Adaptive Data Compression for High-Performance Low-Power On-Chip Networks

Yuho Jin* Ki Hwan Yum† Eun Jung Kim*

*Department of Computer Science
Texas A&M University
College Station, TX 77843 USA
{yuho,ejkim}@cs.tamu.edu

†Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249 USA
yum@cs.utsa.edu

Abstract

With the recent design shift towards increasing the number of processing elements in a chip, high-bandwidth support in on-chip interconnect is essential for low-latency communication. Much of the previous work has focused on router architectures and network topologies using wide/long channels. However, such solutions may result in a complicated router design and a high interconnect cost. In this paper, we exploit a table-based data compression technique, relying on value patterns in cache traffic. Compressing a large packet into a small one can increase the effective bandwidth of routers and links, while saving power due to reduced operations. The main challenges are providing a scalable implementation of tables and minimizing overhead of the compression latency. First, we propose a shared table scheme that needs one encoding and one decoding tables for each processing element, and a management protocol that does not require in-order delivery. Next, we present streamlined encoding that combines flit injection and encoding in a pipeline. Furthermore, data compression can be selectively applied to communication on congested paths only if compression improves performance. Simulation results in a 16-core CMP show that our compression method improves the packet latency by up to 44% with an average of 36% and reduces the network power consumption by 36% on average.

1. Introduction

With a current processor design trend, future chip multiprocessors (CMPs) will accommodate many cores and large caches for high performance. This ever-increasing integration of components in a single chip has embarked the beginning of interconnect-centric designs [1]–[4]. At the same time, as the shrinking technology continues to exacerbate the imbalance between gate and wire in terms of delay and power, on-chip interconnects are predicted to be a crucial factor in designing a CMP system [5], [6]. On-chip networks have been gaining wide acceptance for the communication architecture in decentralized designs such as Intel 80-core Teraflop [7], Tilera 64-core [8], TRIPS [9], and RAW [10].

The design of a low-latency on-chip network is critical to provide 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 [11], [12] and network topologies [13]. However, due to the stringent 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 [14]. Therefore, we focus on maximizing bandwidth utilization in the existing network.

Data compression has been adopted in hardware designs to improve performance and power. Cache compression increases the cache capacity by compressing block data and accommodating more blocks in a fixed space [15], [16]. Bus compression also expands the bus width by encoding a wide data as a small size code [17], [18]. Recently data compression is explored in the on-chip network domain for performance and power [4].

In this paper, 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 processes diverse value patterns adaptively rather than taking static patterns based on zero bits in a word [4]. The table-based approach can easily achieve a better compression rate by increasing the table size. However, the table for compression requires a huge area to keep data values on a flow [1] basis. In other words, the number of tables depends on the network size, because communication cannot be globally managed in a switched network. To address this problem, we present a shared table scheme that stores 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 negative impact of compression on performance.

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

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.

Our contributions regarding data compression in on-chip networks are summarized as follows:

- We introduce an adaptive table-based data compression framework in on-chip network architectures.
- We explore compression optimization techniques; implementing a scalable and area-efficient table, reducing the encoding latency by integrating encoding into flit injection, and dynamically applying compression depending on workload.
- We identify 69% compressibility of cache traffic in a suite of scientific and commercial benchmarks for a 16-core tiled CMP design. Additionally, a large portion of communication data (40% for all values and 74% for top four most frequently values) is shared by nodes in the network.
- The detailed simulation results show that data compression improves the latency by 31% and saves the energy by 36%. Furthermore, dynamic compression management achieves 36% latency improvement with 4% energy saving.

The paper is organized as follows. We briefly discuss related work in Section 2. We describe a table-based data compression in Section 3. Compression optimization schemes are presented in Section 4. In Sections 5 and 6, we describe evaluation methodologies and present simulation results, followed by the conclusion in Section 7.

2. Related Work

This research is motivated by a large body of prior work in architectures that utilize data values produced by applications. 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 [19]. Furthermore it is shown that programs have a set of frequent values across all load and store instructions [20].

