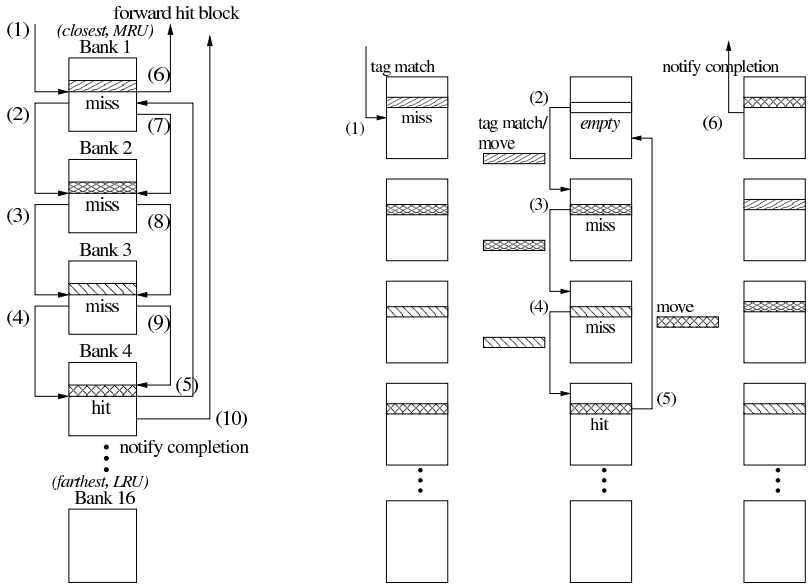
C. Designing Communication for Cache Architecture

In this section, we explain communication operations in mesh-connected banks to manage blocks for LRU replacement. We present Fast-LRU replacement that merges tag matching and replacement operations in a single packet. During the course of this work, fast LRU replacement is extended with a multicast network. Finally, we develop an analytical performance model for different communication designs.

1. Fast-LRU Replacement

In this section, we propose *Fast-LRU replacement* with multicast to reduce the long latency of the LRU replacement scheme after examining its optimization with unicast.

We briefly explain cache operations in D-NUCA [8]. As shown in Figure 1 (a), the cache is broken into multiple banks that can be accessed through a mesh network. One column of the mesh represents a set of a set-associative cache, which is statically determined by the low-order bits of the block address. Cache blocks in a set are spread across multiple banks in the column. Each set distributed in a column is called a *bank set*. Thus, the cache system searches for a block by first selecting the column, selecting the set within the column using direct-mapping, and finally performing a tag-match on distributed blocks. Note that each way has a different network latency depending on the distance from the core. Although a bank set can be distributed on every column to give approximately equal access time across all bank sets, we do not consider this configuration due to the larger hop count.



(a) LRU Replacement (b) Fast-LRU Replacement

Fig. 1. Two Unicast LRU Replacement Schemes in a Networked Cache

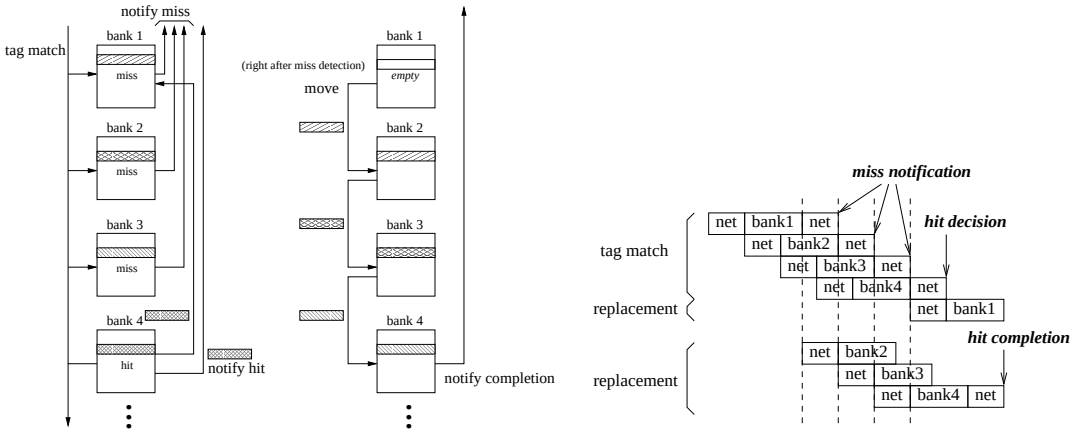
To reduce the average access time, we should place frequently used data in the banks closer to the core, which can be achieved with LRU replacement. The LRU generates 14% higher cache hit rate than Promotion [8], which swaps the hit block with a block in the bank that is next closest to the core. Therefore, it makes the first way and the last way be placed on the closest (MRU) bank and the farthest (LRU) bank, respectively. However, maintaining the LRU order in a bank set requires a large overhead because it incurs many swaps of blocks between banks.

Figure 1 (a) shows the LRU replacement policy by depicting the required communications among banks. Assuming that a data request is a hit in Bank 4, the request traverses from Bank 1 to Bank 4 ((1) $\sim$ (4)). Then the hit block is sent to Bank 1 ((5)), resulting in the corresponding blocks in Bank 1, 2, and 3 being moved to Bank 2, 3, and 4 ((7) $\sim$ (9)), respectively. In this example, the total communication

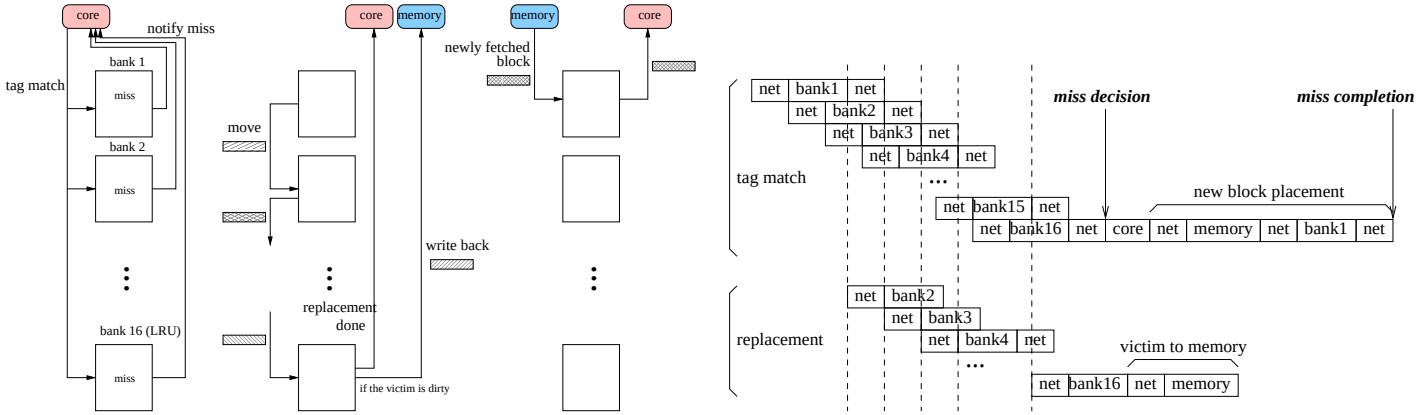
time is 21 hops including the notification of completion; the initial tag-match time to find a hit bank is 7 (1+2+2+2) hops and the remaining part is 14 (4+2+2+2+4) hops. It is clear that the cache hit latency can be decomposed into two parts— tag-match and move (replace) operations. Therefore, the total communication time for block movement after finding a hit can easily exceed the initial tag-match time. A cache miss needs tag-match along all banks and a new block placement to the MRU bank incurs multiple block movements.

In Figure 1 (b), the proposed Fast-LRU replacement allows the tag-match operation to overlap with the replace operation. If there is a miss in a bank, the corresponding block in the bank is evicted and immediately transferred to the next bank with the data request ((1) $\sim$ (4)). Unless the request is a hit in the MRU bank, the corresponding block in the bank is pushed to the next bank consecutively until the final (LRU) bank. Once there is a hit in a bank, that block is transferred to the MRU bank where its corresponding frame is already empty ((5)). If all the banks generate misses, the request is forwarded to the off-chip memory, and the requested memory block is read. Then this new block is stored in the MRU bank and sent to the core. Since the invariant property of LRU is that all the banks ahead of the hit bank generate misses, the total communication time of the Fast-LRU replacement scheme is 12 (1+2+2+2+4+1) hops in Figure 1 (b). In addition, this scheme almost halves the number of bank accesses since both tag-match and replacement are performed simultaneously.

Next, we further investigate how to reduce the cache access time by exploiting the multicast router proposed in Section E. Even though Fast-LRU replacement reduces the hit latency, a hit on the LRU bank or a miss on a cache (i.e. all banks produce misses) still suffers from the long latency, which is the sum of the bank access time over all banks in a bank set and the network latency. Multicast relieves this problem



(a) Cache Hit with Time Diagram



(b) Cache Miss with Time Diagram

Fig. 2. Fast-LRU Replacement with Multicast Support

by allowing concurrent accesses of multiple banks for tag-match.

Figure 2 (a) illustrates a cache hit for Multicast Fast-LRU replacement with the time diagram of each operation. When the multicast router attached to the MRU bank (Bank 1) receives a data request, it forwards the request to two destinations, the MRU bank and the second MRU bank (Bank 2), at the same time. The router attached to the second MRU bank (Bank 2) also forwards the request to the attached bank and the next bank (Bank 3), and so on. If the requested block is found in the MRU bank, no block replacement is required and the core is notified of a cache hit. Otherwise, the MRU bank initiates Fast-LRU replacement by sending its evicted block to the second MRU bank. Each non-MRU bank that experienced a cache miss waits for the evicted cache block from the next closest bank from the core. As soon as it receives the evicted block, it also sends its own evicted block to the next LRU bank. This operation stops at the bank where the requested block is found, since the requested block is sent to the MRU bank, not the next LRU bank. Note that the replacement packet (including an evicted block) can never catch the data request packet and that the data request packet always reaches the LRU bank if the router performs arbitration in a FIFO manner. Each non-MRU bank should not initiate its evicted block transfer to the next LRU bank unless it receives an evicted block from the previous bank.

A cache miss occurs when all the banks in the bank set produce misses as depicted in Figure 2 (b). It incurs the off-chip memory access and write-back from the LRU bank (Bank 16) to the memory, if necessary. The core waits for all the banks to report misses, and then invokes the memory access. When the memory sends a new block to the MRU bank, the MRU bank sends this newly incoming block to the core. The block movement is similar with that of a cache hit, except the LRU bank notifies the core of the replacement completion and writes back the victim to the memory if the

block is dirty.

In this scenario, more optimization techniques can be used to improve the performance further.

- In a cache hit, the hit notification and the hit block can be merged into a single packet. This packet is replicated at the router that is attached to the MRU bank. Then the original packet goes to the MRU bank for LRU replacement while the replica is sent to the core.

- In a cache miss, the core does not have to wait for the miss notifications from all the banks, if some banks have invalid cache blocks. Initially the cache is empty and all the blocks are invalid. As cache blocks are read into the banks, all the invalid blocks are pushed down towards the LRU bank. Therefore, the core receives a miss notification from an invalid block, thus implying that all the banks farther than this bank have invalid blocks. In fact, the core should keep track of miss notifications from all the banks that are closer to the core than the bank that produces a compulsory miss. This can expedite the memory access and deliver the required data to the core earlier.

2. Performance Gain Estimation

We build an analytical model of communications for each replacement policy and estimate the efficiency of Multicast Fast-LRU replacement. It can be used as an early design tool to estimate the networked cache performance. Furthermore, it can be generalized to other replacement policies and be applicable to predict performance for other workloads.

A c-bank, w-way cache system requires a $c \times w$ mesh network. We can break down its access latency into bank access latency (l_b), memory access latency (l_m), and

network latency (l_n). The network latency [2] ($l_n(d)$) is proportional to the hop count, d. It also can be represented as $l_n(i \rightarrow j)$ between two communication entities, i and j, and each entity is one of core (C), banks ($\{b_{x,y} | x \in [1,c], y \in [1,w]\}$), or off-chip memory ($M$). Bank $b_{x,y}$ stands for y-th way in x-th column of the mesh.

We show how the average access time can be estimated in the memory hierarchy including a L2 cache and an off-chip memory with different communication designs.

Average Access Latency: To estimate the average L2 cache access latency (L_P) for a replacement policy (P), we consider hit latency (H_P) and miss latency (M_P) separately. When the request comes into the network, it accesses banks in only one column x of the mesh network. Let $h_{x,y}$ be the hit rate of bank $b_{x,y}$ and $H_P(x,y)$ be the hit latency. Summing up each hit rate of all banks is less than 1 ($\sum_{x=1}^{c} \sum_{y=1}^{w} h_{x,y} \leq 1$). Then, column x's portion for the average access latency is addition of two parts: the sum over each bank's hit latency weighted by its hit rate, and the product of the miss rate and miss latency. The miss rate is computed by subtracting the sum of the hit rate of all banks in column x. The miss rate is later weighted by the ratio of requests going to the column x to total requests, r_x ($\sum_{x=1}^{c} r_x = 1$). We calculate L_P as:

$$L_p = \sum_{x=1}^{c} \left[\sum_{y=1}^{w} (h_{x,y} \cdot H_p(x,y)) + (r_x - \sum_{y=1}^{w} h_{x,y}) \cdot M_p(x) \right]$$

Unicast LRU, Unicast Fast-LRU, and Multicast Fast-LRU policies are denoted as *u-LRU*, *u-fLRU*, and *m-fLRU*.

Unicast LRU: The cache hit on the MRU bank (the first way, $b_{x,1}$), $H_{\textit{u-LRU}}(x,1)$, does not need block movement. Its latency includes a single bank access, l_b, and a network round-trip between core and the MRU bank, which is the sum of $l_n(C \rightarrow b_{x,1})$

[2] For simplicity, we assume that our network is contention free.

and $l_n(b_{x,1} \rightarrow C)$. However, the cache hit on other banks, $H_{\ u\text{-}LRU}(x,y)$, needs multiple bank accesses. The hit on the bank $b_{x,y}$ causes banks from $b_{x,1}$ to $b_{x,y-1}$ to generate misses and each network latency to the neighbor is 2 hops, which is $l_n(2)$. Additionally it needs recursive block movements from the hit bank to the MRU bank to maintain the order for LRU replacement.

$$H_{\ u\text{-}LRU}(x,1) = \underbrace{l_n(C \rightarrow b_{x1}) + l_b + l_n(b_{x,1} \rightarrow C)}_{hit}$$

$$H_{\ u\text{-}LRU}(x,y) = \underbrace{l_n(C \rightarrow b_{x,1}) + l_b + (y-1)\cdot(l_n(2)+l_b)}_{(y-1)\ misses + y-th\ bank\ hit} + \underbrace{l_n(y) + l_b + (y-1)\cdot(l_n(2)+l_b) + l_n(b_{x,y} \rightarrow C)}_{move}$$

The cache miss latency, $M_{\ u\text{-}LRU}(x)$, has three components. It detects cache miss by visiting all w banks, goes to the memory controller to fetch a new block, and then places it to the MRU bank. This placement of a new block incurs additional block movement until the bank does not have a valid block. If the y'-way bank is the last one for movement, the number of additional movements is y'.

$$M_{\ u\text{-}LRU}(x) = \underbrace{l_n(C \rightarrow b_{x,1}) + l_b + (w-1)\cdot(l_n(2)+l_b)}_{w\ misses} + \underbrace{l_n(b_{x,w} \rightarrow M) + l_m}_{memory} + \underbrace{l_n(M \rightarrow b_{x,1}) + l_b + y'\cdot(l_n(2)+l_b) + l_n(b_{x,y'} \rightarrow C)}_{place}$$

Unicast Fast-LRU: Fast-LRU replacement has no extra movements caused by placing the hit block to the MRU bank, because the block is already moved to its neighbor. Furthermore, in case of a miss, when a new block from the memory is placed on the MRU bank, the corresponding frame in the MRU bank is already invalid. Therefore, it does not have y' movements for Unicast LRU policy.

$$H_{\text{u-fLRU}}(x,1) = \underbrace{l_n(C \to b_{x,1}) + l_b + l_n(b_{x,1} \to C)}_{hit}$$

$$H_{\text{u-fLRU}}(x,y) = \underbrace{l_n(C \to b_{x,1}) + l_b + (y-1)\cdot(l_n(2) + l_b)}_{(y-1)\ misses + move + y-th\ bank\ hit} + \underbrace{l_n(y) + l_b + l_n(b_{x,1} \to C)}_{move}$$

$$M_{\text{u-fLRU}}(x) = \underbrace{l_n(C \to b_{x,1}) + l_b + (w-1)\cdot(l_n(2) + l_b)}_{w\ misses} + \underbrace{l_n(b_{x,w} \to M) + l_m}_{memory} + \underbrace{l_n(M \to b_{x,1}) + l_b + l_n(b_{x,1} \to C)}_{place}$$

Multicast Fast-LRU: Because multicast packets for the cache request produce a copy at each router and each of them arrives at a bank concurrently, the hit latency consists of the hit bank access and the round-trip starting at the core. Note that block movements for LRU replacement initiated by the hit bank are not on the critical path compared with the Unicast Fast-LRU scheme. When the miss notification from the last bank arrives at the core, the corresponding request is identified as a miss, and hence, the core directly asks the memory for sending a new block to the MRU bank.

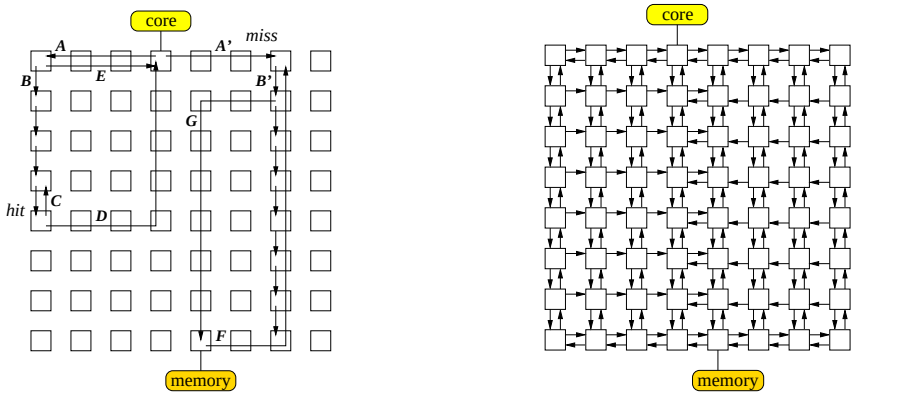
$$H_{\text{m-fLRU}}(x,y) = \underbrace{l_n(C \to b_{x,y}) + l_b + l_n(b_{x,y} \to C)}_{hit}$$

$$M_{\text{m-fLRU}}(x) = \underbrace{l_n(C \to b_{x,w}) + l_b + l_n(b_{x,w} \to C)}_{miss} + \underbrace{l_n(C \to M) + l_m}_{memory} + \underbrace{l_n(M \to b_{x,1}) + l_b + l_n(b_{x,1} \to C)}_{place}$$

D. Interconnection Network Topology and Layout

We analyze the utilization of the interconnection network for large scale cache systems and explore a few alternatives for simplification and optimization. The limited chip resource should be efficiently distributed to both computational and communication units. Over-provisioning causes the waste of valuable chip resources, while under-

provisioning incurs performance degradation. Therefore, the chip design needs a careful analysis of each unit's resource requirement and their interactions to draw the optimal configuration.



(a) Communication Patterns (b) Minimal Link Requirement

Fig. 3. An 8×8 Mesh Network for a Large Scale Cache System

Figure 3 (a) shows all possible XY routing communication patterns of a large scale L2 cache system on an 8×8 mesh network. The core is attached to the fourth router on the top row and the memory is attached to the fifth router on the bottom row. Forwarding a data request to the appropriate bank set needs the traversal of the first row (**A**) and the traversal of banks within a column (**B**) until there is a hit in one bank. When a bank replies the requested cache block to the core, the communication path is either **D** from a non-MRU bank or **E** from a MRU bank. The data movement between two banks occurs only in one column of the mesh (**B** or **C**). While a cache miss has the same communication patterns (**A'** and **B'**) as a cache hit, it fetches a new data block from the memory and places it on the MRU bank (**F**). After this new block placement, an evicted dirty block is replaced with the block in the next farther bank (**B'**) or is written back to the memory (**G**), if dirty.

One observation of these patterns is that a single direction is used in most horizontal links except on the first row, the last row, and links between the core-attached column and the memory-attached column. Figure 3 (b) shows the minimum set of links after removing all the unnecessary horizontal links. In general, we can remove $(n-2)^2$ links among the total $4(n-1)^2$ links of an $n \times n$ mesh, which reduces the link area by 25%.

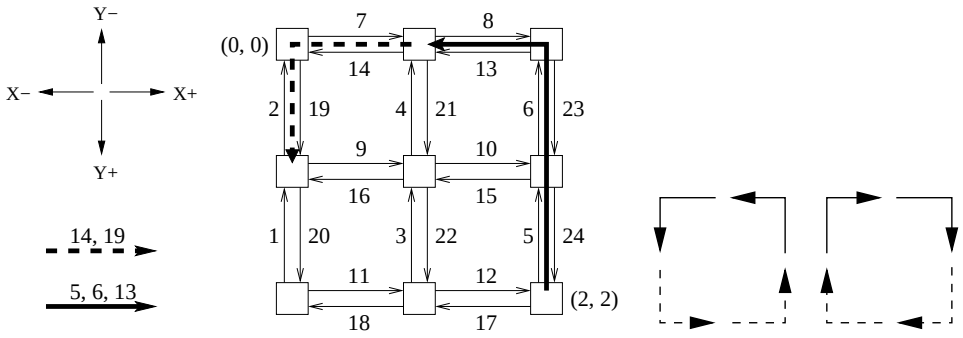
Another observation is that the horizontal links except on the first row, are infrequently used, because they are used only when a bank needs to communicate with the core (i.e. the cache controller) or the memory. Considering the data locality, horizontal link utilization for the last row becomes low because those links are only used for the memory accesses. The number of these underutilized links[3] is $n^2 - 2$.

Inputs: coordinates of source bank(X_{src}, Y_{src})
coordinates of destination bank(X_{dest}, Y_{dest})
Outputs: Selected output $Channel$
Procedure:
$X_{offset} := X_{dest} - X_{src}$;
$Y_{offset} := Y_{dest} - Y_{src}$;
if $Y_{offset} \geq 0$ **then**
if $X_{offset} > 0$ **then** $Channel := X+$;
else if $X_{offset} < 0$ **then** $Channel := X-$;
else if $Y_{offset} = 0$ **then** $Channel := Internal$;
else $Channel := Y+$;
endif
else
$Channel := Y-$;
endif

Fig. 4. XYX Routing Algorithm

They can be eliminated at the expense of small bandwidth loss. This simplified

[3]In a mesh network requiring a minimal number of links, rows from the second to the last second have n vertical links. The last row has $2(n-1)$ vertical links. Therefore, the total number of underutilized links is $n(n-2) + 2(n-1)$.



(a) Totally Ordered Channels (b) Turn Model

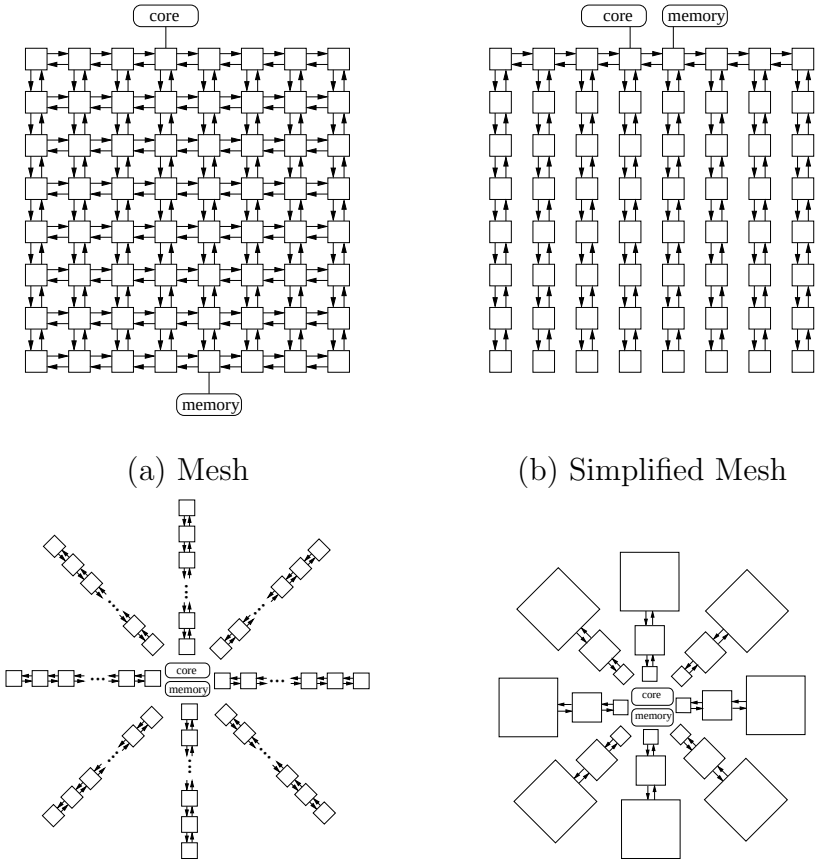
Fig. 5. Deadlock Freedom in XYX Routing

mesh can remove additionally 25% of total links. However, it causes the change of the routing scheme since communications from the banks to the core/memory start in the Y direction first, which violates the XY routing rule. Thus, we propose a new routing scheme called XYX routing to overcome this problem. XYX routing is deadlock-free because we enforce the total order of channels in the mesh network.

Figure 4 shows the XYX routing algorithm for k-ary 2-cubes. The routing function routes packets in the X dimension first and in the Y dimension next for the XY direction or in the Y dimension first and in the X dimension next for the YX direction. Figure 5 (a) and (b) show the deadlock freedom in a 3×3 mesh network. When all the channels are ordered totally in Figure 5 (a), packets that lie on the different channels do not produce a cycle in the network. Any path in XYX routing follows increasingly numbered channels such as two paths, (14, 19) and (5, 6, 13). XYX routing algorithm allows only four turns out of total eight turns in the turn model [48]. Cycles are prevented by prohibiting four of the turns, as shown in Figure 5 (b).

In the view of the router design, expunging horizontal links leads to the elimination of all the input buffers associated with those links and to the simplification of both the arbiter and the crossbar. This simplified router also reduces the pipeline

latency as well as saves its area.



(a) Mesh (b) Simplified Mesh

(c) Halo with Uniform Banks (d) Halo with Non-uniform Banks

Fig. 6. Domain-Specific Development of Network Designs for Large Cache Systems

One of the disadvantages in a mesh network is uneven network latencies for MRU banks depending on the distance from the core. Since there is only one path between the core and one special MRU bank, the leftmost or rightmost MRU banks cannot avoid suffering from long network latency (Figure 6 (a) and (b)). So it is crucial to provide a direct communication path between the core and each MRU bank. For this purpose, we choose a topology in which the core is located in the same distance

from all the MRU banks. We call this topology a *halo* network shown in Figure 6 (c). The core plays a role of a *hub* to control the departure and the arrival of cache requests. A bank set is distributed over multiple banks within a *spike* branched from a hub, which bidirectionally connects all the banks in the order of the way. In this design, we assume that the cache controller supports multiple ports/interfaces to the networked cache. We put a small queue for each spike such as a multiple issue queue. Thus, a cache request is first stored in each spike queue to be subsequently forwarded to the L2 cache. When the size of these queues becomes very high to connect many spikes, multiple spikes can share one queue with an address decoding logic. Spike queue sharing will reduce the cost of queues and reduce the concentration degree to the core/memory, hence achieving better scalability. However, the spike access will be delayed when there are multiple cache requests for different spikes in the queue.

When the size of all banks that build a spike is identical, the banks positioned at the end of the spike cannot fill the increased area entirely in Figure 6 (c). Although we can curve a spike and connect banks within a spiral spike, this layout incurs the longer wire delay than the straight spike layout. If the bank size increases along the spike, we can reduce the wasted area and draw a compact design by tightly integrating banks on a chip die as shown in Figure 6 (d). As a bank is located farther from the core, its size becomes larger and its access time increases due to the increased capacity. Therefore, capacity-increased banks have more than one way. A halo network incorporated with non-uniform size banks has a topological benefit by giving the same access time to all the MRU banks, and it enjoys a better area utilization over a halo network with the uniform size banks. Note that the memory controller is located in the center of the cache. To access the off-chip memory from the memory controller, the wire delay in a halo is longer than in a mesh network.

Mathematical model described in Section 2 can be directly applied to the halo

network with small modification. The required change is that the halo network has zero hop count in x direction due to the direct links for each bankset. Thus all the latency components account for only the y coordinate in the mesh network. Adoption of non-uniform banks in the halo network requires different bank latency l_b and network latency, resulting in an accumulated form for miss latency calculation.

E. Designing Multicast Single-Cycle Router

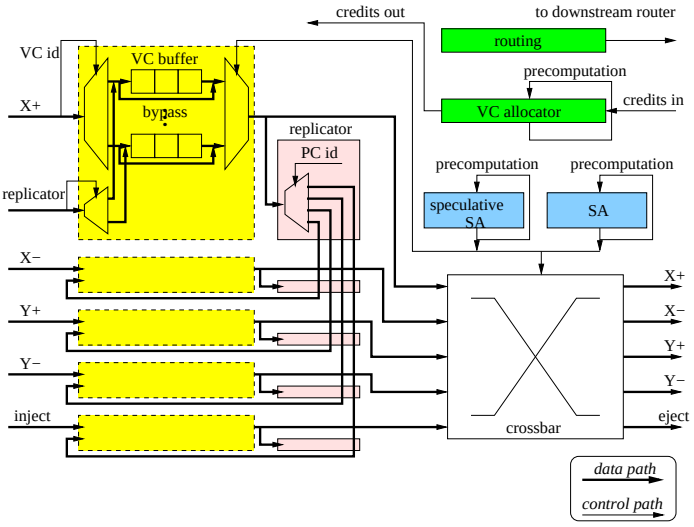


Fig. 7. Single-Cycle Router Architecture with Multicast Support

In this research, we use wormhole routers due to small buffer requirement and high throughput. Figure 7 shows the major components of a wormhole router. It has 5 Physical Channels (PCs) that connect four neighbors and one injection/ejection unit, and each PC is divided into multiple Virtual Channels (VCs). VCs from the same PC share one crossbar port to reduce the crossbar complexity. While middle/tail flits require three operations (input buffering, switch allocation, and switch traversal),

head flits require two additional operations (routing and VC allocation).

Although an aggressive design can merge adjacent operations into a single-cycle operation, this dependency still exists. The design of the router pipeline has a high impact on the internal clock time of the router. Recent results of the different single cycle router designs [17, 49, 50], showed 12 – 35 (fanout-of-four) FO4 delays, which is larger than the optimal pipeline delay (6 – 8 FO4) in the modern superscalar processor [18]. Although increasing complexity of the router pipeline may increase the clock cycle time, the following techniques can break and reduce the serial dependency between different operations, thus achieving a single-cycle router.

- **Lookahead routing:** The routing decision is made one hop ahead of the current router. It eliminates the routing delay by removing the serial dependency between routing and VC allocation like the SGI SPIDER chip [19]. The routing outcome is stored in a flit and hence used to allocate VC or the switch output port in the next router.

- **Buffer bypassing:** If the input buffer designated for the VC is empty, a flit directly goes to the crossbar or the VC allocator (if it is a head flit) through a bypass path without being stored at the tail entry of the input buffer. It also removes the delay for the read and write operations of buffer.

- **Speculative switch allocation:** Switch allocation is performed speculatively at the same instance of VC allocation so that a head flit enters into the crossbar right after VC allocation [20]. This speculative switch allocator is only applied to head flits and cannot be granted if the normal switch allocator reserves all the available ports. In this way both VC and switch allocations are performed at the same cycle.

- **Arbitration pre-computation:** Arbitration of competing requests for a VC or a switch port is pre-computed and stored for the next arbitration after one output VC in VC allocator or input/output port of the crossbar in switch allocator is granted [17]. The grant signals of the arbiter are generated as the product of the pre-computed grant enable signals and incoming arbitration requests. Hence arbitration outcomes are prepared one cycle ahead and latched for the next cycle.

Our router design maximizes these chances and eventually delivers a flit in a single clock cycle by shortening the critical path in a traditional pipelined router. All these techniques except lookahead routing work well in a lightly loaded network where the possible cases for each scheme occur frequently. However, when the router experiences high contention, the router cannot transfer flits in one cycle. For example, when speculative switch allocation is failed, switch allocation and VC allocation are done with different cycles. Additionally, when the switch arbiter does not grant this request due to port contention for other requests, switch allocation cannot be performed at the same cycle of the switch traversal.

We recognize that multicast plays a vital role in deciding a cache hit/miss in the networked cache system to access multiple banks concurrently. Multicast replicates flits inside the router to forward each copy to a destination. *Synchronous replication* copies flits after reserving all destination ports in a lock-step, which easily results in a deadlock situation. *Asynchronous replication* allows the router to forward flits to a subset of the destination ports. While synchronous replication does not require extra buffers, asynchronous scheme needs additional buffers in order to hold flits until all copies are transmitted. We cannot apply either replication scheme to the router design directly for a following reason: With a stringent chip area constraint, synchronous

replication is a natural choice to implement multicast; however, synchronous replication in wormhole switching is susceptible to deadlocks because of the small buffer size. Therefore, we aim to design a replication scheme without extra buffers and to handle each replica independently in an asynchronous manner.

Communication pattern analysis in Section D shows that PCs in some routers within the network are not fully utilized. This property makes us choose a hybrid scheme that exploits the underutilized input buffer space to store replicated flits. The router shown in Figure 7 copies the original flit to one VC of a different PC, when a multicast packet needs replication. A replicator selects a PC that has at least one free VC. When there are multiple PC candidates, least frequently used one is chosen. A free VC of another PC can be easily obtained by checking the input buffer status. If there is no available VC for replication, flit forwarding is blocked. We observe that blocking rarely occurs in the cache systems. This hybrid scheme mitigates the multicast overhead in a low latency router since the changes in the existing VC/switch allocators are not required, and only replication logics are needed to find a free VC of other PCs and connecting wires to their input buffers.

F. Experimental Methodology

We use sim-alpha simulator [51] that models an Alpha 21264 core [52] to generate L2 cache accesses. The clock frequency of the core is scaled to 5 GHz. To measure the contention effect of the banks and the interconnection networks in detail, a separate L2 cache simulator with an interconnection network was developed. Since the input of the simulator is all the L2 cache accesses from the core, the sim-alpha directly sends a chunk of L2 accesses to the L2 simulator. We use Cacti [53] to estimate the latency and the power consumption of a cache bank. The latency and power

consumption of global-level wires is estimated considering the first order RC effect [54] for delay-optimized repeater insertion at 65nm technology. We obtain the resistance and capacitance of the wire from ITRS [55], and the wire length is determined by the bank size. Orion [56] is used to report power consumption for input buffers, crossbars, and arbiters inside the routers. Each stage in the router pipeline takes one cycle. All latency values are converted to the cycle unit. The base configuration is a 16MB L2 cache by interconnecting 256 64KB banks with a 16×16 mesh network. The core is attached at the center of the top row, and the memory is attached at the center of the bottom row to evenly distribute traffic. Main parameters are summarized in Table IV.

Table IV. System Parameters

L1 I & D caches	64B block, 2-way, 64KB, 3-cycle hit
Flit buffer size	4 flits
Number of VCs per PC	4
Flit size	128 bits
Packet size	address packet (1 flit), data packet (5 flits)
Memory	64B block, 130 cycles + 4 cycles per 8 bytes

When we evaluate the different size of banks for the networked cache, we use Table V to model the latency and power consumption per bank access. As the bank size increases, the latency and power consumption of the bank and the link increase together. We determine the link length for the bank from the area of both the bank and the router. In 65nm, a repeated wire in the link exhibits 121 ps/mm for latency and 2.38 mW/mm for power consumption.

Table V. Latencies and Energies for Bank Access

Bank size	Bank		Link	
	Tag matching(+replacement) delay (cycles)	Energy (nJ)	Delay (cycles)	Energy (nJ)
64KB	2 (3)	0.1392	1	0.0285
128KB	4 (5)	0.2207	2	0.0400
256KB	4 (5)	0.3743	2	0.0528
512KB	5 (6)	0.6661	3	0.0759

To measure various design impacts, we use SPEC2000 benchmarks for simulation. We skip 2 billion instructions, warm up the L2 cache for the next 100 million instructions, and measure the performance for remaining instructions. Table VI shows the IPC with the perfect L2 cache and L2 cache access behavior of each benchmark.

A 32-bit address can be divided into 4 fields: tag (12 bits), index (10 bits), bank-column (4 bits), and offset (6 bits). The *bank-column* is used to select one out of 16 columns while the *index* identifies one block in each bank of the selected column. With uniformly sized 64KB banks, the bank is organized as a direct-mapped cache, and the blocks distributed to each bank on the same column form a 16-way bank set.

In the traditional cache design, the cache is connected to two buses: address bus and data bus. Since a cache network delivers a structured data as a packet, the switched network does not need the separate network for address and data. In wormhole switching, converting a packet into flits requires the overhead data [33] for each flit such as flit type (2 bits for specifying head/middle/tail), size (7 bits for the size of flit data), routing (16 bits for source and destination only required for a head flit), and communication flag (1 bit for unicast/multicast). Since the link width

Table VI. SPEC2000 Benchmarks Used for Experiments

benchmark name	instr. exec.	perfect L2 IPC	L2 read	L2 write	L2 accesses per instr.
applu(FP)	500M	0.43	9.444M	4.428M	0.028
apsi(FP)	1B	0.40	12.375M	8.204M	0.021
art(FP)	500M	0.40	63.877M	13.578M	0.155
galgel(FP)	2B	0.43	19.415M	4.137M	0.012
lucas(FP)	1B	0.44	19.506M	13.226M	0.033
mesa(FP)	2B	0.40	2.907M	2.656M	0.003
bzip2(INT)	2B	0.39	16.301M	4.233M	0.010
gcc(INT)	500M	0.29	26.201M	14.827M	0.082
mcf(INT)	250M	0.34	29.500M	15.755M	0.181
parser(INT)	2B	0.38	18.257M	6.915M	0.013
twolf(INT)	1B	0.38	20.283M	7.653M	0.028
vpr(INT)	1B	0.41	12.459M	5.024M	0.017

is 16B, a read request packet or a notification packet that has only 32-bit address needs one flit even with the flit overhead data. When a packet includes a block data for write request, replacement, memory access, or hit data forwarding, one packet consists of 32-bit address, 64B data and overhead bits, which are divided into 5 flits.

G. Experimental Results

We present the L2 cache latency access result and its analysis to inspect the efficiency of communication support for Fast-LRU replacement in Section 1. Impact of various interconnect designs to the overall system performance is examined in Section 2. The area consumption of router, wire, and cache in each interconnect design is analyzed in Section 3.

1. Performance of Multicast Fast-LRU Replacement

Figure 8 shows how the total average cache access latency is divided into bank access, network traversal, and memory access for benchmarks in 16MB L2 cache with uniform

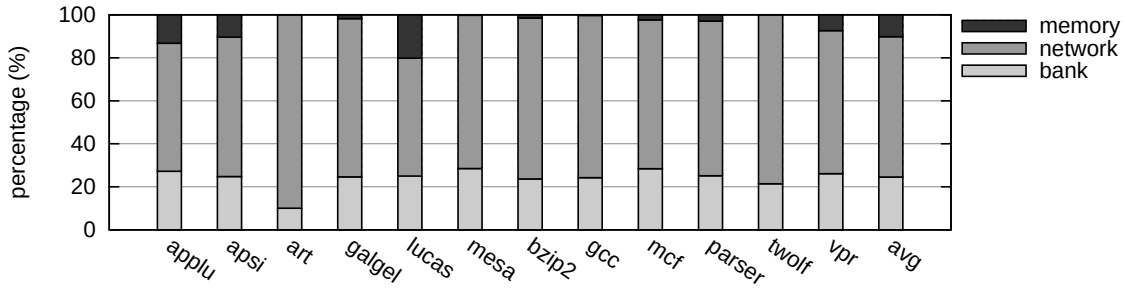


Fig. 8. Latency Distributions of L2 Cache Accesses in the Unicast LRU Environment

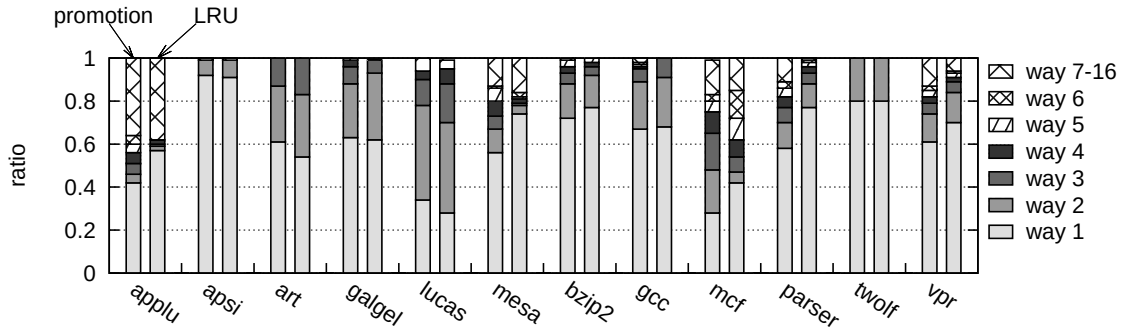
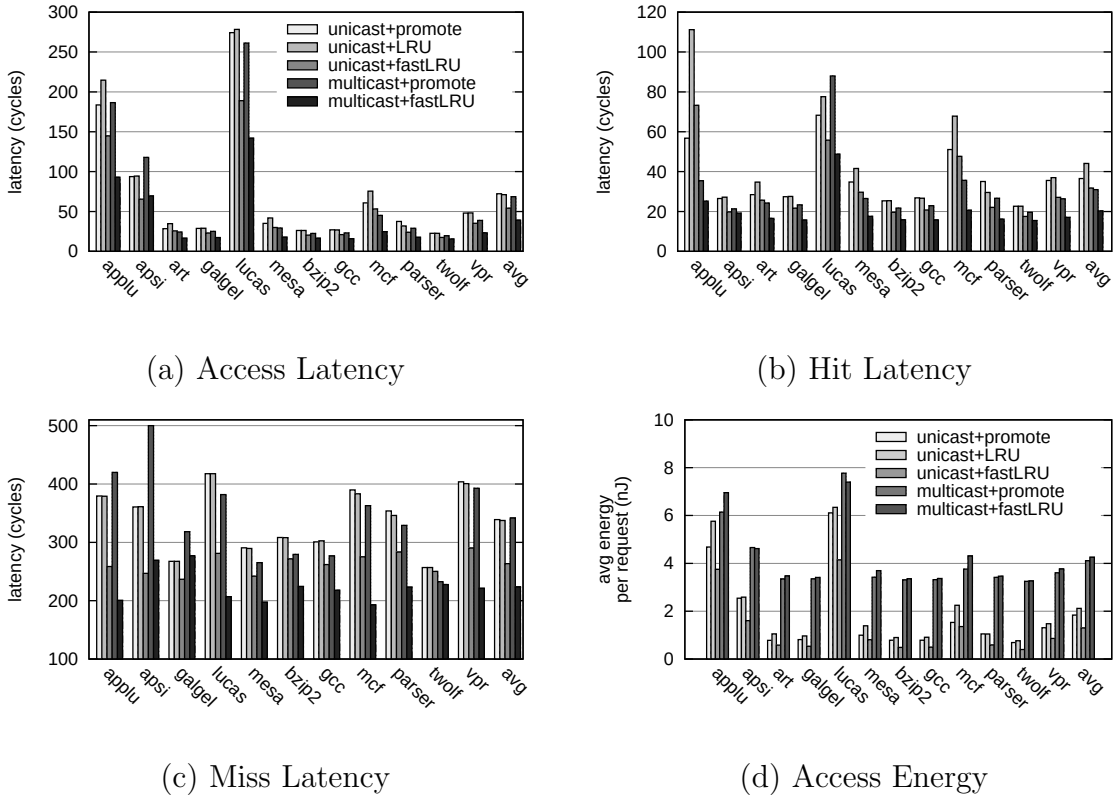


Fig. 9. Way Distributions of Promotion and LRU

size banks. A significant portion of the total average latency is network access (65% on the average) while bank access (25%) and memory access (10%) are relatively small. In Figure 9, we show the way distributions of two different cache replacement schemes, Promotion [8] and LRU. Excluding *art*, *galgel*, and *lucas*, LRU is a better policy than Promotion since more hits occur in the MRU (fastest) banks. Specifically, LRU shows a hit increase by 5-19% at the MRU banks.

In Figure 10, we compare performance results from the Multicast Fast-LRU with Multicast/Unicast Promotion [8] [4] Unicast LRU, and Unicast Fast-LRU. Figure 10

[4]In our implementation of cache miss for Promotion, the incoming block from the memory evicts the data in the closest bank and causes recursive replacement. In [8],



(a) Access Latency

(b) Hit Latency

(c) Miss Latency

(d) Access Energy

Fig. 10. L2 Cache Access Latency and Energy Comparison

(a) illustrates the average access latency while hit and miss latencies are depicted in Figure 10 (b) and (c) [5]. In the unicast environment, LRU naturally increases the average access latency by 4.4% over Promotion, but Fast-LRU reduces it by 30.2%. Recall that Fast-LRU has fewer bank accesses than Promotion, and it concentrates hits to the MRU banks. Multicast Fast-LRU reduces the average access latency by 46% over Unicast LRU and 27% over Unicast Fast-LRU. Also Multicast Fast-LRU reduces the average hit and miss latencies of Unicast LRU by 48% and 32%,

---

the victim block can be directly moved to the memory (zero-copy) or once to the lower-priority bank (one-copy), resulting in losing critical data from the cache.

[5] We omit *art* results in Figure 10 (c) since there is no cache miss except compulsory misses during our simulation.

respectively. Its latency improvement over Multicast Promotion is 37%, and the IPC is improved by 20%. Figure 10 (d) shows energy consumption results for various policies. For unicast, the results show that Fast-LRU outperforms Promotion and LRU by reducing total amount of traffic and the number of bank accesses. For multicast, Fast-LRU increases the average energy consumption by 3% over Promotion. This result occurs mainly because Fast-LRU needs more packets than Promotion to reorganize spread blocks in a column for the LRU order.

Furthermore, if we look at the performance results of individual benchmarks, we have more interesting observations. In *lucas*, which has the lowest hit rate (0.41) among all the benchmarks, the hit latency in Multicast Promotion is increased by 29% over Unicast Promotion because hit requests should compete with increased number of operations to place incoming block from the miss requests. In *mcf* and *parser*, Multicast Fast-LRU shows almost 59% and 53% reduction of the average latency over Unicast Promotion because it increases the hit rate of MRU banks by 48% and 33% and decreases the hit latency by 60% and 54%.

2. Performance Comparison of Different Interconnection Network Designs

In this section we inspect the performance of the L2 cache with varying network size, network topology, bank size, and the position of the core and the memory. We evaluate six designs summarized in Table VII. All configurations have the same cache capacity (16MB) and use efficient Multicast Fast-LRU replacement. Design A, the baseline configuration, uses a 16×16 mesh network to connect 256 64KB banks. Design B uses the same size simplified network by removing most horizontal links and moving the memory controller next to the core shown in Figure 6 (b). A small mesh network (16×4) and large banks (256KB) are incorporated in Design C. Design D still uses a mesh network, but non-uniform size banks are used. One bank set for

a column comprises five banks: two 1-way 64KB banks, one 2-way 128KB bank, one 4-way 256KB bank, and one 8-way 512KB bank in the order of the distance from the core. As a result, Design D maintains the same associativity as Design A. In 16×5 mesh, we set the same 3-cycle link delay in the horizontal direction as for 512KB bank while the link delay in the vertical direction increases as the bank size increases. Design E and Design F use the halo topology with 16 spikes. While Design E has sixteen uniform banks in a spike, Design F has five non-uniform banks in a spike similarly to one column of the Design D mesh network. The memory controller located at the center of the chip increases the wire delay for the off-chip memory. Those increased delays are 16 and 9 cycles in Designs E and F, respectively. We set the size of the spike queue as two entries in the halo topology.

Table VII. Different Interconnection Network Designs

Design	Interconnection Network	Bank Size
A	16×16 mesh	uniform (64KB)
B	16×16 simplified mesh	uniform (64KB)
C	16×4 simplified mesh	uniform (256KB)
D	16×5 simplified mesh	non-uniform
E	spike-16 halo (16 spikes)	uniform (64KB)
F	spike-5 halo (16 spikes)	non-uniform

Figure 11 shows the relative IPC normalized to that of Design A. The simplified mesh network in Design B achieves almost the same performance results as Design A despite the decreased bandwidth due to horizontal link elimination. Even low hit rate benchmarks, *applu* and *lucas*, show the IPC increase by 7% and 10%. The main

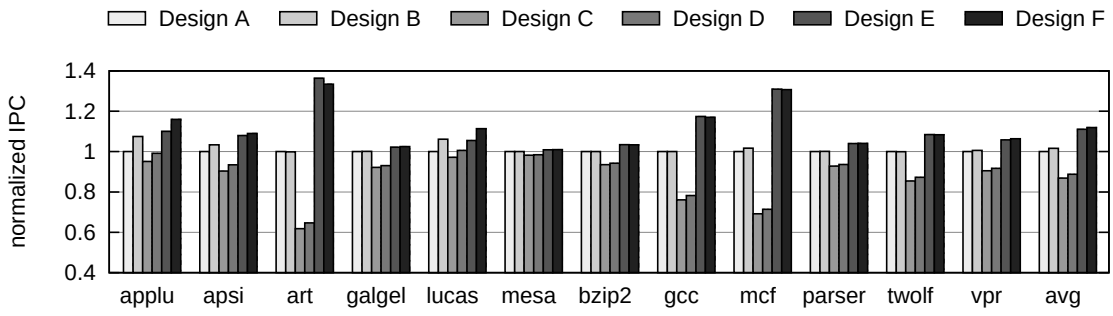


Fig. 11. Performance Comparison in Different Designs

reason of the performance enhancement is the reduced miss latency. Designs C and D show the average performance degradation by 14% and 12% respectively, due to the long wire latency from the large bank size. The halo topology in Designs E and F improve performance by 12% and 13%. Design E has longer miss latency than Design F due to the larger chip size, which increases the wire delay to the memory. High miss rate benchmarks such as *applu*, *apsi*, and *lucas* take better advantage of this. However, *art* having no misses in our simulation shows performance degradation due to increased wire delay for large banks. Design F achieves 1.13 times the IPC increase over Design A. We observe this improvement in both high and low hit rate benchmarks (1.33 times in *art* and 1.19 times in *lucas*). Compared with Multicast Promotion NUCA with a mesh network, Multicast Fast-LRU NUCA with a halo network offers 1.38 times of IPC increase.

We plot average energy consumption per request in Figure 12 and break it down into router, link, and bank in Figure 13. In Design A, router, link, and bank show 12%, 26%, and 62% of the total power consumption, respectively. Design B shows the same power consumption for bank as Design A, but reduces it for link and router by 17% because of the reduced distance between core and memory when handling cache misses.

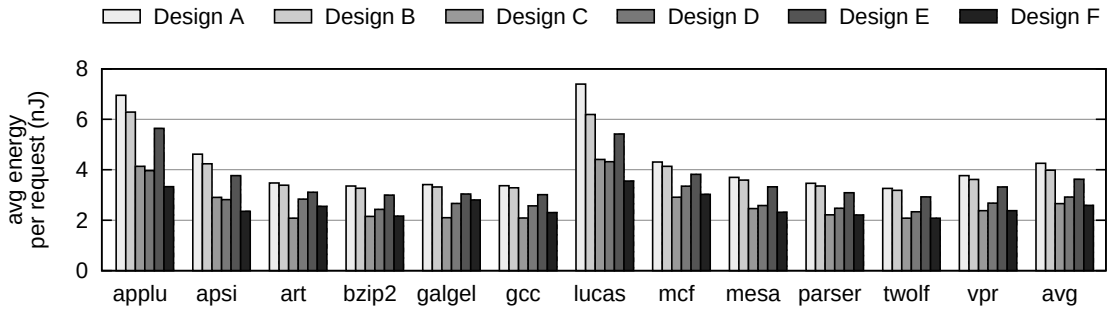


Fig. 12. Energy Comparison in Different Designs

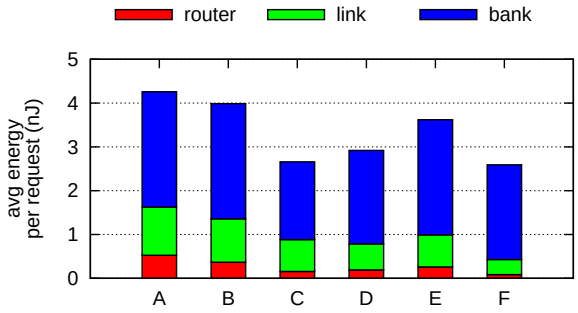


Fig. 13. Energy Breakdown in Different Designs

When we compare Design B and Design C, the large network (16×16 simplified mesh) with small banks (64KB) consumes 1.5 times more power than the small network (16×4 simplified mesh) with large banks (256KB). In fact, large bank incurs an increase for bank access in terms of latency and dissipated power, but reduces the hop count and the energy consumption of routers due to the small network. However, as shown in Figure 13, Design C reduces energy consumption for bank access by 32% and network traversal by 35% over Design B. This result occurs mainly because one 256KB bank shows only 67% power consumption of four 64KB banks based on Cacti. Recent study on bank count showed that 16 or 32 banks achieve the delay-optimal and energy-optimal points in 32MB cache [41].

Design D with non-uniform banks shows comparable power consumption to Design C. The LRU replacement mechanism that offers many hits to banks closer to the core trades off an increase in bank energy for a decrease in router and link energy. Design F provides the smaller hop count than Design E in the halo network. Compared with Design D that has the same bank organization, the halo network in Design F uses the short links for the small banks, while the mesh network in Design D uses the same links where the length is determined by the largest bank. As a result, the halo network takes only 17% of power consumption when accessing cache in Design F.

3. Area Comparison of Different Network Designs

We estimate the required area on the banks, routers, and links of a 16MB L2 cache system. The bank area is extracted from Cacti [53]. Orion [56] is used for the router area, accounting for mainly flit buffers and crossbars. The link area is computed from its width and length. Assuming the wire pitch is $1\mu m$, a bidirectional link that transmits 128-bit flit consists of 256 wires, which has $256\mu m$ width. To estimate the link length expanding one tile, we use the sum of both router and bank areas. We assume that there is no additional area for repeaters and latches in a wire because wires are not routed over banks.

Table VIII. Area Analysis of Network Designs

Design	bank (%)	router (%)	link (%)	L2 cache (mm^2)	chip (mm^2)
A	58.2	8.7	32.5	466.23	552.49
B	74.3	5.4	20.3	365.17	478.59
E	75.3	5.5	19.2	360.44	1447.65
F	92.1	1.9	6.0	294.76	359.03

Table VIII describes the area consumption of each component for four designs discussed in Section 2. The last column is the size of the minimal rectangular chip that includes the L2 cache. Design A (16×16 mesh) uses almost 41% of the cache area for the network. Design B (16×16 simplified mesh) consumes the 49% smaller network area than Design A by removing almost half of the links and incorporating the simple 3-port router that takes up only 45% area for the 5-port router. For halo networks (E, F), we assume that a $4mm \times 4mm$ core is placed in the center of a L2 cache. Design E (a halo network connecting uniform size banks) has almost the same area for the L2 cache as Design B, but its L2 cache uses only about a quarter of the total die. Applying the non-uniform size banks (Design F) not only reduces 17 times the unused chip area over Design E, but also consumes 63% of the L2 area of Design A. The main reason for its compact layout is the small size of the network that requires fewer routers and links. Area estimation for Design F is based on the configuration shown in Figure 14.

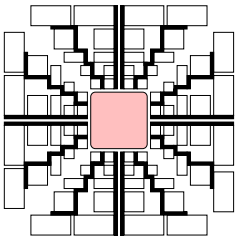


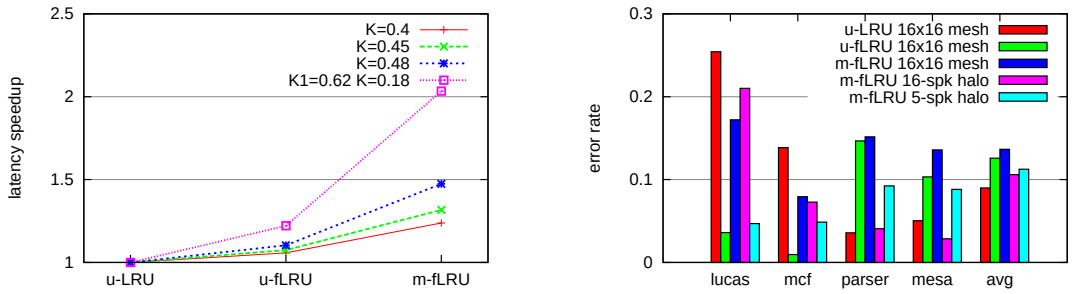
Fig. 14. Spike-5 Halo Network Design for L2 Cache

4. Performance Model Evaluation

To simplify the analysis, we have the following assumptions: The contention-free network makes the network latency as $l_n(d) = d + f - 1$, where f is the number

of flits in one packet for the serial latency in wormhole switching. Data packet and address packet have each 1 and 5 flits. We use the same latency parameters for bank, memory, router, and link as our simulation.

We first apply our analysis to a 16×16 mesh network, assuming that a hit distributions follows the geometric progression with common ratio as K. In other words, a cache hit at y-th way has the K^y probability and an overall miss rate can be computed as $1 - \sum_{y=1}^{16} K^y$. We further assume that cache requests are evenly distributed to each column in the mesh network.



(a) Synthetic Hit Distribution in Mesh (b) Comparison with Experiment Results

Fig. 15. Performance Gain Estimation Using Analytical Model

Figure 15 (a) shows the performance gain of Multicast Fast-LRU on different hit behaviors determined by K. As cache hits concentrate to the closer banks (larger K), Multicast Fast-LRU shows the large performance improvement. For K=0.48, the speedups of Multicast Fast-LRU over Unicast LRU and Unicast Fast-LRU are 1.10 and 1.47 each. Moreover, we highly skew the hit rate (K_1=0.62) only for the first way and use geometric progression (K=0.18) to other ways, similarly to the benchmark behavior as shown in Figure 9. This distribution shows that Multicast Fast-LRU achieves more than 2 times latency improvement over Unicast LRU. It implies that

the interconnect to banks for the first way should be carefully designed, because its latency is critical to the overall performance.

Next, we use each benchmark simulation result for model validation by assigning each bank hit rate in the model. Figure 15 (b) shows the error rate of our analytical model for three replacement policies in the mesh network and Multicast Fast-LRU in two halo networks. Error rate is calculated as $|\frac{Model-Exp}{Model}|$, where $Model$ and Exp are the access latency from our model and simulation. *lucas* and *mcf* have a high miss rate and give a high load to a network, while *parser* and *mesa* show the opposite characteristic. In *lucas* and *mcf*, u-fLRU can reduce large traffic reduction (better accuracy) but increased traffic due to multicast shows low accuracy again. In *parser* and *mesa*, the experiment show a longer miss latency than the analytical model due to the high access burst for the main memory despite their low miss rates. The overall difference between analytical results and experiments is within 14% on average.

The main error source is resource contention from bursty cache traffic such as pipeline stalls in router and memory so that the underlying model mostly underestimates the latency. Introducing a contention model to our analytical model will improve the accuracy. For example, we can use the blocking probability inside the switch of the router and use the queuing model for the average service time of the cache bank or the memory. However, when we consider relatively a high hit rate of the cache and a low injection rate of the network, parameterizing the average behavior have limitations.

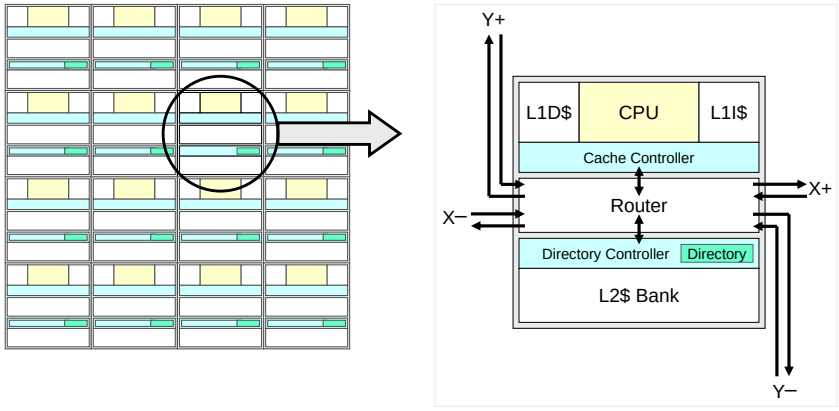
CHAPTER IV

COMMUNICATION CHARACTERIZATION TOWARDS RECONFIGURABLE NOCS

A. Motivation

As Chip Multi-Processors (CMPs) have emerged as a promising way to provide high performance and to increase processor efficiency, we should examine on critical issues regarding these goals. One of the bottlenecks to performance and efficiency in CMPs is communication [57]. To alleviate communication problems, first we need to understand the communication behavior of CMP applications. Motivated by the fact that programs in microprocessor exhibit very different runtime behaviors with a certain pattern [58, 59, 60, 61, 62], we attempt to characterize the communication behavior of CMP applications. This will give better understanding of the interplay between communications and core architectures, and lead us the new design spaces and optimizations of CMPs.

Although CMPs are currently evolving in different directions with various design goals, a scalable CMP design is necessary to accommodate a large number of cores in a die. On-chip interconnection networks [33] are a recent communication architecture paradigm that overcomes the negative effect of technology scaling on global interconnects [1]. Furthermore, the interconnection network coupled with directory-based coherence protocols, can leverage a more scalable design over the shared bus with snoop-based coherence protocols. The tiled CMPs, as shown in Figure 16, can easily support many cores and share the large L2 cache. Each tile has a processor core, private L1 caches and a slice of on-chip L2 cache, and a router. A router is the basic



(a) 16-core Tiled CMP (b) Structure of One Tile

Fig. 16. Tiled CMP with On-Chip Networks

building block of inter-tile communication for a switched network [1].

Most studies on the performance of interconnection networks use synthetic workloads, assuming that the temporal behavior represented as a message inter-arrival time follows an exponential distribution and the spatial behavior is given by a traffic pattern, such as random or permutation traffic. Recognizing the importance of real application behavior assessments in the computer architecture research, there have been previous studies that use the application-driven workloads to evaluate interconnection networks in parallel machines [63, 64, 65, 66]. Though CMP is similar to Symmetric Multiprocessing (SMP) on a chip, on-chip communications have significantly different characteristics, such as short latency, wide channel, and chip budget constraints. However, to the best of our knowledge, there has been no attempt to understand the communication behavior of CMPs in detail.

Recently there has been much interest in investigating the runtime behavior of

[1]It has two local ports to support L1 and L2 caches and multiple inter-tile connection ports In Figure 16 (b), it has 4 inter-tile connection ports for a mesh network.

applications [58, 60, 62]. It has been demonstrated that the time-varying application behavior can be understood through phase analysis. Each phase has portions of execution that exhibit similar behavior regardless of temporal adjacency and represents a distinct behavior that is significantly different from those of other phases. This can be exploited by means of phase-based reconfigurable hardware [59, 61]. However, all of the prior research based on phase analysis was conducted only in a processor architecture domain. We believe that CMPs will take more advantage of reconfiguration for high performance and low power due to their nature of redundancy. Therefore, our research focuses on the development of a phase-oriented communication characterization/prediction framework, in contrast to the traditional analysis techniques or the traffic models to synthesize workload.

In this chapter, parallel applications are used to characterize their communication behavior in a tiled-CMP architecture. The full system simulator is used to collect all the communication activities of each application in the CMP system. On-chip network simulator using these traces is used to analyze performance/power behavior related with the communication behavior for different network designs.

We apply the conventional characterization methods to the communication workload for volume, time, and space aspects. Volume behavior shows that the total volume can be directly estimated from the packet injection rate despite of two different packet sizes. Temporal behavior shows that there is a high variation among injection rates of each source. Spatial behavior shows that there are hot destinations that each source prefers to communicating with, which can generate hot spots in the network. We observe that the temporal/spatial behavior of an application cannot be represented as a single inter-arrival time distribution or a particular traffic pattern. To account for this time-varying spatial-temporal characteristics, a phase-based method groups similar regions of the application's execution into the same phase by clus-

tering techniques. We propose a communication-aware feature vector construction and a period detection method. Using the on-chip network simulator, we show that the changes in the performance and power consumption strongly correlate with the phase changes in the application even with different network designs. We propose two phase prediction mechanisms to control the network proactively rather than reactively. Periodicity in communication behavior motivates the design of String Match predictor that aligns the recent behavior to the past history, while tolerating a variability of periods. To handle a workload with weak or little periodicity, we propose Tournament predictor that combines String Match and Last Value predictors. We evaluate its prediction effectiveness for frequency/voltage scaling to links to reduce power consumption, and distributed injection throttling to improve performance if the bandwidth is insufficient to the communication requirement.

B. Related Work

There is considerable research evaluating the workload of parallel applications in chip-to-chip style shared memory machines and clusters. Most studies used scientific or commercial workloads [66, 67]. Singh et al. argued that communication analysis for parallel applications is incomplete without considering its relationship with local data replication [68]. Based on this insight, Abandah and Davidson divided the application characterizations into configuration independent and configuration dependent parts [65].

Besides memory characterization for application-specific working set and sharing degree, a few studies focused on communication characterization itself. Chodnekar et al. showed that the inter-arrival time cannot always be deduced from a proper distribution function, and there is a spatial distribution unlike the uniform traffic pattern [63]. Heirman et al. observed long bursts in a set of node pairs and showed

the possible performance improvement by adding the extra links [64]. Johnson showed that exploiting the communication locality can give a performance benefit in a non-uniform latency interconnection network based on the performance model [69].

As an interconnection network is becoming a scalable solution for on-chip communication, some CMP design proposals use an on-chip network for a memory hierarchy [27, 70, 28]. Though it is conceptually similar to the traditional off-chip processor-memory network, on-chip communications have several different features: much shorter latency, about a few processor clock cycles per hop, which depends on the physical location in a die; high bandwidth channels due to abundant wiring resources; and chip budget constraints, such as power and area. Recently, Soteriou et al. suggested the on-chip traffic model based on hop count, burstiness, and spatial injection distribution [71]. Varatkar and Marculescu exploited the self-similar property of MPEG-2 video applications in System-on-Chip (SoC) architecture [72]. However, those parameters fail to capture the time-varying and architecture-dependent characteristics of applications.

On the other hand, there have been significant efforts to analyze dynamically changing application behaviors in microprocessor design [58, 59, 60]. Researchers primarily focused on phase analysis studies for the efficient simulation and the dynamic optimization of single-thread applications. The application execution is divided and grouped into multiple similar regions. Basic block vector for control flow [58], data locality [61], and hardware counter for power consumption [62] are used for features in application phase classification. Selection of the proper feature can separate the application behavior into well-defined phases for the problem domain. However, there has been no attempt to analyze communication phases of multi-thread applications using communication-aware features.

C. Characterization Methodology

The goal of this research is characterizing communications from the coherence protocol in tiled CMPs that share the L2 cache. We present two strategies to separate network architecture-independent and network architecture-dependent analyses. The network architecture-independent strategy analyzes the communication behavior that each application produces due to its sharing amount and replication effect from other architectural components such as finite cache and coherence mechanism. The network architecture-dependent strategy investigates the communication cost for each unique behavior across different network platforms. We use two simulators for each strategy. The network architecture-independent behavior is analyzed using traces that the CMP simulator generates. The on-chip network simulator that uses the collected traces is employed to measure the impact of the interconnection network architecture.

The CMP simulator simulates the instruction execution in the core of each tile and the on-chip cache accesses in the memory hierarchy. The simulator consists of Simics [73] that can fully simulate a system, and GEMS [74] that provides a detailed memory system. Simics is configured as UltraSPARCIII+ multiprocessors running Solaris 9. We intentionally increase the network bandwidth modeled in the underlying network of the full system simulator. Point-to-point network is used to relax the constraint given by the network topology and reduce the network simulation time. Table IX shows our configuration. We obtain the basic results from CMP simulation and collect traces for all tile-to-tile communications for further analysis. The trace has the information about the source and destination tiles, the type, the size, and the generation time. Because of the long execution time of an application, we summarize its runtime communication characteristics for each one million-cycle interval.

We build the cycle-accurate interconnection network simulator, and use the traces

Table IX. 16-core and 64-core System Parameters

Configuration	16-core	64-core
Private L1 I & D cache	32KB, 4-way, 2 cycles, LRU	32KB, 4-way, 2 cycles, LRU
Shared L2 cache	16MB, 8-way, 6 cycles, LRU	32MB, 16-way, 10 cycles, LRU
Memory controllers	8	8
Cache block size	64B	
Memory latency	256 cycles	
Network	3 cycles per hop, 16B link width	
Cache coherence	write-invalidation MESI protocol	

generated by the CMP simulator as an input. It models pipelined virtual-channel (VC) routers and links. The router architecture resembles that described in [20]. After a head flit determines the downstream router in the routing stage, the VC allocator assigns one VC in its input channel. The switch allocator arbitrates the use of a switch in a flit level, and then a flit is allowed to traverse the switch. An additional speculative switch allocator permits the VC allocator to operate at the same cycle, if there are available switch ports after normal switch allocation.

We assume that one tile size is $3mm \times 3mm$, and the total chip size is 144 mm^2 for a 16-core model. Link traversal is assumed to take one cycle and buffers are inserted if its latency does not fit one cycle for long-channel topologies. The power consumption of routers and links is modeled from Orion [56] with 1.0V supply voltage and 4GHz clock in 45nm technology. The on-chip network simulator is used to measure the packet latency and the network power consumption.

We examine the communication behavior for five different networks in Section 4. The router in each network has four 4-flit deep VCs. Wormhole-switching and credit-based flow control are used. A brief explanation of each network follows.

- **Mesh:** It is the most widely used in a 2D chip design for its simplicity. The packet is routed first in X dimension and next in Y dimension. The switch in a

router has 6 ports that consist of 4 ports to communicate neighboring routers and 2 local ports for private L1 cache and shared L2 cache.

- **Torus:** It has an additional wrap-around channel in each dimension compared with a mesh network. Folding the torus network provides channels of equal length, which are two times as long as those in a mesh network. We use the deadlock-free dimension-order routing suggested in [75].

- **Hierarchical Mesh (H-Mesh):** It is constructed from a mesh network by adding 2-hop express channels in each dimension so that the router supports a maximum of eight neighbor channels [76]. To reduce the hop count, packets are routed first on express channels and then regular channels in the order of dimension.

- **Doubled Radix-4 Mesh (D-Mesh):** Each router services four tiles and is connected to either of two mesh networks. The generated packet from the source tile travels in one mesh network selected in the round-robin fashion using dimension-order routing.

- **Fat Tree:** Its structure consists of 4-ary tree layers, and two dimensional layout forms butterfly [77]. It uses adaptive routing to the common ancestor (upward) and deterministic routing to the destination (downward). The router has four up-channels and four down-channels.

We study the runtime communication behavior for standard OpenMP benchmarks (SPEComp2001) [78]. We chose medium sized benchmarks, compiled them on a Sun Studio 11 compiler, and executed with reference data sets. Before tracing communication data in each benchmark, we fastforward initial stages of the program by inserting Simics magic breakpoints because most initialization is executed only

Table X. Summary of Executed Benchmarks

	16-core			64-core		
Benchmark	billion instr. executed	L2 misses per instr.	injection load (B/instr)	billion instr. executed	L2 misses per instr.	injection load (B/instr)
ammp	1.630	0.100	11.5	1.986	0.195	22.0
applu	8.335	0.047	6.1	1.755	0.145	18.1
apsi	18.493	0.003	0.5	8.581	0.012	2.0
art	3.412	0.137	14.7	2.420	0.116	13.3
equake	9.271	0.064	7.2	1.169	0.132	14.9
fma3d	9.718	0.046	5.5	1.833	0.154	19.2
gafort	38.855	0.015	1.8	8.390	0.027	4.0
mgrid	10.986	0.050	6.0	1.940	0.137	16.3
swim	8.744	0.080	9.6	2.529	0.105	12.9
wupwise	38.072	0.017	2.0	5.853	0.021	2.7

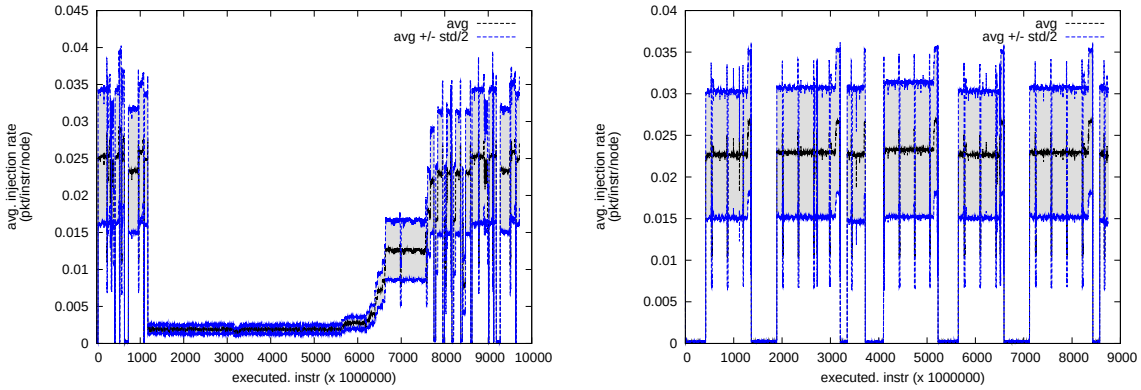
once and is hardly related to its own characteristics. Table X summarizes all the benchmarks used in this work.

D. Characterizing Spatio-Temporal Behavior

In this section, we characterize time-varying behavior of cache traffic in temporal and spatial aspects together. We show the the injection rate variation (temporal property) with different spatial abstractions such as source, destination, and source-destination pair.

Generally, the temporal side of communication is described as the injection rate of the source, which is defined as the average number of injected packets per cycle. The synthetic traffic mostly uses the parameter λ of an exponential distribution to determine the average inter-arrival time.

Additionally, it assumes the same injection rate over all injection processes. To observe the time-varying characteristics of benchmarks, we measure the average and the standard deviation over packet injection rates of each tile for one interval. Fig-



(a) Temporal Change in fma3d (b) Temporal Change in swim

Fig. 17. Spatio-Temporal Traffic Change in 16-Core

ure 17 shows the long-term transition of the average injection rate (solid line) overlapped with the line that is the sum of the average and the standard deviation (dotted line). Although the packet injection rate of an individual tile is not shown, we notice that all tiles have a tendency to change in unison. The difference between the two lines exhibits the difference in each tile's injection rate. Most benchmarks show a significant difference between a tile's injection rate for a part of execution, which is illustrated as a large area between two lines.

Prior work shows that program execution highly depends on the structure of the program. In OpenMP programs that achieve loop-level parallelism, we observe this property as a recurring traffic pattern as shown in Figure 17. We instrumented the program by inserting special instructions at the begining and ending part of the main loop body and sub-procedures. Traffic behavior in each iteration is almost identical when we align the time scale of each iteration at the same point. We believe that this periodic behavior is one of the most useful communication characteristics. If an application is expected to show the past behavior again and the future behavior is speculative, it can be used by any reconfigurable hardware/software mechanism.

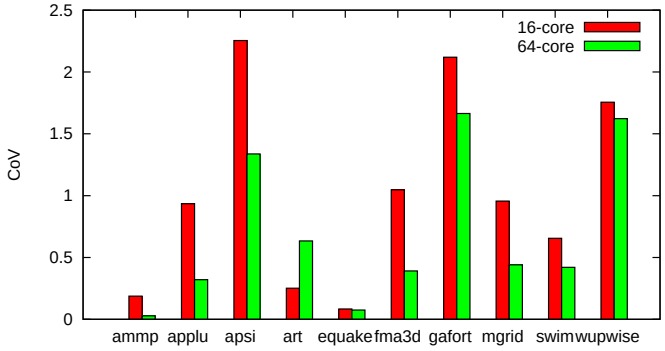
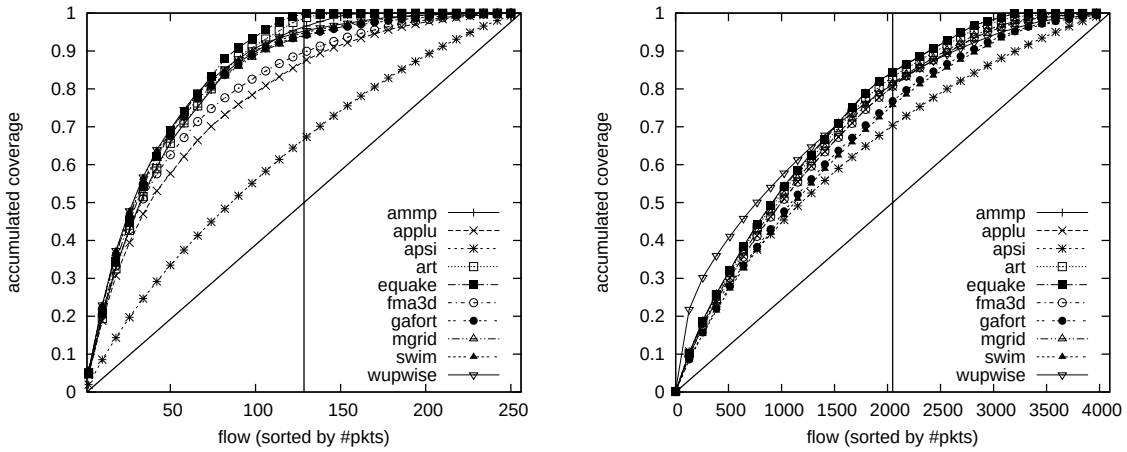


Fig. 18. Temporal Variation Summary

To identify the degree of temporal variability in each benchmark, we take an average injection rate as one sample for each interval. We then calculate the average and the standard deviation of all interval samples. Finally, we use a Coefficient of Variation (CoV) metric, which is defined as the ratio of standard deviation to the average. CoV represents the degree of dispersion in a distribution. Figure 18 shows that *apsi*, *gafort*, and *wupwise* have large temporal variability over samples (large CoV value). We also observe that the relative variablity of one benchmark to other benchmarks is maintained across 16-core and 64-core.

We further analyze spatial distribution in the unit of the flow. We first calculate the number packets for each flow over the total execution and sort them in the ascending order. Figure 19 shows the cumulative contribution on the total traffic according to the number of flows. The half of the total flows contributes 91.6% and 79.9% of the total number of packets in 16-core and 64-core systems, respectively. Even when we consider only top 25% of the total flows, the traffic coverage is 71.1% and 51.3% each. Note that the diagonal lines in Figure 19 are the counterpart of the uniform traffic pattern. These results show that there are some sources and destinations dominating communications.



(a) 16-core (b) 64-core

Fig. 19. Flow Dominancy Analysis

In summary, all the above analysis shows that parallel program execution in cache-coherence systems has very a unique property across different programs. Another important conclusion is that they are very dissimilar from synthesized traffic with a fixed set of simple parameters. Here, we take a systematic approach for classifying intervals into similarly-behaved groups in terms of both time and space.

E. Phase-based Characterization

In the previous section, we found the traditional traffic analysis models cannot capture runtime dynamic behavior. Motivated by our finding, we propose a phase analysis methodology to solve this problem.

First, after constructing the communication-aware feature vector for each interval sample, we use clustering techniques to group similar samples together. We examine the relationship between the classified phases and performance/power behaviors in several network designs.

1. Feature Vector Construction

Because a router in a network has two injection channels from the L1 cache and the L2 cache (directory), the sum of the two injection rates determines how one tile affects the network. To entirely account for both the temporal distribution and the spatial distribution for one interval, we build an N^2-dimensional feature vector for the N-core network. Each element of the feature vector specifies the average injection rate of a *flow* from one source tile to one destination tile.

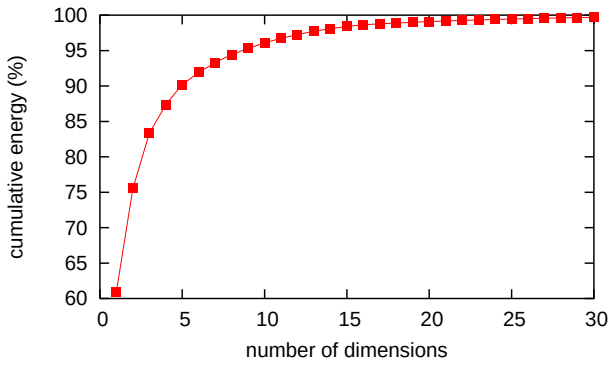


Fig. 20. Dimension Reduction in Flow Feature Vector

However, because the size of the feature vector has a quadratic dependence on the size of the network, finding communication patterns in large networks easily falls into a classical problem of the *curse of dimensionality* in machine learning and requires a long analysis time. We use Principle Component Analysis (PCA), which reduces the original N^2 features into a small number of features without much loss of information [79]. PCA sorts N^2 principle axes (eigenvectors) in the order of decreasing eigenvalues, which represent the distribution of the original data's energy among each of the eigenvectors. Selecting a subset of the eigenvectors projects high dimensional data into low dimensional subspace in a way that is optimal in a sum-squared error

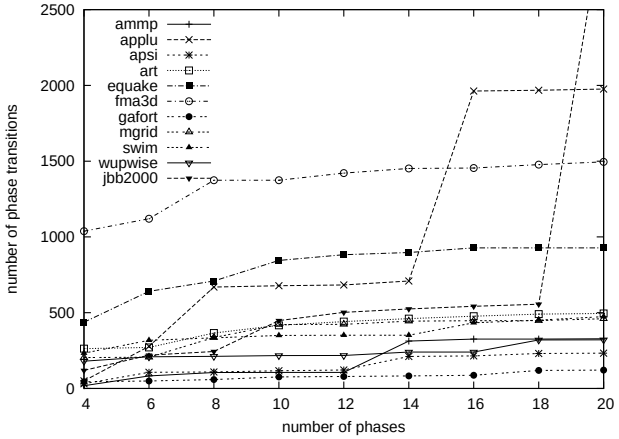
sense. We select the number of reduced dimensions for each benchmark by allowing the cumulative energy in the reduced dimensions to be within 95%. Figure 20 shows the relationship of each flow's injection rate behaviors. It implies that there is a strong dependence among flows. Even in *art* benchmark, which has the highest variability, we find that 25 dimensions are enough to represent its original data.

2. Phase Classification

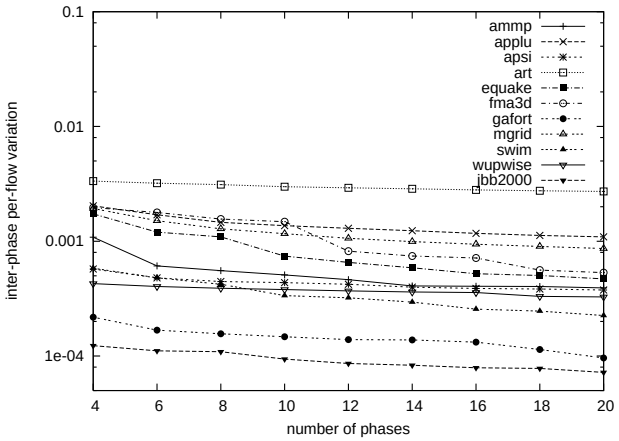
Phase is defined as a set of intervals where each interval's behavior does not change significantly, regardless of temporal adjacency [59]. All the intervals in the same phase need to have similar per-flow packet injection rates. To merge similar intervals in one phase, we use agglomerative clustering, which starts with n singleton clusters and forms a sequence by successively merging the two closest clusters. When we stop merging in the middle of iterations, the resulting clusters are classes (phases) that group similar samples. To minimize variation in the same cluster, Ward's method is applied to compute a distance between clusters as the error of sum-of-squares, meaning it minimizes the within-cluster variance [80]. We use the PCA-reduced feature vector for clustering.

Without prior knowledge of the data samples, it is not easy to determine the number of clusters. Considering that the final clusters are configuration states that the system can support, assigning too many clusters frequently causes a configuration change, and therefore incurs the large reconfiguration overhead. On the other hand, assigning too few clusters causes the loss of information and misconfiguration of the system to an incorrect state. Instead of determining the number of clusters in terms of the quality of resulting clusters [79], we regard it as a system parameter to adapt its structure or a function that maps a phase to a proper configuration.

Figures 21 (a) and (b) show the number of phase transitions and the inter-



(a) Phase Transitions

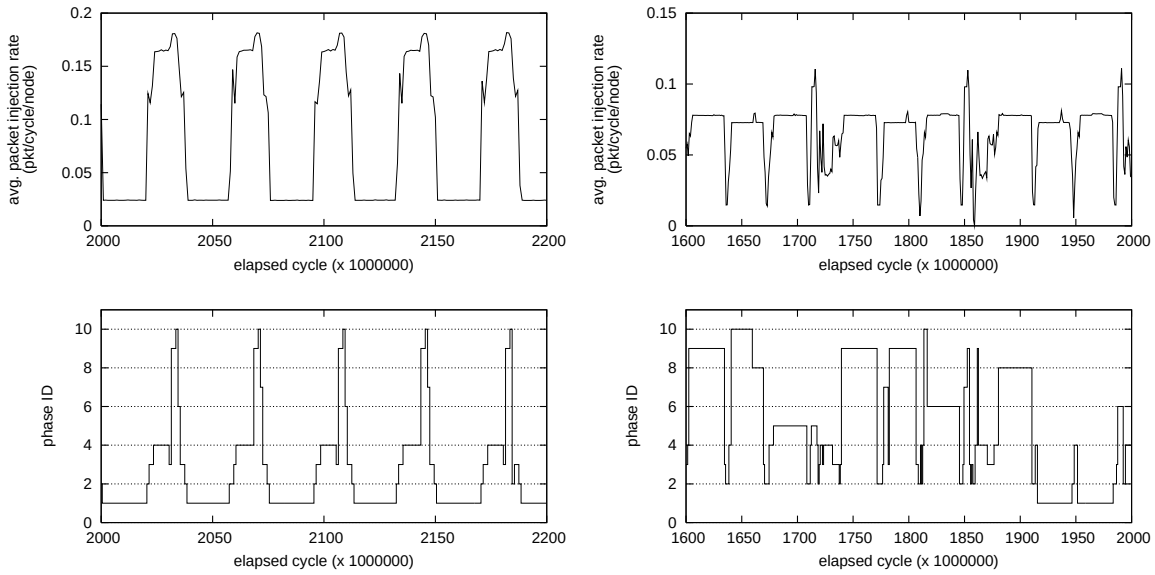


(b) Inter-Phase Variation

Fig. 21. Clustering Result Comparison for Different Number of Phases

phase variation for varying number of phases. To obtain the inter-phase variation, we calculate per-flow standard deviations of each phase, weight them by the portion of execution that the phase covers, and take the average on all the flows. Obviously, the large number of phases produces the larger number of transitions and smaller inter-phase variation than the small number of phases. Compared with the clustering results using the original 256 features, the reduced features only increase 5% and 1% of inter-phase variation for 10-phase and 20-phase on average. We use 10-phase

results for further analysis.



(a) equake (b) mgrid

Fig. 22. Phase Classification Toward Time-Varying Analysis

Figure 22 shows the classified phase change over execution time (top graphs) with the overall network behavior summarized as the average packet injection rate (bottom graphs). In *equake*, even though each peak looks similar, two slightly different consecutive peaks are repeated. In *mgrid*, we observe that the repetition on the average injection rate is not clear unlike *equake* result, because each flow behaves much differently among repeating regions.

3. Phase-Classified Performance and Power Behavior

We investigate how the classified phases are correlated with the architectural metrics. We start with the relationship between performance and phase in the mesh network. Figures 23 and 24 show time-varying average packet latency and power behaviors

coupled with the characterized phases, taken from the middle of *equake* and *swim*. Because each phase represents the part of execution as one distinct spatio-temporal distribution, the change of packet latency and power consumption follows phase transitions.

In *equake* as shown in Figure 23, execution regions that belong to Phases 2, 3, 9, and 10 show much high latency because the large portion of total packets drains into a small set of tiles. For Phase 2 (Figure 23 (c)), tile 3 receives 30% of the total packets and has only two channels to support the tile in the corner of the mesh topology. The asymmetry in the mesh topology and unbalanced traffic pattern cause this congestion in parts of the network, although its average offered load is much smaller than the saturation load in the uniform distribution. Interestingly, Phase 2 has the average offered packet load of 0.13, which is 20% lower than 0.16 in Phase 4 (Figure 23 (d)), where the injection rate of all the flows are almost uniform. This weak correlation between offered load and average latency is observed in the TRIPS on-chip network [81].

In *swim* as shown in Figure 24, the similar load imbalance also appears on the part of execution regions, while it does not have congestion like *equake*. For Phase 1 (Figure 24 (c)), only four tiles send 72% of total packets and receive 49% of total packets on average. For Phase 8 (Figure 24 (d)), the load concentration to tile 0 causes relatively high latency and low power consumption compared with other phases. While Phase 1 has the average offered load 4.33 times over Phase 8, it shows the network power consumption 2.97 times over Phase 8. In other words, Phase 8 has a larger portion of long-distance communications than Phase 1.

In summary, when the workload shows the spatially unbalanced communications, the packet delivery latency and the power consumption are highly sensitive to the spatio-temporal distribution. Furthermore, the severe imbalance incurs congestion

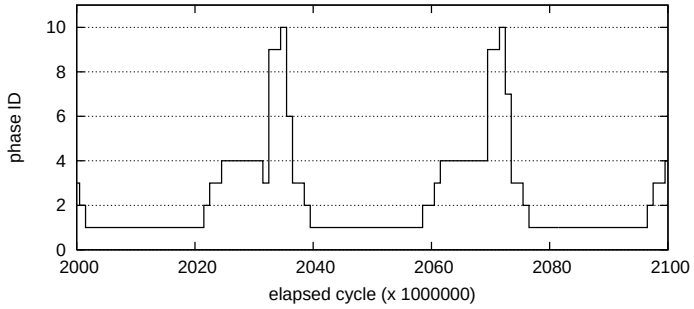
although the offered load is relatively low, unlike the uniform traffic.

4. Network Topology Effects

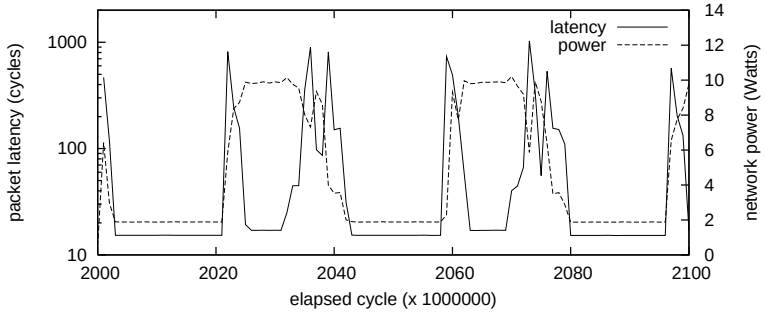
Figure 25 shows the time-varying average packet latency and network power consumption according to phase transitions in each network topology for *swim* and *mgrid*. We see that the latency fluctuation trend in each network is almost identical. As expected, the long-channel networks show the latency improvement by reducing the hop count. For 1724M-1733M cycle and 1772M-1778M cycle regions in *mgrid*, we observe the latency change, while the phase classification result has no phase transitions. Further investigation into those regions shows that burstiness for the short time in each interval is different even though the average burstiness in each interval is similar. It should be emphasized that this is a problem of the particular interval length used here, rather than a problem of phase characterization of communication, and further work can develop the variable length interval dissection schemes based on fine-grain dynamic burstiness analysis.

Figure 26 shows packet latency, network power consumption, energy-delay (ED) product, and average hop count in 200M-cycle simulation results, taken from in the middle of each benchmark execution to cover at least 80% of all the different phases. Packet latency and ED product is normalized to that of the mesh network.

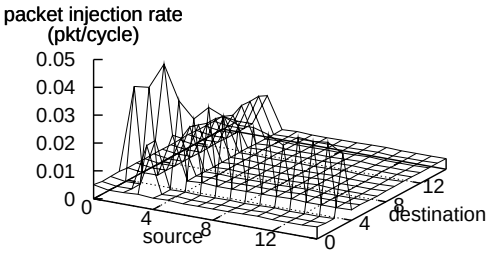
As Figure 26 (a) depicts, all long-channel networks show performance improvement over the mesh network. The fastest D-mesh reduces 54% of the packet delivery latency in the mesh network by reducing 45% of the average hop count in the mesh network. Fat tree is also an effective topology for decreasing the hop count, but does not always outperform a mesh network since its router shows longer contention in the 10-port switch than in the 6-port switch of a mesh network under the high offered load. D-mesh, where a router has also 10-port switch, solves this contention problem



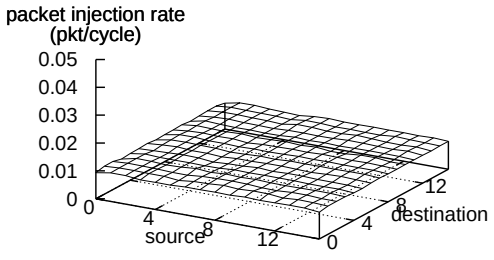
(a) Classified Phases



(b) Performance Behavior

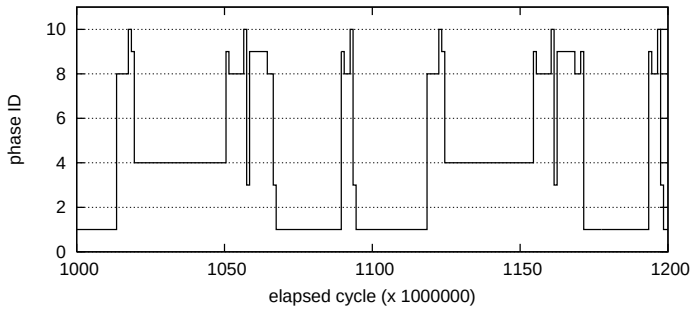


(c) Spatio-temporal Distribution in Phase 2

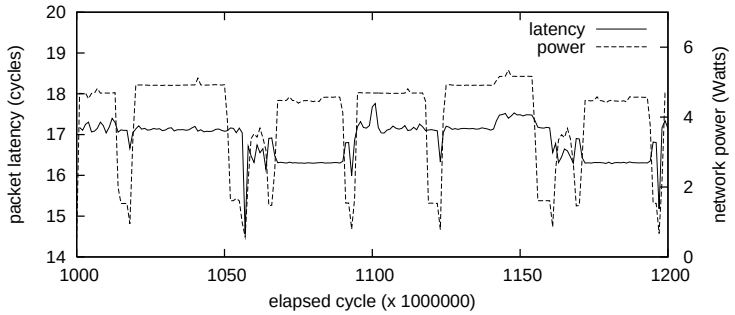


(d) Spatio-temporal Distribution in Phase 4

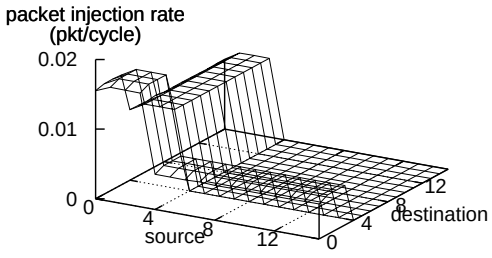
Fig. 23. Relation between Phases and Performance Behaviors in *equake*



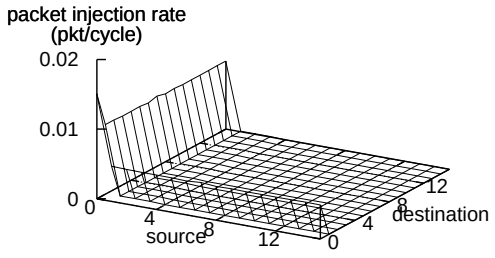
(a) Classified Phases



(b) Performance Behavior

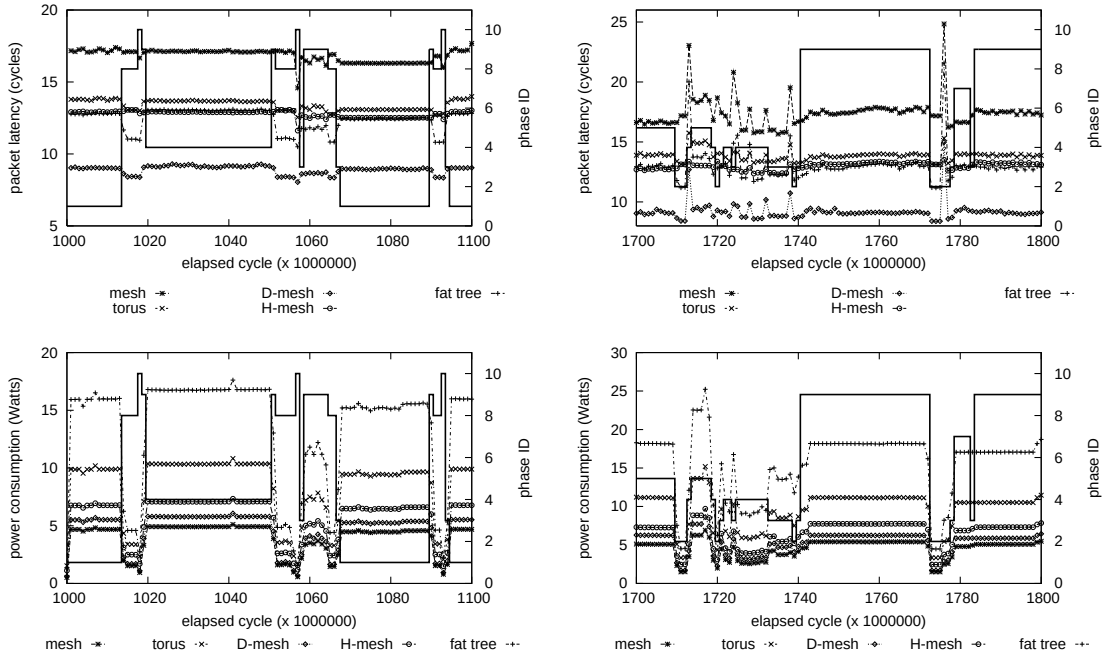


(c) Spatio-temporal Distribution in Phase 1



(d) Spatio-temporal Distribution in Phase 8

Fig. 24. Relation between Phases and Performance Behaviors in *swim*

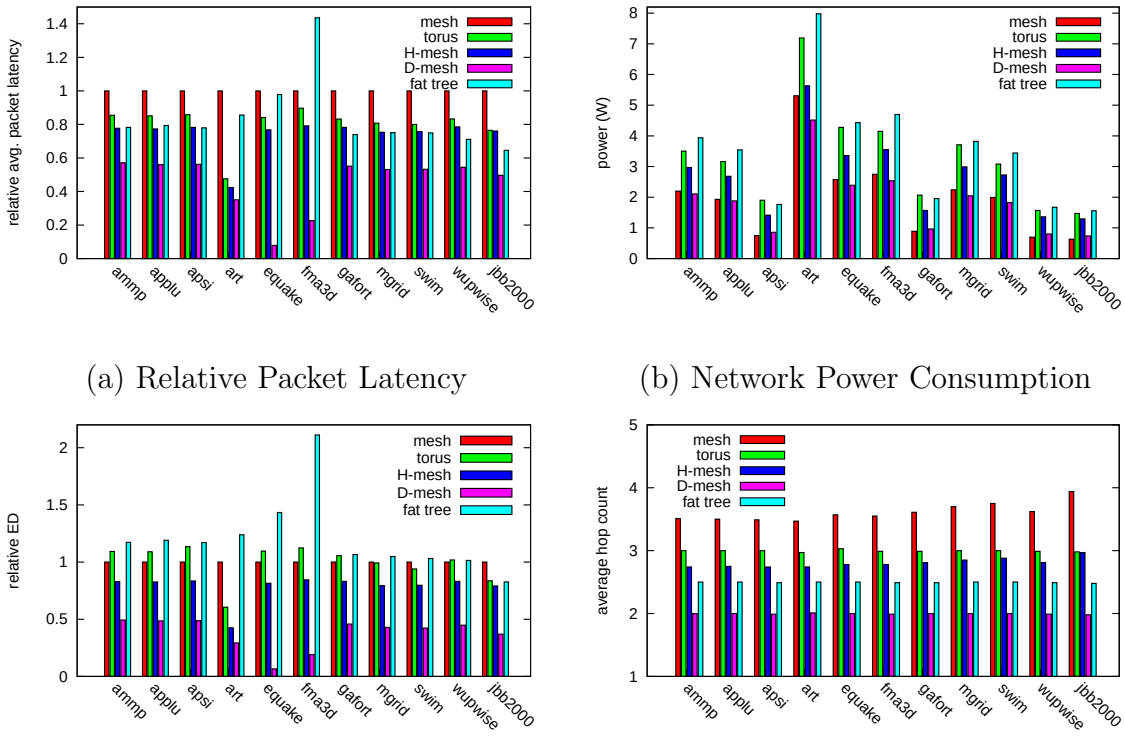


(a) *swim* (b) *mgrid*

Fig. 25. Time-Varying Packet Latency and Power Consumption in Different Networks

of the high-radix switch by distributing the overall load into two mesh networks.

Figure 26 (b) shows that the torus network consumes power 85% higher than the mesh network because its hop count reduction does not trade off the increased channel length to span two tiles. Fat tree network exhibits highest power consumption due to the long channel and high-radix router. However, we should mention that the large-size torus and fat tree networks can achieve better power consumption than mesh networks because the effect of hop count reduction is large. Figure 26 (c) shows that D-mesh is most energy-efficient, which corroborates the benefits of a second network in synthetic workloads [13].



(a) Relative Packet Latency

(b) Network Power Consumption

(c) Relative Energy-Delay Product

(d) Average Hop Count

Fig. 26. Performance Comparison in Different Topologies

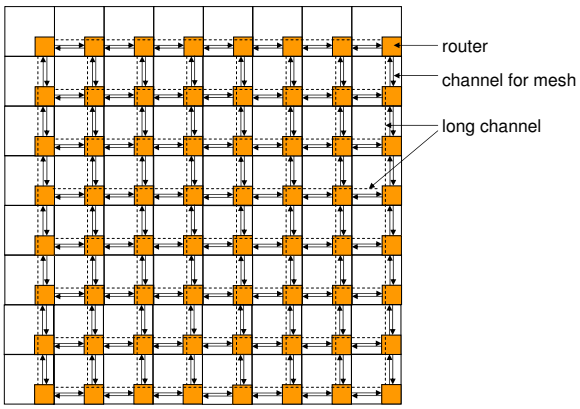
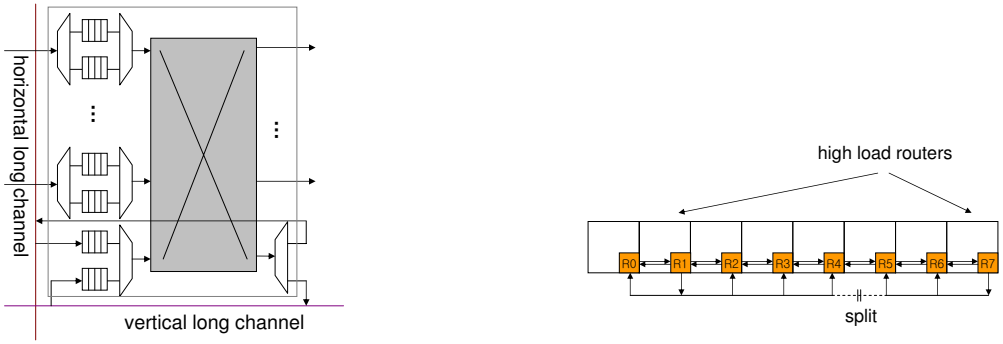


Fig. 27. Mesh Network with Long Channels

F. Applying Characterized Results to Long Channel Configuration

We apply traffic characterization results to long-channel reconfiguration. The physical implementation has two networks: The baseline network has a high-bandwidth scalable mesh topology, and the supplementary network has only long channels span the chip edge. Each long channel can connect all routers aligned in one direction as shown in Figure 27. Long channels are implemented on an ample metal resource, which is relatively cheaper than transistor resource and would increase with increasing metal layers. These long channels are used for some routers that require long-distance communications. Packets that traverse through long channels bypass intermediate routers. Bypassing routers help to enhance the latency by avoiding a router latency as well as the energy consumption.

Figure 28 (a) shows the router architecture supporting our heterogeneous network. The router has 4 ports to west, east, south, and north directions, 1 port to tile, 1 port to long channels. One long vertical and one long horizontal channels share one port of the router using different VCs. We assume that a long channel is directly to connected to the core as well as the router. Because we allow each long channel



(a) Router (b) Split Long Channel

Fig. 28. Hybrid Network Architecture

to have a single source and multiple destinations, it is similar to a multi-drop bus. Unlike an unidirectional channel used for a regular topology, a long channel provides bi-directional packet transfer, hence doubling efficiency in the use of the interconnect. Moreover, arbitration is not needed for resolving conflicts from sending requests. As a result, the transaction time of the long channel is determined only by the interconnect latency, unlike two times of the interconnect latency for request and grant in a bus. If a long channel provides a direct path to the tile, a packet starts to traverse a long channel first or ends transmission at a long channel. This feature saves one router latency for packets that starts or ends transmission at a long channel.

Furthermore, we investigate increasing parallelism in a long channel by splitting a long channel into two short channels as shown in Figure 28 (b). Note that split short channels are still longer in the length than the channels in the baseline mesh network. Like a single long channel, two split channels have each source and a set of destinations. In Figure 28 (b), two packet deliveries (R1→R4, R7→R5) on the same long channel occur simultaneously if split.

Now we discuss a reconfiguration algorithm of long channels. First, we need to

decide if each long channel is split. If split, a split point should be determined. Next, we need to select one router as a source on the long channel or split the channels. This decision process occurs periodically with available characterized results and changes configuration.

When we have a traffic matrix where each element has a packet injection rate between a source and a destination (flow), we can calculate the zero-load latency for a baseline topology. We use a deterministic dimension order routing algorithm that allows a packet to forward first in one dimension and next in the other dimension. The rate of flow can be further divided into each rate for horizontal or vertical direction. We describe two schemes to configure the long channel.

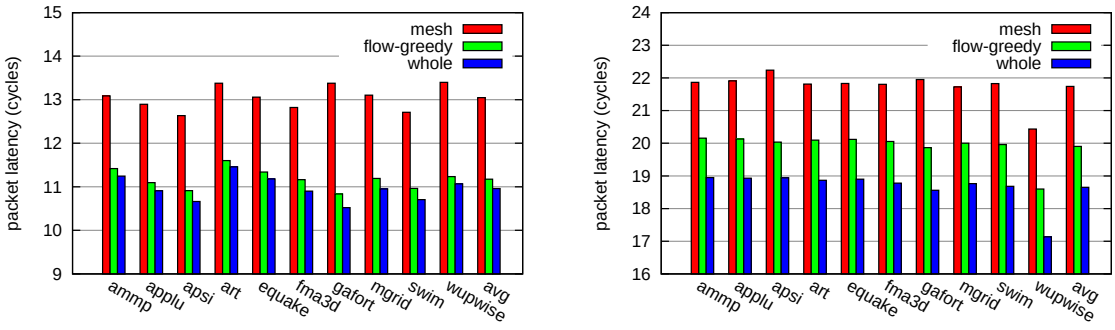
Flow-greedy: We first sort all flows in ascending order of the packet injection rate. We allocate each flow to long channels one by one in the greedy manner. When one flow does not fully utilize one long channel, we split this long channel into two short channels and dedicate one split channel to this flow. Other split channel can be used for another flow. When the flow takes a turn in a mesh topology, we can provide two (horizontal or vertical) long channels to this flow. If some part of channel is already allocated to other flow which has higher injection rate than the current flow, the maximum part of the long channel can be used. This algorithm offers a latency benefit when there are a small number of dominating flows in communication.

Whole: The next algorithm does not have a channel splitting process. Based on the traffic matrix, we can compute the total traversing rate of each router in horizontal or vertical direction. When a flow needs communication in both directions, the rate of this flow is accumulated to the all the routers on the path in either a horizontal or a vertical direction. After we map the injection rate of all flows to the routers, each router has the global information on the all flows that use this router. Finally, we select one router that has the highest rate across routers as the source for the long

channel.

We further exploit latency improvement in different routing algorithms. In a mesh network, XY or YX routing algorithms lead a different long-channel configuration. We calculate the enhanced latency for each routing algorithm and select a routing algorithm for the use of the long channels. For this objective, the baseline mesh network must provide deadlock freedom for XY and YX packet traversal. We divide a set of VCs into XY packets and YX packets. We enforce packet traversal using only one VC set.

When the long channel configuration is changed with a new characterized result, the related routers (sources in the long channel) stop using the long channel and use the baseline network. After no packets on the long channel are confirmed, the long channel configuration is changed and then newly assigned routers for the long channel source start to send packets on the long channel. This localizes synchronization overhead in routers on the same long channel. Moreover, this does not introduce any deadlock because packets always move in XY or YX direction.



(a) 16-core (b) 64-core

Fig. 29. Performance for Mesh with Reconfigured Long Channels

Figure 29 shows latency reduction when long channels are applied to the mesh network. In 16-core, average packet latency is improved by 14% from flow-greedy scheme and by 16% from whole scheme. In 64-core, average packet latency is improved by 8% from flow-greedy scheme and by 14% from whole scheme. Whole scheme achieves better performance improvement because it uses the aggregated rate of flows rather than the rate of one flow.

CHAPTER V

ADAPTIVE DATA COMPRESSION IN NOCS

A. Motivation

The design of a low latency on-chip network is critical to provide the overall high system performance, because the network is tightly integrated with the processors as well as the on-chip memory hierarchy operating with a high frequency clock. To provide low latency, there have been significant efforts on the design of routers [50, 82] and network topologies [14]. However, due to the stringent constraints such as power and area budgets in a chip, simple routers and network topologies are more desirable. In fact, conserving metal resource for link implementation can provide more space for logic such as cores or caches [23]. Therefore, we focus on maximizing bandwidth utilization in the existing network.

Data compression has been adopted in hardware designs to improve performance and save power. Cache compression increases the cache capacity by compressing recurring values and accommodating more blocks in a fixed space [10, 83]. Bus compression also expands the bus width by encoding the wide data as the small size code [84, 85]. Recently data compression is explored in the on-chip network domain for performance and power [12].

In this research, we investigate adaptive data compression for on-chip network performance optimization, and propose a cost-effective implementation. Our design uses a table-based compression approach by dynamically tracking value patterns in traffic. Using a table for compression hardware can process diverse value patterns adaptively rather than taking static patterns [12]. However, the table for compression

requires a huge area to keep data patterns on a flow [1] basis. In other words, the number of tables depends on the network size, since communication cannot be globally managed in a switched network. To address this problem, we present a shared table scheme that can store identical values as a single entry across different flows. In addition, a management protocol for consistency between an encoding table and a decoding table works in a distributed way so that it allows out-of-order delivery in a network.

We demonstrate performance improvement techniques to reduce the negative impact of compression on performance. Streamlined encoding combines encoding and flit injection processes into a pipeline to minimize the long encoding latency. Furthermore, dynamic compression management optimizes our compression scheme by selectively applying compression to congested paths.

B. Related Work

This research is motivated by a large body of prior work in value-centric architectures. Particularly, our work shares some common interests with cache compression and bus compression.

Value locality: Value locality as a small portion of recurred values by load instructions was reported in programs to predict load values [86]. Furthermore it is shown that programs have a set of frequent values across all load and store instructions [9].

Cache compression: Cache compression has been proposed to expand the cache capacity by packing more blocks than given by the space [10, 83]. Alameldeen, et al. used the frequent pattern compression (FPC) scheme to store a variable number

[1] A flow represents a pair of source and destination. Therefore, an n-node network has n^2 flows.

of blocks in the data array of the L2 cache [10]. Due to the increased hit latency for decompression, they developed an adaptive scheme to determine if a block is stored in a compressed form. However, apart from compression hardware cost, cache compression requires significant modification to existing cache designs for the flexible associativity management.

Bus compression: Bus compression can increase the bandwidth of a narrow bus for wide data. Bus-Expander stores the repeated high order bits of data into a table [84]. For one data transfer, the index into the table is sent along with the lower bits of data. All the tables on the bus maintain the same content by snooping. However,a snooping mechanism is not suitable for switched networks. Also a replacement in a table causes another replacements in all the tables, though a newly placed data is directly relevant to only two tables in a sender and a receiver. This global replacement can evict a productive index for compression and result in a low compression rate. Power Protocol takes a similar scheme for bus energy reduction [85].

Bus encoding techniques have been proposed to reduce the energy consumed in high-capacitance buses [87, 88, 89]. By detecting bit transition patterns on a bus, encoding hardware converts data into a low-transition form stored in a table. Introducing a special code for encoding further reduces energy consumption of the bus. An extra-bit line is needed to indicate whether the data is encoded or not. Bus-invert method transmits either original or inverted data depending on which would result in a small number of bit transitions [90]. Transition pattern method encodes data into pre-built code that accounts for both inter-wire and intra-wire transitions [87]. It requires an encoder at the sender side and a decoder at the receiver side to synchronize consistency. A Content-Addressable Memory (CAM)-based value table is usually used to store repeated data and convert them into the energy-efficient encoded indices [85, 91, 89].

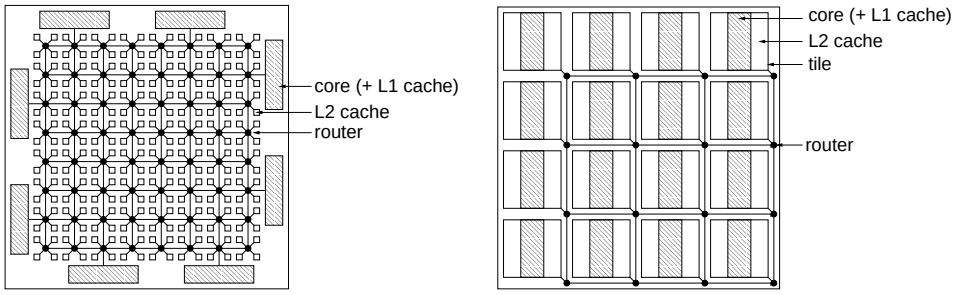
Most of these schemes assume bus-style interconnects, where data for compression is perfectly synchronized across all the nodes. A switched network, where each node needs to communicate with multiple nodes asynchronously, makes this problem challenging. Simply duplicating tables on a per-flow basis is not scalable towards a large scale network with many cores. Moreover, compression can increase the communication latency because a compression process is performed before a communication process. Thus we need to develop a compression solution to minimize the negative impact on performance.

C. Data Compression in On-Chip Networks

In this section, we briefly present the on-chip network architecture and discuss benefits from data compression. Next, we propose a table-based data compression scheme to reduce the packet payload size.

1. On-Chip Network Architecture

Each processing element (PE) such as a core, a cache, and a special processing engine is interconnected through a network. A switched network consists of routers and links whose connection determines a network topology with a proper routing algorithm.



(a) 8-core SNUCA-CMP (b) 16-core TILED-CMP

Fig. 30. On-Chip Networks in CMPs

Figure 30 shows each network layout for 8-core and 16-core CMP systems. The first design is SNUCA-CMP integrating many fast cache banks to reduce a long access time of the monolithic L2 cache [27]. In an 8×8 mesh network, each router connects four cache banks and bridges to four neighboring routers [2].

The second design is TILED-CMP connecting homogeneous tiles to aim for the many-core paradigm. Each tile has a core, private L1 caches, a part of a shared L2 cache, and a router. An N-core CMP has an N-tile network. Each router has two local ports for L1/L2 caches in its own tile and four ports to neighbor tiles for a mesh network.

In both designs, most communication is cache requests/responses and coherence operations for shared memory systems. Traffic has a bimodal-length distribution, depending on whether communication data includes a cache block or not. In other words, a packet payload has one of the followings: address only (*address packet*), or both address and cache block (*data packet*).

Router: The router uses wormhole switching for small buffer cost, virtual channels (VCs) for low Head-Of-Line (HOL) blocking, and credit-based flow control. The pipeline stages of a conventional router consist of route computation (RC), VC allocation (VA), switch allocation (SA), and switch traversal (ST) [16]. First, the RC stage directs a packet to a proper output port of the router by looking up a destination address. Next, the VA stage allocates one available VC of the downstream router determined by RC. The SA stage arbitrates input and output ports of the crossbar, and then successfully granted flits traverse the crossbar (ST). In this 4-stage pipeline, the RC and VA stages are required only for head flits.

[2]Most routers have eight ports. Additionally, twelve routers have nine ports to connect a core (eight routers at periphery) or a memory controller (four routers in the center).

In our study, we use a 2-stage pipeline, which adopts lookahead routing and speculative switch allocation [20]. Lookahead routing removes the RC stage from the pipeline by making a routing decision one hop ahead of the current router. Speculative switch allocation enables the VA stage to be performed with the SA stage simultaneously. A separate switch allocator finds available input and output ports of the crossbar after the normal switch allocator reserves them.

Network Interface: A network interface (NI) allows a PE to communicate over a network. An NI is responsible for packetization/de-packetization of data and flit fragmentation/assembly for flow control as well as high-level functions such as end-to-end congestion and transmission error control.

Link: Links for connecting routers are implemented as parallel global wires on metal resources. Setting a link width equal to the address packet size may increase link utilization and allow more metal resources for power and ground interconnects. Buffered wires are used to fit a link delay within a single cycle [1].

2. Compression Support

In a switched network, communication data is transmitted as a packet. At a sender NI, a packet payload is split into multiple flits for flow control, and then enters into a network serially. After traversing the network, all flits belonging to the same packet are concatenated and restored to the original packet.

If a specific value appears repeatedly in communication, it can be transmitted as an encoded index, while any non-recurring value is transmitted in the original form. This is done by accessing a value encoding table that stores recurring values in the sender NI. When a packet with an encoded index arrives, it is restored to the original value by accessing a value decoding table in the receiver NI. Because the index size is much smaller than the value size, encoding can compress the packet.

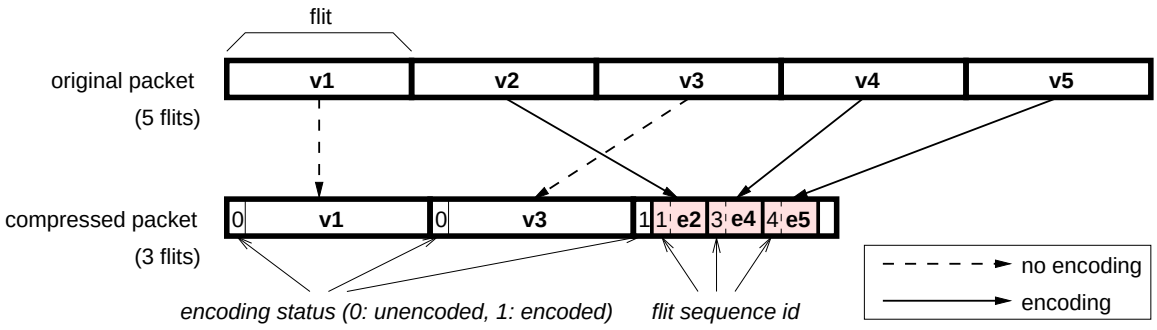


Fig. 31. Packet Compression Example

Figure 31 shows an example of how to encode a packet payload data as flits. We assume that the value size for a single encoding operation is the same as the flit size. In a compressed packet, the encoded indices (`e2`, `e4`, `e5`) follow the original data (`v1`, `v3`). This structure enables multiple flits to be successfully packed into a flit. Although it changes the data order in the original packet, it simplifies encoded index alignment with unencoded values and flit assembly at the receiver side. Because packet data can be partially compressed, reconstruction of the original packet requires two additional data: One bit indicating an encoding status for each flit and a flit sequence identifier for encoded flits to arrange all the flits in the order of the original packet data. Additionally, we do not consider a specific energy-aware coding when building an index [87, 88, 89], because sharing links for different flows makes it hard to predict wire switching activities.

We show how packet compression changes the packet delivery latency and power consumption. The contention-free packet delivery latency (T_0) consists of router delay (T_r), wire delay (T_w), and serialization latency (T_s). The hop count (H) determines the total router delay as (HT_r) and the total length of wire that affects wire delay (T_w). Serialization latency (T_s) is determined as L/b, where the packet length is L

and channel bandwidth is b.

$$T_0 = HT_r + T_w + T_s(= L/b) \tag{5.1}$$

As the network load increases, more conflicts on ports and channels contribute to longer intra-router delay or wire delay so contention delay (T_c) is appended to T_0.

Compressing a long packet (L) into a short packet ($L' \leq L$) reduces serialization latency at the cost of encoding and decoding latencies (T_e and T_d).

$$T_0' = HT_r + T_w + T_s'(= L'/b) + T_e + T_d \tag{5.2}$$

Compression may increase the normal contention-free latency. However, in wormhole switching, the reduced packet size can achieve better resource utilization. Because the average load for each router is reduced, this leads to less contention for shared resources.

Energy consumption E_p of a packet is given by

$$E_p = L/b(DE_{link} + HE_{router}), \tag{5.3}$$

where D is the Manhattan distance, H is the hop count, E_{link} is the unit length link energy consumption, and E_{router} is the router energy consumption. By reducing L to L', compression reduces the number of flits in a packet from L/b to L'/b. Hence the router and link energy for a packet can be reduced at the cost of encoder and decoder energy (E_{enc} and E_{dec}). It can be derived as:

$$E_p' = L'/b(DE_{link} + HE_{router}) + E_{enc} + E_{dec}. \tag{5.4}$$

Longer-distance communication packets (larger D and H) can save more energy, because the required energy for routers and links becomes much larger than the energy for compression. This additional energy for compression mainly depends on

the size of value tables. Next, we explain table organizations to store recurring values.

3. Table Organization

In an n-PE network, each PE needs n encoding tables to convert a value into an index and n decoding tables to recover a value from a received index. We call this organization *private table* scheme, because it maintains a separate table for each flow. The encoding table that has value-index entries is constructed using a CAM-tag cache, where a value is stored in a tag array for matching while an associated encoded index is stored in a data array. Those indices can be pre-built or read-only because they do not need to be altered at runtime. In the decoding table that has index-value entries, the received index is decoded to select the associated value. Since the decoding table is simply organized as a direct-mapped cache, the received index can uniquely identify one value. A PE address is used to activate a proper table.

One encoding table and its corresponding decoding table need to be consistent to precisely recover a value from an encoded index. Both tables have the same number of entries and employ the same replacement policy. If a packet data causes a replacement in the encoding table, it must also replace the same value in the decoding table upon arrival. Furthermore, the network must provide in-order packet delivery to make replacement actions for both tables in the same order. To guarantee in-order delivery, a network needs a large reorder buffer at receivers, which requires additional area cost, or it should restrict dynamic management such as adaptive routing.

The private table scheme relies on the decoding ability from per-flow value management. This does not provide a scalable solution as the network size increases. A substantial chip area must be dedicated for implementing private tables. Moreover, it is possible that an identical value is duplicated across different tables, because each table is exclusively used for a single flow. Therefore, despite the large table capacity,

the private table scheme cannot manage many distinct values effectively.

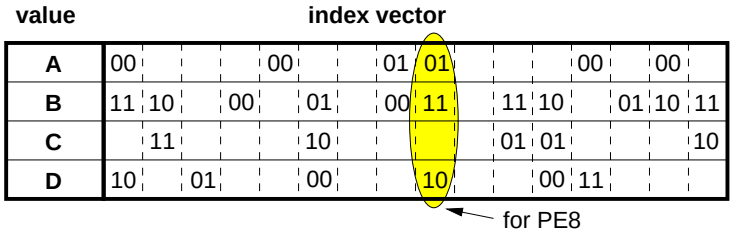
D. Optimizing Compression

In this section, we present table organization and its management to overcome a huge cost of the private table scheme. We propose two performance improvement techniques; overlapping encoding with flit injection and dynamically controlling compression for workload.

1. Shared Table

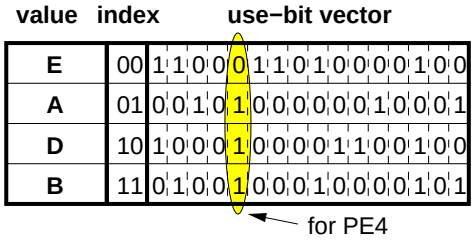
Table Structure: Each PE has one encoding table and one decoding table by merging the same values across different flows. We call it *shared table* organization. Value analysis in two CMP architectures reveals that one sender transmits the same value to a large portion of receivers and vice versa (See the detailed results in Section 1). Therefore, having a network-wide single entry for each value in tables can dramatically reduce the table size. Unlike the private table scheme, a receiver finds value patterns used for encoding. When a receiver places a new value in the decoding table, it notifies the corresponding sender of the new value and the associated index. After a sender receives the index for the new value, it can begin to compress that value.

In the encoding table, a value is associated with multiple indices constructed as a vector. The position of the index vector indicates one PE as the receiver. Each element has an index value that will select one entry in the corresponding decoding table. In the decoding table, one entry has three fields: a value, an index, and a use-bit vector. Each bit in the use-bit vector tells if the corresponding sender transmits the associated value as index. Figure 32 shows the structure of each table for a 16-PE network, where the encoding table is for PE4 and the decoding table is for PE8. The encoding table shows that **A**, the value of the first entry, is used by six receiver PEs



Each element shows a binary index for value at decoder.

(a) Encoding Table in PE4



Each bit indicates a value status of encoder.

(b) Decoding Table in PE8

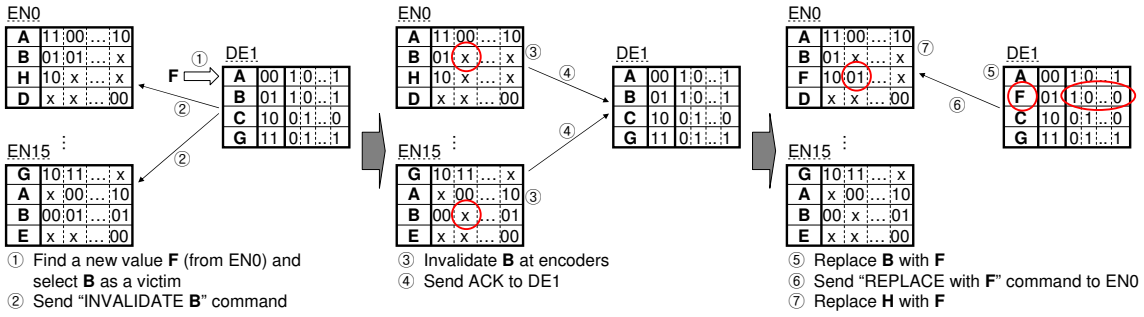
Fig. 32. Shared Table Structure

(0, 4, 7, 8, 12, 14). Likewise, the decoding table shows that **A**, the value of the second entry, is used by four sender PEs (2, 4, 11, 15). The encoding table in PE4 indicates that three values (**A**, **B**, and **D**) can be encoded when transmitted to PE8.

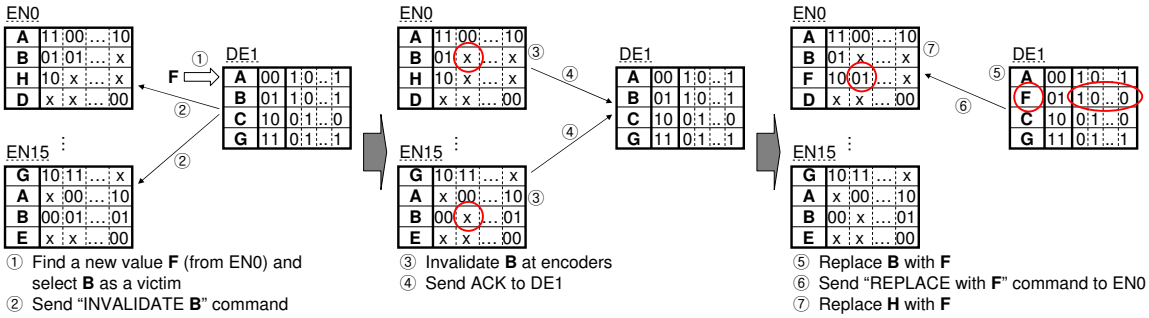
Table Consistency: Value-index association between a sender and a receiver must be consistent for correct compression/decompression. In other words, a sender must not transmit an index from which a receiver cannot restore the correct value in its decoding table. Specifically, an index associated with the value in the decoding table can be used in multiple encoding tables. Changing a value associated with an index in the decoding table requires some actions in tables that use the index. Thus, a consistency mechanism is required between encoding tables and decoding tables.

For this purpose, we propose a simple management protocol for the shared table scheme. Note that a receiver tracks new values. As a result, inserting a new value

into the decoding table starts at a receiver. When a specific value appears repeatedly, the receiver does one of the following two operations – If a new value is not found in the decoding table, the receiver *replaces* an existing value with the new value. If a value is found but the use-bit for the sender is not set, the receiver *updates* the corresponding use-bit of the decoding table. After either replacement or update, the receiver notifies the corresponding sender of the associated index for the new value. Finally, the sender inserts the index and the new value in the encoding table.



(a) Replacement



(b) Update

Fig. 33. Shared Table Management

Figure 33 (a) illustrates a replacement example with two encoding tables (EN0 and EN15 for PE0 and PE15) and one decoding table (DE1 for PE1) in a 16-PE

network. DE1 has two values (**A** and **B**) for EN0 and three values (**A**, **B**, and **G**) for EN15. When a new value **F** comes to DE1 (①), the decoding table needs a replacement for **F** and decides to evict **B**. Then, it requests all the related encoding tables (②) for invalidation of **B** (③) and waits for invalidation acknowledgment from the encoding tables (④). DE1 replaces an old value **B** with a new value **F** (⑤) and then sends replacement to related encoding tables (⑥ and ⑦).

Figure 33 (b) illustrates an update example. A sender (EN0) transmits a new value **G**, which is in the decoding table but the use-bit for EN0 is not set. DE1 sets the corresponding bit of the use-bit vector (②) and sends UPDATE command for **G** to EN0 (③). Finally, EN0 has **G** (④).

This management protocol makes sure that the encoding table encodes only values that the decoding table has. Note that an encoding table update action for a sender is initiated by a receiver. The decoding table can have more entries than the encoding table to accommodate many distinct values from different senders. Furthermore, it does not need an in-order packet delivery mechanism.

Increasing Compression Effectiveness: Because a single decoding table handles value locality from multiple flows, the shared table may experience many replacement operations due to the increased number of non-recurring values, causing a low compression rate. One replacement operation in a decoding table requires at least two packet transmissions to be consistent with an encoding table, increasing control traffic.

To mitigate this problem, we employ another table, value locality buffer (VLB), that filters undesirable replacement for the decoding table. VLB has a simple associative structure where each entry has a value and a counter for frequency. When a value arrives at a receiver, it is stored in VLB. VLB follows the least frequently used replacement. Whenever a value hit occurs in VLB, the counter of the entry

Table XI. Value Table Area Analysis

	Private Table		Shared Table	
Encoder	value:	$v \cdot e \cdot n$ (CAM)	value:	$v \cdot e$ (CAM)
	index:	$\log e \cdot e \cdot n$ (RAM)	index vector:	$\log d \cdot e \cdot n$ (RAM)
Decoder	index:	0	index:	$\log d \cdot d$ (CAM)
	value:	$v \cdot d \cdot n$ (RAM)	value:	$v \cdot d$ (CAM)
			use-bit vector:	$d \cdot n$ (RAM)
			VLB:	$v \cdot d$ (CAM)

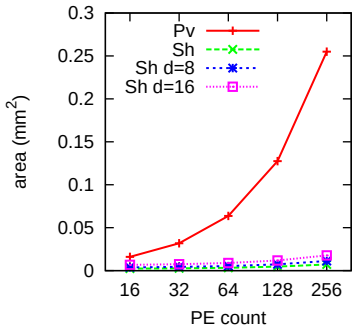


Fig. 34. Area Comparison for Private and Shared Tables

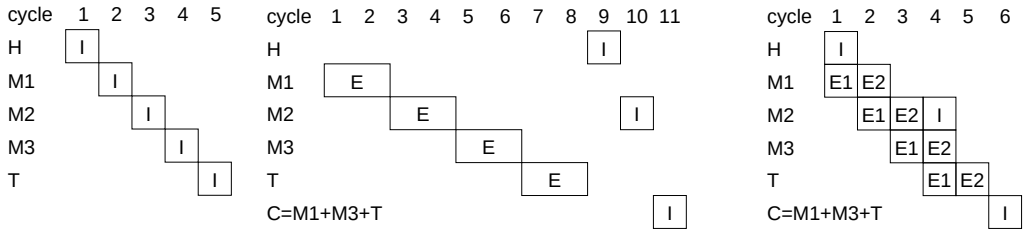
is increased by one. When a counter is saturated, the associated value results in replacement for a decoding table. In other words, VLB is used for confirming the temporal locality of new values.

Table Area: The associative search part of a table needs CAM implementation and other parts are constructed as RAM. To compare the private and shared table schemes, we estimate each area as the sum of CAM and RAM cells. Table XI shows the required number of RAM and CAM cells for each scheme, where n is the PE count, v is the value size in bits, and e/d is the number of entries in encoding/decoding

table. We do not account for counters and valid bits. We scale the cell area for 45nm from [92]. It gives us 0.638 μm^2 for RAM (6 transistors) and 1.277 μm^2 for CAM (9 transistors). Figure 34 shows huge area overhead of Private table (Pv) for 8-entry 8B value as the number of PEs increases. Though the decoding table has more entries than the encoding table such as 8 (Sh d=16) and 16 (Sh d=32), the increased area for shared table is still scalable.

2. Streamlined Encoding

To hide the long encoding latency, we propose streamlined encoding. In general, packet injection into a network begins after all flits of the packet participate in the encoding process. In this situation, the long access time of an encoding table and multiple table accesses increase the overall network latency as depicted in Eq. 2. Furthermore, we exploit the fact that multi-cycle encoding for the table access can be divided into multiple stages in a pipeline using a cache pipelining technique [93]. Integrating flit injection and encoding processes into a pipeline can make each process work concurrently.



(a) No Encoding - 5 cycles

(b) Encoding - 11 cycles

(c) Streamlined Encoding - 6 cycles

Fig. 35. Streamlined Encoding Example

Figure 35 shows the latency benefit of streamlined encoding for a 5-flit packet,

which requires 5 cycles for flit injection (Figure 35 (a)). Figure 35 (b) shows that 8 cycles are required to encode for four flits (M1, M2, M3, T) when the encoding table access needs 2 cycles [3] . Then, injection takes 3 cycles for two uncompressed flits (H, M2) and one compressed flit(C). Figure 35 (c) shows that streamlined encoding increases the latency by only 1 cycle using a three-stage pipeline, which has two stages for encoding and one stage for injection.

Early injection of head flits can reduce cycle stalls in a router pipeline by decreasing resource conflicts required for following flits. If a head flit reserves a necessary VC early, following flits can avoid stalls for VC allocation. However, it may cause low buffer utilization due to increased VC reservation time.

3. Dynamic Compression Management

Despite streamlined encoding, the decoding latency for compressed packets is still required and increases the packet delivery latency. In a lightly loaded network, compression is not favored because packets are delivered almost free of contention. Moreover, if the packet data is found to be incompressible, encoding and decoding operations just increase the network latency.

To further optimize performance, we propose a dynamic management of data compression to workload. It identifies routers that experience high congestion, and applies data compression to packets that go through congested paths. Because compression begins at a sender side, compressing all the packets going through congested paths can help to alleviate congestion. Congestion is detected by measuring a specific delay component from a network architecture.

Congestion detection at senders: A packet generated by a PE is stored in

[3] In cache traffic, a head flit for a packet is constructed with only the memory address, so that a head flit does not need compression.

a buffer of the attached NI. Each packet waits until it is fragmentefragmented into flits and injected to a router. When one PE instantly generates many packets, it causes to oversubscribe the injection port bandwidth of the router. In this case, a packet stays in the NI buffer for a long time. This *queuing delay* is used as one unit of congestion abstraction. By estimating the queuing delay, compression is applied to packets if the queuing delay is beyond a threshold. In our experiment, zero is used for the threshold value. When considering shallow buffers in a wormhole router and the use of back pressure, congestion is instantly propagated to the network if not controlled immediately. Additionally, we use a buffer status of both NI and router for a compression decision. If the NI buffer has at least one packet or the router buffer is full, it implies that there is a non-zero queuing delay for the next or current packet. If none of three conditions is met, the packet data is not compressed.

Congestion detection at receivers: Congestion also arises in the middle of delivery paths, because routers and links are shared. When a flit or a packet fails to reserve a necessary resource, it is stored in the flit buffer, hence, increasing the network delay. This type of congestion appears as *contention delay*, which is an extra delay component added to the zero-load delay (Eq 1.). Contention delay is computed by subtracting zero-load delay from measured network delay, when a sender attaches a network injection timestamp to a packet. A receiver makes a decision for compression of a sender by comparing a measured contention delay against a given threshold. Additionally, a receiver keeps a compression status of each sender and sends a control packet only if the status is changed. The threshold can be preset at the design time or be adjusted for a specific application.

Estimating delay: The last k packets are used to estimate the queuing delay and contention delay. A small k can detect highly bursty injection for a short period of time and react congestion immediately. Meanwhile, a large k can smooth out

workload behavior but does not increase control traffic much.

E. Methodology

Our evaluation methodology consists of two parts. First, we used Simics [73] full-system simulator configured for UltraSPARCIII+ multiprocessors running Solaris 9 and GEMS [74] that models directory-based cache coherence protocols to obtain real workload traces. Second, we evaluated the performance and estimated the dynamic power consumption of varying compression schemes using an interconnection network simulator that models routers, links, and NIs in detail.

Table XII shows main parameters of 8-core SNUCA-CMP and 16-core TILED-CMP designs for 45nm technology as shown in Figure 30. We chose the 4GHz frequency, which respects power limitation in future CMPs guided by [94]. All the cache related delay and area parameters are determined by Cacti [95]. Both designs accommodate 16MB networked L2 cache and superscalar cores configured similarly with [27]. Assuming that each chip area is $400mm^2$, the cache network in SNUCA-CMP has $2mm$ hop-to-hop links to connect $1mm^2$ banks and then the partitioned L2 cache lies on $256mm^2$. Tiles in a 4×4 mesh network of TILED-CMP are connected with $5mm$ long links.

The network simulator that takes care of generated packets from the CMP simulator, models the detailed timing and power behaviors of the routers, links, encoders and decoders. The router in each CMP design is configured to fit its pipeline delay [4] for one cycle of 4GHz clock using the logical effort model [20]. Each router has 4-flit buffers for each virtual channel and one flit contains 8B data. We estimate router

[4]VC allocator (R→p) has the longest latency among others and determines the delay of a router pipeline in both designs. Therefore, the number of VCs is selected for one clock cycle time.

Table XII. CMP System Parameters for Data Compression

CMP design	SNUCA-CMP	TILED-CMP
clock frequency	4 GHz	4 GHz
core count	8	16
L1 I & D cache	2-way, 32 KB, 2 cycles	2-way, 32 KB, 2 cycles
L2 cache	16-way, 256×64 KB, 3 cycles (per bank)	16-way, 16×1 MB 10 cycles (per bank)
L1/L2 cache block	64B	64B
memory	260 cycles, 4 GB DRAM	260 cycles, 8 GB DRAM
coherence protocol	MOSI	MSI
network topology	8×8 mesh	4×4 mesh

power consumption from Orion [56]. High connectivity networks need high radix routers that require a longer pipeline delay and consume more power than low radix routers. Thus higher radix routers in SNUCA-CMP require more power consumption than those in TILED-CMP. Table XIII (a) shows the router pipeline delay and the energy consumption of each router component. The second column specifies the channel property as the number of physical channels (p), the number of VCs (v), and the flit buffer depth (d).

To overcome the long global wire delay, repeaters are inserted to partition the wire into smaller segments, thereby making the delay linear with its length. Wire delay and power characteristics are drawn from Chapter II. Because the link power behavior is sensitive to value patterns, we consider the actual bit pattern crossing a link and the coupling effect on adjacent wires [87]. We divide the wire capacitance (c_w) into two parts: wire-substrate capacitance (c_s) and inter-wire capacitance (c_i). c_i is known to become more dominant than c_s as technology shrinks [89]. Thus we can drive energy drawn in a multi-wire link (E_{link}).

$$E_{link} = 0.5V_{dd}^2(\alpha(\frac{k_{opt}}{h_{opt}}(c_0 + c_p) + c_s) + \beta c_i)L, \tag{5.5}$$

where α and β are the transition counts for wire-substrate and inter-wire, respectively. At 45nm targeting year 2010, global wires having 135nm pitch has 198 fF/mm, where inter-wire capacitance is four times higher than wire-substrate capacitance. Table XIII (b) shows the delay and power models of the global wire.

Table XIII. Delay and Power for Interconnect for Data Compression

CMP	Channel (p, v, d)	Delay (ns)	Buffer (pJ)	Crossbar (pJ)	Arbiter (pJ)	Leakage (pJ)
TILED	6, 3, 4	0.250	11.48	34.94	0.22	9.05
SNUCA	8, 2, 4	0.230	15.30	61.23	0.32	15.12
	9, 2, 4	0.235	17.22	77.12	0.39	18.73

(a) Router

Delay	Dynamic power		Leakage power
	wire-substrate	inter-wire	one wire
183 (ps/mm)	1.135 (mW/mm)	0.634 (mW/mm)	0.0016 (mW/mm)

(b) Link

The benchmarks considered in this research are six parallel scientific (SPEComp) and two server (SPECjbb2000, SPECweb99) workloads. SPEComp programs are compiled on a Sun Studio 11 compiler and executed for parallel regions with reference data sets.

F. Experimental Results

We conducted experiments to examine how communication compression affects the performance and power consumption of on-chip interconnection networks. In a 64B cache block and 8B-wide channel networks, we assume that the address packet has a single flit and the data packet is broken down to nine flits, where the first flit has an address and other flits have the part of cache block data starting from its most significant bit position. We apply compression only to cache block data, because address

compression requires another table and does not give a high return for packet length reduction. Furthermore, we use four 2B-entry tables to compress the corresponding part of 8B flit data concurrently.

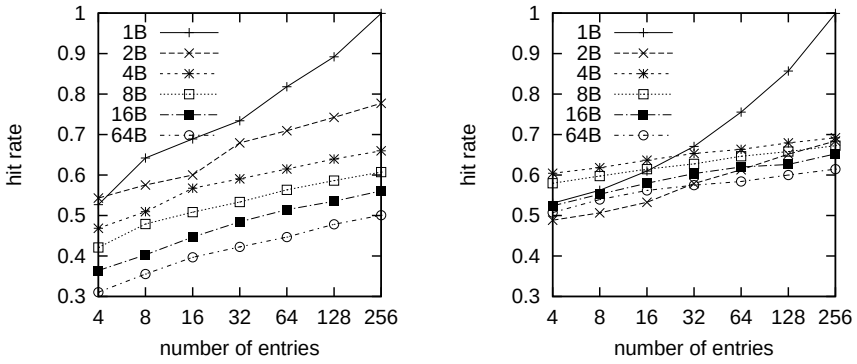
1. Compressibility and Value Pattern

Since one cache block contains 16 4B words, data redundancy can exist within a cache block. In addition, a value pattern detection method such as LRU and LFU affects compressibility. LRU replacement gives significance to the recently used one, while LFU replacement runs based on the reuse frequency. We put two fixed-size tables at sender and receiver sides like private table and use a hit rate as a compressibility metric. We changed the table size by varying the number of entries from 4 to 256 and the size of entry from 1B to 64B.

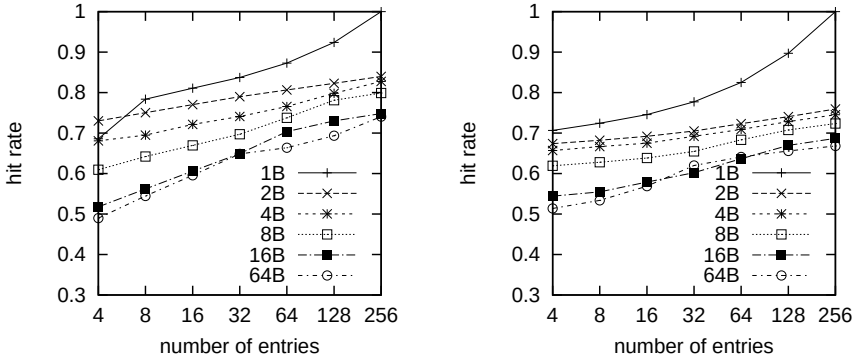
Figure 36 shows the trend of the average hit rate for two replacement policies [5]. As the size of entry is smaller or the number of entries is larger, the hit rate is better. For a fixed size table, making the entry size smaller increases the hit rate better than providing more number of entries due to the partial value redundancy. In 128B-table with LFU in TILED-CMP, 2Bx64 has 5%, 13%, and 30% higher hit rate than 4Bx32, 8Bx16, and 16Bx8, respectively. Although it is not easy to show which replacement policy is better in our experiments, LFU hit rate is less sensitive across the different number of entries in the table, which implies that multi-threaded programs have a set of frequent values like single-threaded programs [9]. This result shows high compressibility in cache traffic even with small tables.

Figure 37 compares compression ratios for three compression schemes: FPC (Frequent Pattern Compression), our table based approach, and LZW-variant compression algorithm. FPC compression algorithm detects six different sizes of zero data

[5]We put the value table at the router rather than each node in SNUCA-CMP.



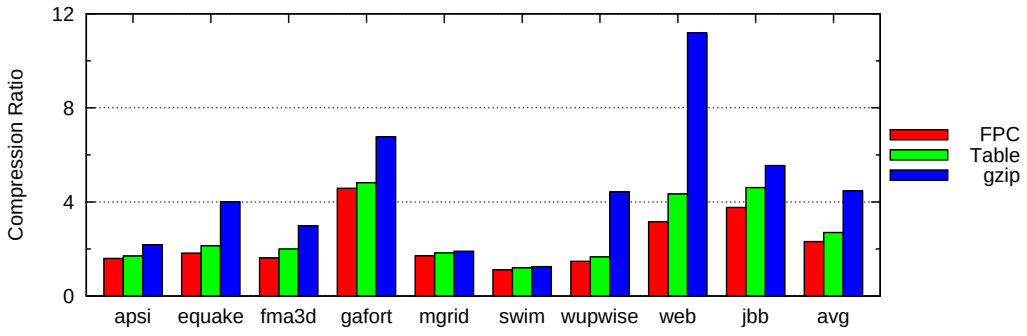
(a) LRU for SNUCA (b) LFU for SNUCA



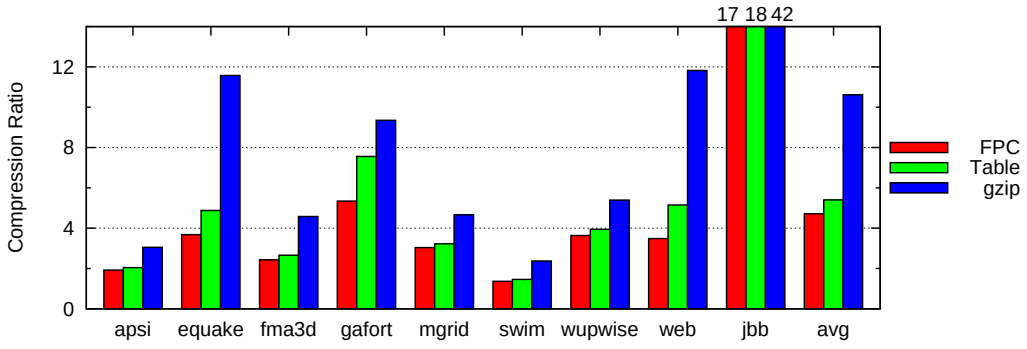
(c) LRU for TILED (d) LFU for TILED

Fig. 36. Communication Data Compressibility

and one word of repeated bytes. Therefore, it needs 3-bit prefix compression overhead for each word. We use the gzip unix utility for LZW compression, which combines the LZW deflate algorithm with Huffman encoding of codewords in the dictionary. Our table approach uses the 8-entry table, which needs a 3-bit index representation for compressed data. Additional overhead due to the compression is 1 bit compression status for each word. While our scheme and FPC achieves the fast compression, our table based compression provides 16% higher compression ratios than FPC. Compared with the idealistic compression algorithm, the table based approach achieves 56% of the gzip compression.



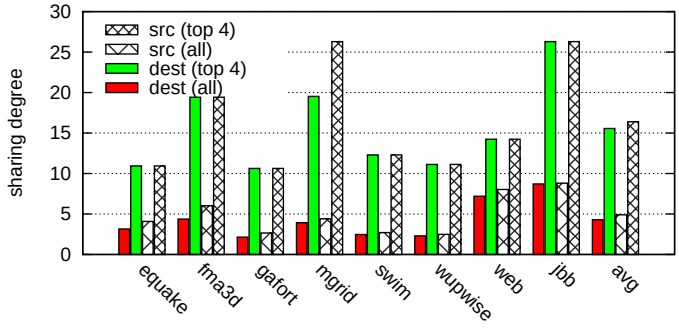
(a) SNUCA



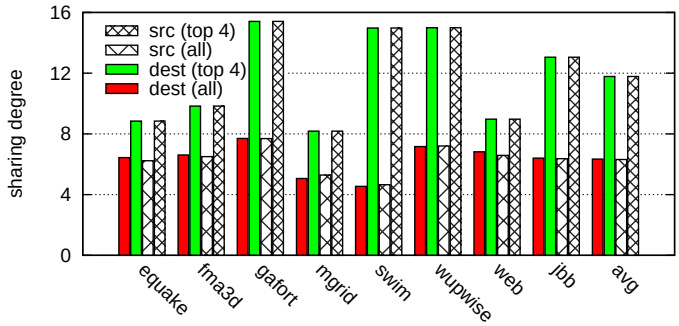
(b) TILED

Fig. 37. Compression Ratios

We examine value sharing property by analyzing values across different flows that have a common source (sender) or destination (receiver). The destination sharing degree is defined as the average number of destinations per value. For a 10K-cycle interval, we calculate the destination sharing degree for one source by taking the average number of destinations of each value, weighting it with the percentage of accesses that each value accounts for, and summing up the weighted destination counts. We finally take the average for all sources. Similarly, we obtain the source sharing degree. We do the same analysis considering only top n values ordered by the number of accesses.



(a) SNUCA-CMP



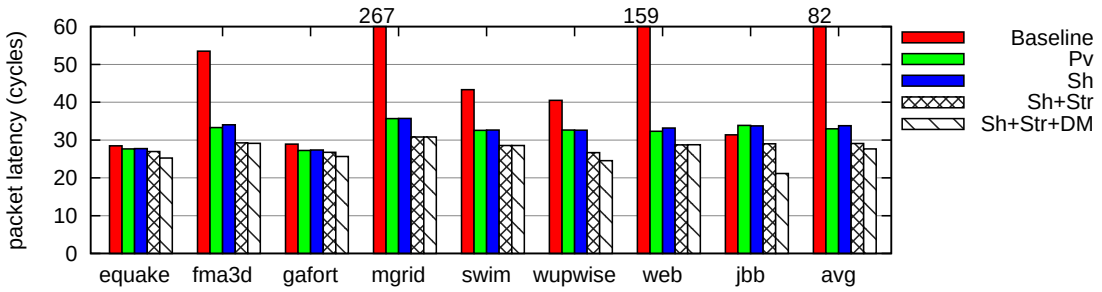
(b) TILED-CMP

Fig. 38. Value Spatial Distribution

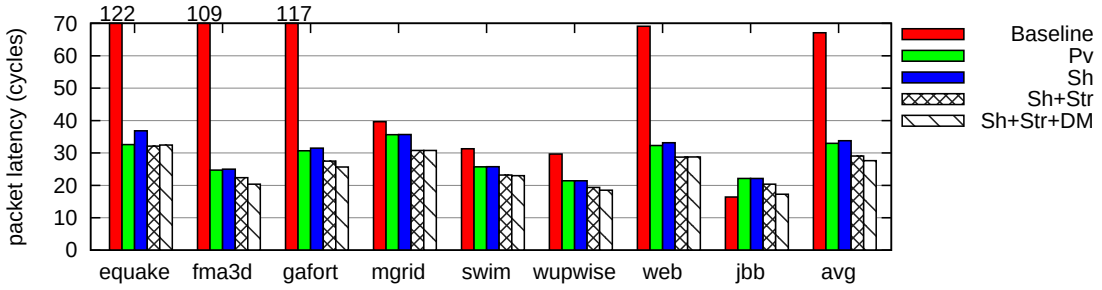
Figure 38 shows sharing degrees for 2B value in each benchmark. Regarding top (frequently accessed) four values shows much higher sharing degree than taking all values [6]. Particularly, TILED-CMP shows that top four values are involved almost 12 nodes (75% in the network). This result suggests that organizing encoding/decoding tables by sharing frequent values can keep a high compression rate. We select 2Bx8 table and LFU policy, which is fairly small but has a high hit rate, to evaluate our compression techniques further.

[6]The destination sharing degree (dest) is the average number of destinations for each value. The source sharing degree (src) is the average number of sources for each value.

2. Effect on Packet Latency



(a) SNUCA-CMP



(b) TILED-CMP

Fig. 39. Data Compression: Latency Comparison

Figure 39 shows the average packet latency of different compression architectures compared with the baseline. Private and shared table schemes are indicated by **Pv** and **Sh**. Streamlined encoding and dynamic compression management are indicated by **Str** and **DM**. A shared table has an 8-entry VLB in a decoding table. In most benchmarks, we can see that private table (second bar) achieves 51% and 60% latency reduction in TILED-CMP and SNUCA-CMP. Especially, compression improves the latency dramatically by resolving high congestion in some benchmarks (equake/fma3d/gafort/web in TILED-CMP and mgrid/web in SNUCA-CMP).

Shared table (third bar) achieves almost the same improvement: 50% in TILED-

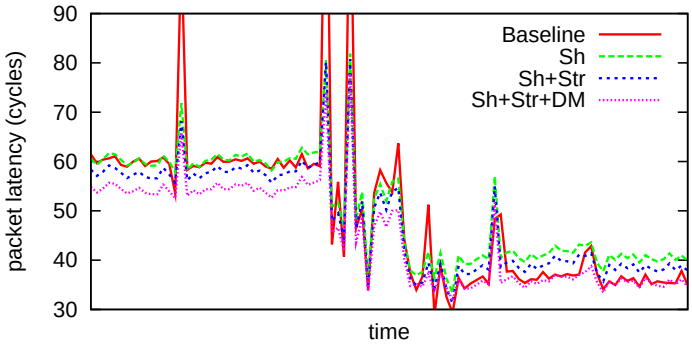
CMP and 59% SNUCA-CMP due to high value sharing as explained in Section 1. It penalizes the latency by only 4.7% (TILED-CMP) and 0.2% (SNUCA-CMP) over private table (second bar) from a compression hit loss. The average hit rate in encoding tables decreases 6.4% and 1.8% each. The hit rates are listed in Table XIV. We found that management traffic for the shared table scheme increases the overall traffic by less than 1%.

Table XIV. Encoding Table Hit Rates for Private and Shared Tables

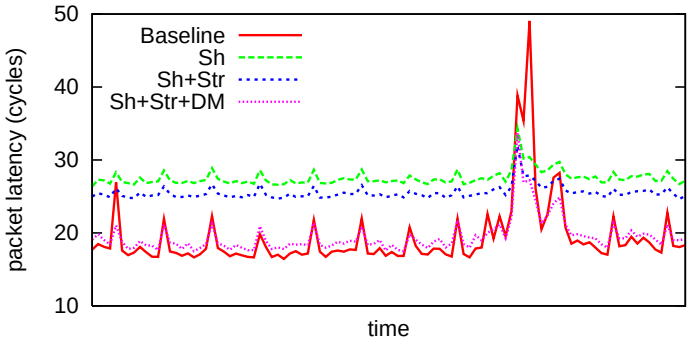
CMP arch.	scheme	equake	fma3d	gafort	mgrid	swim	wupwise	web	jbb	avg.
SNUCA	private	0.358	0.252	0.800	0.455	0.093	0.449	0.583	0.661	0.457
	shared	0.355	0.251	0.792	0.453	0.094	0.447	0.532	0.661	0.448
TILED	private	0.640	0.617	0.795	0.667	0.258	0.741	0.601	0.964	0.660
	shared	0.521	0.525	0.781	0.662	0.253	0.727	0.519	0.955	0.618

In Figure 39, streamlined encoding (fourth bar) reduces the latency for shared table (third bar) on average by 6% (4.3 cycles). Dynamic compression management (fifth bar) further reduces the latency by 3% (2.1 cycles). In summary, the design using all the techniques (fifth bar) improves the latency up to 88% with an average of 63% compared with the baseline (first bar).

Figure 40 shows the runtime latency behavior. As expected, compression increases the latency for the baseline in lightly loaded network, but it significantly decreases the latency by activating packet compression to congested paths. In contrast, dynamic compression management applies compression on-demand for time-varying workload and shows the finely tuned behavior. In other words, it follows the low latency in the baseline architecture by shutting off compression latency overhead in a light load, and lowers down the high latency by eliminating congestion in a high load.



(a) SNUCA-CMP (fma3d)



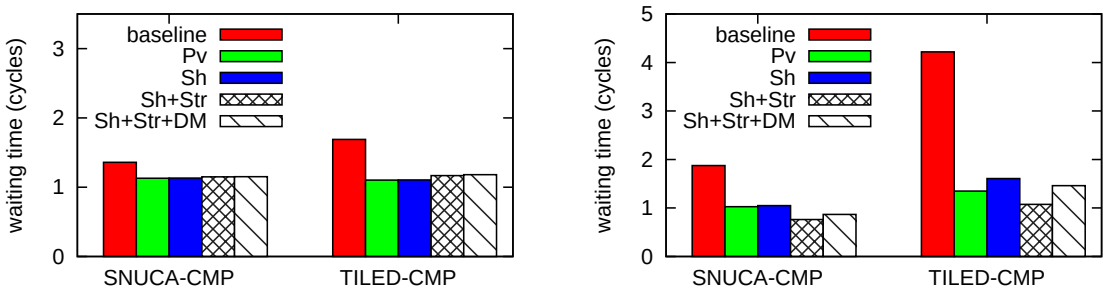
(b) TILED-CMP (wupwise)

Fig. 40. Behavior of Dynamic Compression Management

3. Effect on Contention in Router

We now investigate compression effect on contention in a router pipeline. In wormhole switching, a VC reserved by a head flit cannot be used until a tail flit releases it. The size-reduced packet from compression has less flits so that it can reduce the time to hold a VC for the original packet. For the same reason, packet compression reduces input/output port contention in a switch across competing flows stored in buffers.

Figure 41 illustrates the average waiting delay for VC allocation and switch allocation in five configurations. Each allocation operation takes one cycle in a contention-free router pipeline. When the speculative switch allocation is done with the VC allocation at the same cycle, we regard switch allocation delay as zero cycle.



(b) Virtual Channel Allocation (b) Switch Allocation

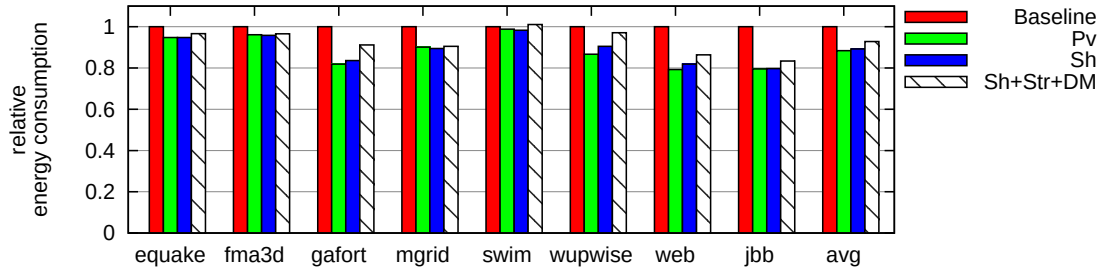
Fig. 41. Data Compression: Waiting Time in Router Components

In TILED-CMP, shared table (third bar) reduces VC allocation delay and switch allocation delay over the baseline (first bar) by 35% and 62%, respectively. In SNUCA-CMP, each delay is reduced by 35% and 44%. As we expected, streamlined encoding (fourth bar) increases the VC allocation delay slightly by 3.3% over non-streamlined encoding (third bar), because packets reserve the necessary VCs earlier but release them at almost the same time as the non-streamlined encoding scheme. Longer VA delay produces fewer competitors to switch ports, resulting in shorter switch allocation delay (i.e. switch contention reduction), because switch allocation is performed after VC allocation. Finally, the configuration using all the techniques (last bar) improves VC allocation delay by 30% and switch allocation delay by 60%.

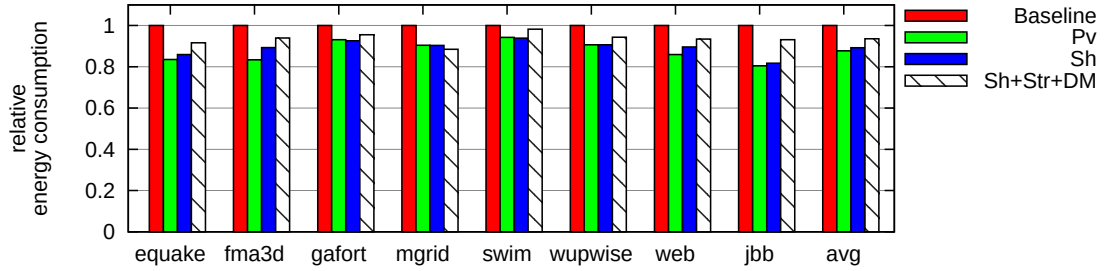
4. Effect on Network Power Consumption

Figure 42 shows energy reduction relative to the baseline [7]. Private table (second bar) and shared table (third bar) saves energy over the baseline (first bar) by 11.9% (TILED-CMP) and 12.2% (SNUCA-CMP) on average, respectively. As long-distance communication data is more involved with compression, energy saving is more effec-

[7]We omit **Sh+Str** that has almost the same result as **Sh**.



(a) SNUCA-CMP



(b) TILED-CMP

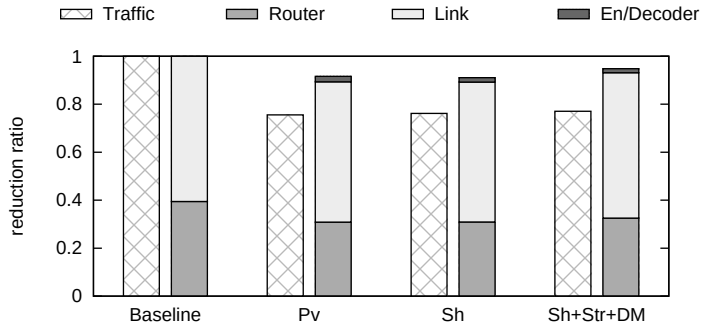
Fig. 42. Data Compression: Energy Comparison

tive. For example, fma3d exhibits the hop count as 3.16 and 17% energy saving for private table in TILED-CMP, while mgrid exhibits the hop count as 2.76 and 10% energy saving. Our examination in swim for very low energy saving exhibits that swim has a huge number of different values and a low hit rate on tables.

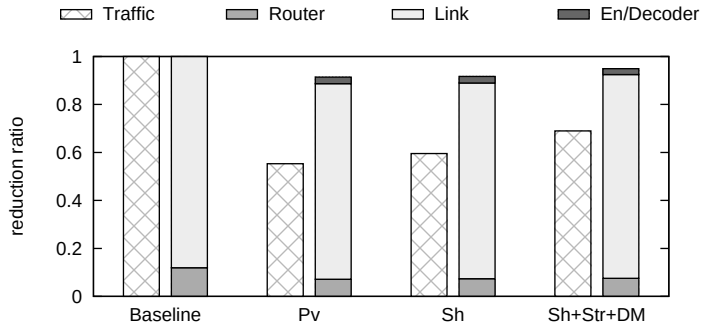
Dynamic compression management (fourth bar) reduces energy saving over shared table (third bar), because it applies compression adaptively and diminishes traffic volume reduction. Note that control packets are required for dynamic management. Energy saving is 6.4% in TILED-CMP and 7.1% in SNUCA-CMP.

In link power estimation, we find that using link utilization overestimates its energy consumption rather than accounting bit patterns. In benchmarks we examined, link utilization shows an average of 7% (up to 24%) while bit pattern analysis for

intra-wire switching gives us an average of 2% activity factor (up to 7%).



(a) SNUCA-CMP



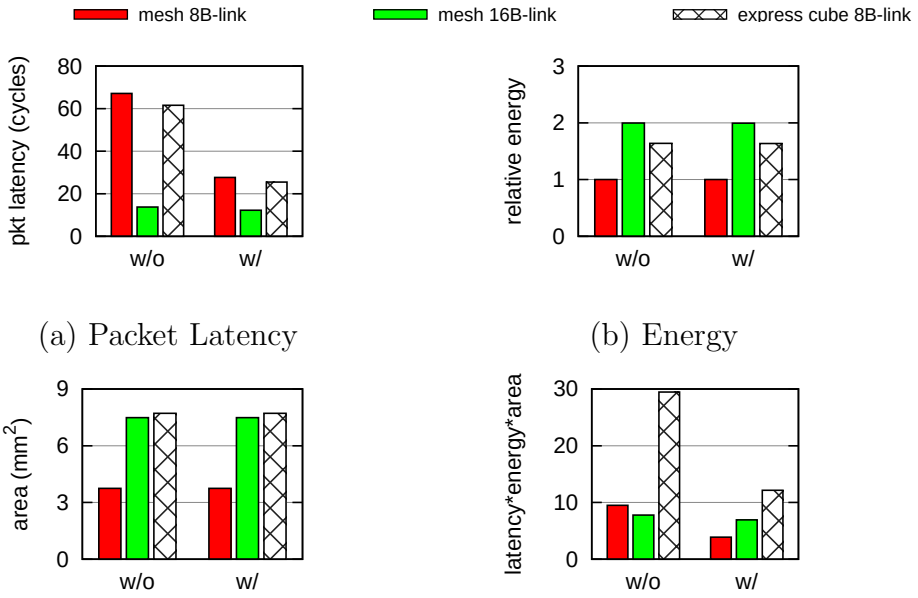
(b) TILED-CMP

Fig. 43. Data Compression: Traffic and Energy Relationship

Figure 43 shows a traffic (left bar) and energy (right bar) relationship in different schemes. It further breaks down each energy consumption in routers, links, and encoding/decoding tables. We observe that energy reduction is less than traffic reduction, because more repeated values in traffic implies smaller switching activity. Additionally, encoding indices from compression introduce a new pattern that is not in the original workload, causing extra switching activities. Routers consume 12% (TILED-CMP) and 39% (SNUCA-CMP) of the total network energy in baseline configurations. Note that SNUCA-CMP has higher-radix and more routers than

TILED-CMP. In fact, energy consumption ratio for each component depends on network parameters (buffer depth, router radix, link length) and workload characteristics (average hop count, bit pattern). In the shared table scheme, encoders and decoders consume only less than 3% (TILED-CMP) and 2% (SNUCA-CMP) of the total energy, because the table cost is minimized in our designs.

5. Comparison with Wide- and Long-Channel Networks



(a) Packet Latency

(b) Energy

(c) Area

(d) Overall Efficiency

Fig. 44. Data Compression: Network Comparison for TILED-CMP

We applied compression scheme (**Sh+Str+DM**) to a mesh network with 16B-wide links and an express cube [96] with one-hop express links. In the express cube, we use a deterministic routing algorithm that first uses the express channels and then regular channels. Figure 44 shows latency, energy, area, and overall efficiency

as a product of them. Left (w/o) and right (w/) groups represent the baseline and compression architectures each. Compression improves the latency on average by 43% in all networks. It should be mentioned that using deterministic routing in the express cube does not fully utilize large path diversity. Since dynamic energy contributes the small portion of the total energy, compression does not save much energy. Moreover, wide and long channels increase static energy consumption and take larger area. Figure 44 (c) shows that the table area is almost negligible (less than 0.01% in total area). In summary, compression improves the overall efficiency by 11% in the 16B-wide link mesh, and by 59% in the express cube.

CHAPTER VI

CONCLUSIONS

The computer industry has seen a shift from using abundant on-chip resources in monolithic designs to replicating components in modular designs. This transition to multi-core processors places significant challenge on interconnect system design. On-chip interconnection networks are a key component toward achieving high bandwidth, low latency, and lower power consumption. This dissertation makes several contributions in the space of communication-centric chips.

First, we recognize that interconnection network designs are critical for performance in a large-scale L2 cache. We explore a communication design for LRU replacement and a network topology development. Specifically, our contribution includes: (i) a single-cycle router architecture with multicast support as the basic building block of the interconnection networks; (ii) Fast-LRU replacement that can reduce the network latency; (iii) appropriate deadlock-free XYX routing algorithm that requires no horizontal links in a mesh except the first row to save area and power; (iv) a new network topology, called a *halo* network, where the MRU banks are of the same distance from the core; and (v) a halo network with non-uniform sized banks, thus reducing the wasted area on the processor die.

Second, we contribute the characterization methodology of on-chip communications for parallel applications in tiled CMPs. From initial analysis, it is hard to summarize overall temporal/spatial behavior of an application as a single inter-arrival time distribution or a particular traffic pattern. For the time-varying and spatio-temporal characteristics, we conduct a phase-based analysis that is widely accepted in the microprocessor domain. We introduce a phase-based characterization using a machine learning approach. We observe that the program has a recurring behavior

due to the loop structure and its performance is determined by the performance of loop execution. We introduce a reconfigurable long-channel architecture that utilizes the characterized communication behavior for one iteration of the loop. Mapping high-load sources or flows to long channels reduces the latency by bypassing intermediate routers on the delivery path.

Third, we explore data compression application for NoC-enabled multi-core systems. We introduce a table-based compression scheme utilizing frequent value patterns. For a low cost of the table implementation, we propose the shared table scheme, which stores identical values into a single entry from different sources or destinations and removes the network-size dependence. We also present the efficient table management protocol for consistency. For a compression latency overhead, we present two performance enhancement schemes. Streamlined encoding reduces the encoding latency by overlapping encoding with flit injection. Applying dynamic compression management increases performance, especially when congestion occurs in a part of the network.

Looking to the future, there are significant opportunities in emerging technologies, workload characterization, and increased hardware support for communication. Leveraging the expertise in interconnection networks and many previous established work will open a new challenge and need innovative idea to redesign the entire system. As evidenced by my dissertation research that applies interconnection knowledge to the cache design value-based communication, co-designing two subsystems in a unified way is critical in multi-core chips.

REFERENCES

[1] R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," in *Proceedings of the IEEE*, pp. 490–504, 2001.

[2] SIA, "International Technology Roadmap for Semiconductors," 2005. http://public.itrs.net.

[3] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[4] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[5] M. B. Taylor, W. Lee, S. P. Amarasinghe, and A. Agarwal, "Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architecture," in *Proceedings of HPCA*, pp. 341–353, 2003.

[6] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture," in *Proceedings of ISCA*, pp. 422–433, 2003.

[7] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. G. McDonald, S. W. Keckler, and D. Burger, "Implementation and Evaluation of a Dynamically Routed Processor Operand Network," in *Proceedings of NOCS*, pp. 7–17, 2007.

[8] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches," in *Proceedings of ASPLOS*, pp. 211–222, 2002.

[9] Y. Zhang, J. Yang, and R. Gupta, "Frequent Value Locality and Value-Centric Data Cache Design," in *Proceedings of ASPLOS*, pp. 150–159, 2000.

[10] A. R. Alameldeen and D. A. Wood, "Adaptive Cache Compression for High-Performance Processors," in *Proceedings of ISCA*, pp. 212–223, 2004.

[11] A. R. Alameldeen and D. A. Wood, "Interactions Between Compression and Prefetching in Chip Multiprocessors," in *Proceedings of HPCA*, pp. 228–239, 2007.

[12] R. Das, A. K. Mishra, C. Nicopolous, D. Park, V. Narayan, R. Iyer, M. S. Yousif, and C. R. Das, "Performance and Power Optimization through Data Compression in Network-on-Chip Architectures," in *Proceedings of HPCA*, pp. 215–225, 2008.

[13] J. Balfour and W. J. Dally, "Design Tradeoffs for Tiled CMP On-Chip Networks," in *Proceedings of ICS*, pp. 187–198, 2006.

[14] J. Kim, J. Balfour, and W. J. Dally, "Flattened Butterfly Topology for On-Chip Networks," in *Proceedings of MICRO*, pp. 172–182, 2007.

[15] M. M. Kim, J. D. Davis, and T. Austin, "Polymorphic On-Chip Networks," in *Proceedings of ISCA*, pp. 101–112, 2008.

[16] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco: Morgan Kaufmann, 2003.

[17] R. D. Mullins, A. West, and S. W. Moore, "Low-Latency Virtual-Channel Routers for On-Chip Networks," in *Proceedings of ISCA*, pp. 188–197, 2004.

[18] M. S. Hrishikesh, D. Burger, S. W. Keckler, P. Shivakumar, N. P. Jouppi, and K. I. Farkas, "The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays," in *Proceedings of ISCA*, pp. 14–24, 2002.

[19] M. Galles, "Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip," in *Proceedings of Hot Interconnect*, pp. 141–146, 1996.

[20] L.-S. Peh and W. J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," in *Proceedings of HPCA*, pp. 255–266, 2001.

[21] Y. Tamir and G. L. Frazier, "High-Performance Multi-Queue Buffers for VLSI Communication Switches," in *Proceedings of ISCA*, pp. 343–354, 1988.

[22] C. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers," in *Proceedings of MICRO*, pp. 333–346, 2006.

[23] D. N. Jayasimha, B. Zafar, and Y. Hoskote, "Interconnection Networks: Why They are Different and How to Compare Them," tech. rep., Microprocessor Technology Lab, Corporate Technology Group, Intel Corp, 2007. http://blogs.intel.com/research/terascale/ODI_why-different.pdf.

[24] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI.* MA: Addison-Wesley, 1990.

[25] "Predictive Technology Model (PTM)," 2007. `http://www.eas.asu.edu/~ptm`.

[26] M. L. Mui and K. Banerjee, "A Global Interconnect Optimization Scheme for Nanometer Scale VLSI with Implications for Latency, Bandwidth, and Power Dissipation," *IEEE Transaction on Electron Devices*, vol. 51, no. 2, pp. 195–203, 2004.

[27] B. M. Beckmann and D. A. Wood, "Managing Wire Delay in Large Chip-Multiprocessor Caches," in *Proceedings of MICRO*, pp. 319–330, 2004.

[28] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA Substrate for Flexible CMP Cache Sharing," in *Proceedings of ICS*, pp. 31–40, 2005.

[29] T. W. Ainsworth and T. M. Pinkston, "Characterizing the Cell EIB On-Chip Network," *IEEE Micro*, vol. 27, no. 5, pp. 6–14, 2007.

[30] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, 2005.

[31] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures," in *Proceedings of ISCA*, pp. 248–259, 2000.

[32] M. Lajolo, M. S. Reorda, and M. Violante, "Early Evaluation of Bus Interconnects Dependability for System-on-Chip Designs," in *Proceedings of Fourteenth International Conference on VLSI Design*, pp. 371–376, 2001.

[33] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proceedings of DAC*, pp. 684–689, 2001.

[34] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[35] G. Varatkar and R. Marculescu, "Traffic Analysis for On-Chip Networks Design of Multimedia Applications," in *Proceedings of DAC*, pp. 795 –800, 2002.

[36] M. Forsell, "A Scalable High-Performance Computing Solution for Networks on Chips," *IEEE Micro*, vol. 22, pp. 46 –55, Sep/Oct 2002.

[37] M. B. Taylor, W. Lee, S. P. Amarasinghe, and A. Agarwal, "Scalar Operand Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 145–162, 2005.

[38] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture," in *Proceedings of ISCA*, pp. 161–171, 2000.

[39] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures," in *Proceedings of MICRO*, pp. 55–66, 2003.

[40] B. M. Beckmann and D. A. Wood, "TLC: Transmission Line Caches," in *Proceedings of MICRO*, pp. 43–54, 2003.

[41] N. Muralimanohar and R. Balasubramonian, "Interconnect Design Considerations for Large NUCA Caches," in *Proceedings of ISCA*, pp. 369–380, 2007.

[42] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches With CACTI 6.0," in *Proceedings of MICRO*, pp. 3–14, 2007.

[43] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs," in *Proceedings of ISCA*, pp. 357–368, 2005.

[44] W. H. Ho and T. M. Pinkston, "A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns," in *Proceedings of HPCA*, pp. 377–388, 2003.

[45] J. Hu and R. Marculescu, "Application-Specific Buffer Space Allocation for Networks-on-Chip Router Design," in *Proceedings of ICCAD*, pp. 354–361,

November 2004.

[46] C. B. Stunkel, R. Sivaram, and D. K. Panda, "Implementing Multidestination Worms in Switch-Based Parallel Systems: Architectural Alternatives and their Impact," in *Proceedings of ISCA*, pp. 50–61, 1997.

[47] X. Lin and L. M. Ni, "Deadlock-Free Multicast Wormhole Routing in Multicomputer Networks," in *Proceedings of ISCA*, pp. 116–125, 1991.

[48] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," in *Proceedings of ISCA*, pp. 278–287, 1992.

[49] R. D. Mullins, A. West, and S. W. Moore, "The Design and Implementation of a Low-Latency On-chip Network," in *Proceedings of ASP-DAC*, pp. 164–169, 2006.

[50] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express Virtual Channels: Towards the Ideal Interconnection Fabric," in *Proceedings of ISCA*, pp. 150–161, 2007.

[51] R. Desikan, D. Burger, S. Keckler, and T. Austin, "Sim-alpha: A Validated, Execution-Driven Alpha 21264 Simulator," Tech. Rep. TR-01-23, The University of Texas at Austin, Department of Computer Sciences, 2001.

[52] R. E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, 1999.

[53] P. Shivakumar and N. P. Jouppi, "Cacti 3.0: An Integrated Cache Timing, Power and Area Model," Tech. Rep. WRL-2001-2, Compaq Computer Corporation, 2001.

[54] R. H. J. M. Otten and R. K. Brayton, "Planning for Performance," in *Proceedings of DAC*, pp. 122–127, 1998.

[55] SIA, "International Technology Roadmap for Semiconductors," 2003. http://public.itrs.net.

[56] H. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: a Power-Performance Simulator for Interconnection Networks," in *Proceedings of MICRO*, pp. 294–305, 2002.

[57] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter, "Interconnect-Aware Coherence Protocols for Chip Multiprocessors," in *Proceedings of ISCA*, pp. 339–351, 2006.

[58] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *Proceedings of ASPLOS*, pp. 45–57, 2002.

[59] T. Sherwood, S. Sair, and B. Calder, "Phase Tracking and Prediction," in *Proceedings of ISCA*, pp. 336–347, 2003.

[60] A. Dhodapkar and J. E. Smith, "Managing Multi-Configuration Hardware via Dynamic Working Set Analysis," in *Proceedings of ISCA*, pp. 233–244, 2002.

[61] X. Shen, Y. Zhong, and C. Ding, "Locality Phase Prediction," in *Proceedings of ASPLOS*, pp. 165–176, 2004.

[62] C. Isci and M. Martonosi, "Phase Characterization for Power: Evaluating Control Flow-Based and Event-Counter-Based Techniques," in *Proceedings of HPCA*, pp. 122–133, 2006.

[63] S. Chodnekar, V. Srinivasan, A. S. Vaidya, A. Sivasubramaniam, and C. R. Das, "Towards a Communication Characterization Methodology for Parallel Applications," in *Proceedings of HPCA*, pp. 310–319, 1997.

[64] W. Heirman, J. Dambre, J. V. Campenhout, C. Debaes, and H. Thienpont, "Traffic Temporal Analysis for Reconfigurable Interconnects in Shared-Memory Systems," in *Proceedings of IPDPS - Workshop 3*, 2005.

[65] G. A. Abandah and E. S. Davidson, "Configuration Independent Analysis for Characterizing Shared-Memory Applications," in *Proceedings of IPDPS*, pp. 485–491, 1998.

[66] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings of ISCA*, pp. 24–36, 1995.

[67] L. A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," in *Proceedings of ISCA*, pp. 3–14, 1998.

[68] J. P. Singh, E. Rothberg, and A. Gupta, "Modeling Communication in Parallel Algorithms: A Fruitful Interaction Between Theory and Systems?" in *Proceedings of SPAA*, pp. 189–199, 1994.

[69] K. L. Johnson, "The Impact of Communication Locality on Large-Scale Multiprocessor Performance," in *Proceedings of ISCA*, pp. 392–402, 1992.

[70] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," in *Proceedings of ISCA*, pp. 336–345, 2005.

[71] V. Soteriou, H. Wang, and L.-S. Peh, "A Statistical Traffic Model for On-Chip Interconnection Networks," in *Proceedings of MASCOTS*, pp. 104–116, 2006.

[72] G. Varatkar and R. Marculescu, "Traffic Analysis for On-Chip Networks Design of Multimedia Applications," in *Proceedings of DAC*, pp. 795–800, 2002.

[73] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[74] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[75] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, vol. 36, no. 5, pp. 547–553, 1987.

[76] S. Hauck, G. Borriello, and C. Ebeling, "Mesh Routing Topologies for Multi-FPGA Systems," *IEEE Trans. VLSI Systems*, vol. 6, no. 3, pp. 400–408, 1998.

[77] C. E. Leiserson, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Trans. Computers*, vol. 34, no. 10, pp. 892–901, 1985.

[78] SPEC, "SPEC OMP (OpenMP Benchmark Suite)," 2001. http://www.spec.org.

[79] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern and Classification*. New York: John Wiley & Sons, 2001.

[80] S. D. Kamvar, D. Klein, and C. D. Manning, "Interpreting and Extending Classical Agglomerative Clustering Algorithms using a Model-Based Approach," in *Proceedings of ICML*, pp. 283–290, 2002.

[81] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger, "Implementation and Evaluation of On-Chip Network Architectures," in *Proceedings of ICCD*, pp. 477–484, 2006.

[82] P. Abad, V. Puente, J.-Á. Gregorio, and P. Prieto, "Rotary Router: An Efficient Architecture for CMP Interconnection Networks," in *Proceedings of ISCA*, pp. 116–125, 2007.

[83] E. G. Hallnor and S. K. Reinhardt, "A Unified Compressed Memory Hierarchy," in *Proceedings of HPCA*, pp. 201–212, 2005.

[84] D. Citron and L. Rudolph, "Creating a Wider Bus Using Caching Techniques," in *Proceedings of HPCA*, pp. 90–99, 1995.

[85] K. Basu, A. N. Choudhary, J. Pisharath, and M. T. Kandemir, "Power Protocol: Reducing Power Dissipation on Off-Chip Data Buses," in *Proceedings of MICRO*, pp. 345–355, 2002.

[86] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value Locality and Load Value Prediction," in *Proceedings of ASPLOS*, pp. 138–147, 1996.

[87] P.-P. Sotiriadis and A. Chandrakasan, "Bus Energy Minimization by Transition Pattern Coding (TPC) in Deep Submicron Technologies," in *Proceedings of ICCAD*, pp. 322–327, 2000.

[88] T. Lv, J. Henkel, H. Lekatsas, and W. Wolf, "A Dictionary-Based En/Decoding Scheme for Low-Power Data Buses," *IEEE Trans. VLSI Syst*, vol. 11, no. 5,

pp. 943–951, 2003.

[89] V. Wen, M. Whitney, Y. Patel, and J. Kubiatowicz, "Exploiting Prediction to Reduce Power on Buses," in *Proceedings of HPCA*, pp. 2–13, 2004.

[90] M. Stan and W. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE Transaction on VLSI*, vol. 3, no. 1, pp. 49–58, 1995.

[91] J. Yang, R. Gupta, and C. Zhang, "Frequent Value Encoding for Low Power Data Buses," *ACM Trans. Design Autom. Electr. Syst.*, vol. 9, no. 3, pp. 354–384, 2004.

[92] M. Zhang and K. Asanovic, "Highly-Associative Caches for Low-Power Processors," in *Kool Chips Workshop, MICRO-33*, 2000.

[93] A. Agarwal, K. Roy, and T. N. Vijaykumar, "Exploring High Bandwidth Pipelined Cache Architecture for Scaled Technology," in *Proceedings of DATE*, pp. 10778–10783, 2003.

[94] N. Kirman, M. Kirman, R. K. Dokania, J. F. Martínez, A. B. Apsel, M. A. Watkins, and D. H. Albonesi, "Leveraging Optical Technology in Future Bus-based Chip Multiprocessors," in *Proceedings of MICRO*, pp. 492–503, 2006.

[95] D. Tarjan, S. Thoziyoor, and N. P. Jouppi, "Cacti 4.0.," Tech. Rep. HPL-2006-86, HP Laboratories, 2006.

[96] W. J. Dally, "Express Cubes: Improving the Performance of k-Ary n-Cube Interconnection Networks," *IEEE Trans. Computers*, vol. 40, no. 9, pp. 1016–1023, 1991.

VITA

Name:	Yu Ho Jin
Address:	Department of Computer Science and Engineering
	Texas A&M University
	College Station, TX 77843-3112
Email Address:	yuho@cse.tamu.edu
Education:	B.S., Computer Science, KAIST, Korea, 1995
	M.S., Computer Science, KAIST, Korea, 1999
	Ph.D., Computer Engineering, Texas A&M University, 2009