Cache compression: Cache compression has been proposed to expand the cache capacity by packing more blocks than given by the space [15], [16]. Alameldeen, et al. used the frequent pattern compression (FPC) scheme to store a variable number of blocks in the data array of the L2 cache [15]. 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 a 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 [17], [18]. Bus-Expander stores the repeated high order bits of data into a table [17]. For one data transfer, the index in 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 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.

Bus encoding techniques have been proposed to reduce the energy consumed in high-capacitance buses [21]–[23]. 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.

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 negative impact on performance.

3. 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.

3.1. On-Chip Network Architecture

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

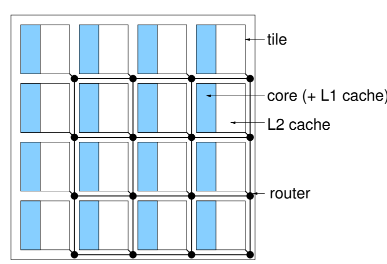


Figure 1. On-Chip Networks in CMPs

Figure 1 shows a 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

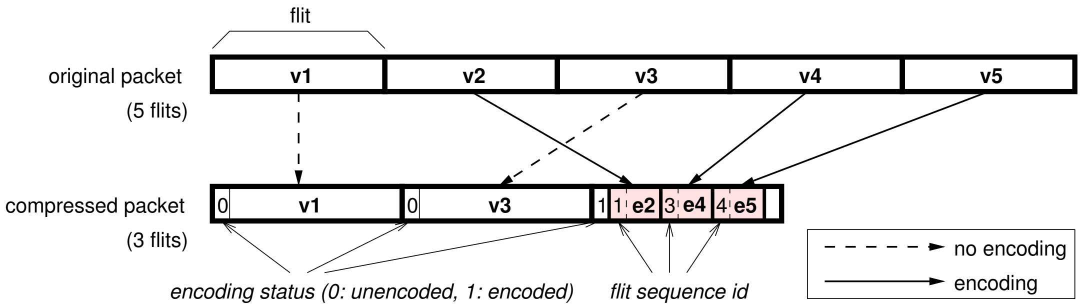


Figure 2. Packet Compression: Among five flits in a packet, three flits are encoded and compressed into the last flit. Compression requires an extra bit in the flit for an encoding status.

local ports for L1/L2 caches and a distributed directory, and four ports to neighbor tiles for a mesh network.

In this on-chip network, 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. Therefore, 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 a 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) [24]. 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 in the ST stage.

In our study, we use a 2-stage pipeline, which adopts lookahead routing and speculative switch allocation [25]. 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, flit fragmentation/assembly for flow control, and other 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 increases link utilization and allows more metal resources for power and ground interconnects. Buffered wires are used to fit a link delay within a single cycle.

3.2. 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 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 packet 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.

Figure 2 shows an example of how to encode a packet payload as flits. We assume that the value size for an encoding operation is the same as the flit size. In a compressed packet, encoded indices (`e2`, `e4`, `e5`) follow unencoded data (`v1`, `v3`). This structure enables multiple flits to be successfully packed into a flit and simplifies encoded index alignment with unencoded values in a flit. 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. Here, we do not consider a specific energy-aware coding when building an index [21]–[23], because sharing links for different flows makes it hard to predict wire switching activities. Next, we explain table organization schemes to store recurring values.

3.3. 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. The decoding table is simply organized as a direct-mapped cache. 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 an 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.

4. Optimizing Compression

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

4.1. 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 reveals that one sender transmits the same value to a large portion of receivers and vice versa (See the detailed results in Section 6.1). 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 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

value	index vector															
A	00				00			01	01				00		00	
B	11	10		00		01		00	11		11	10		01	10	11
C		11				10					01	01				10
D	10		01			00			10			00	11			

for PE8

Each element shows a binary index for value at decoder.

(a) Encoding Table in PE4

value	index	use–bit vector															
E	00	1	1	0	0	0	1	1	0	1	0	0	0	0	1	0	0
A	01	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	1
D	10	1	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0
B	11	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	1

for PE4

Each bit indicates a value status of encoder.

(b) Decoding Table in PE8

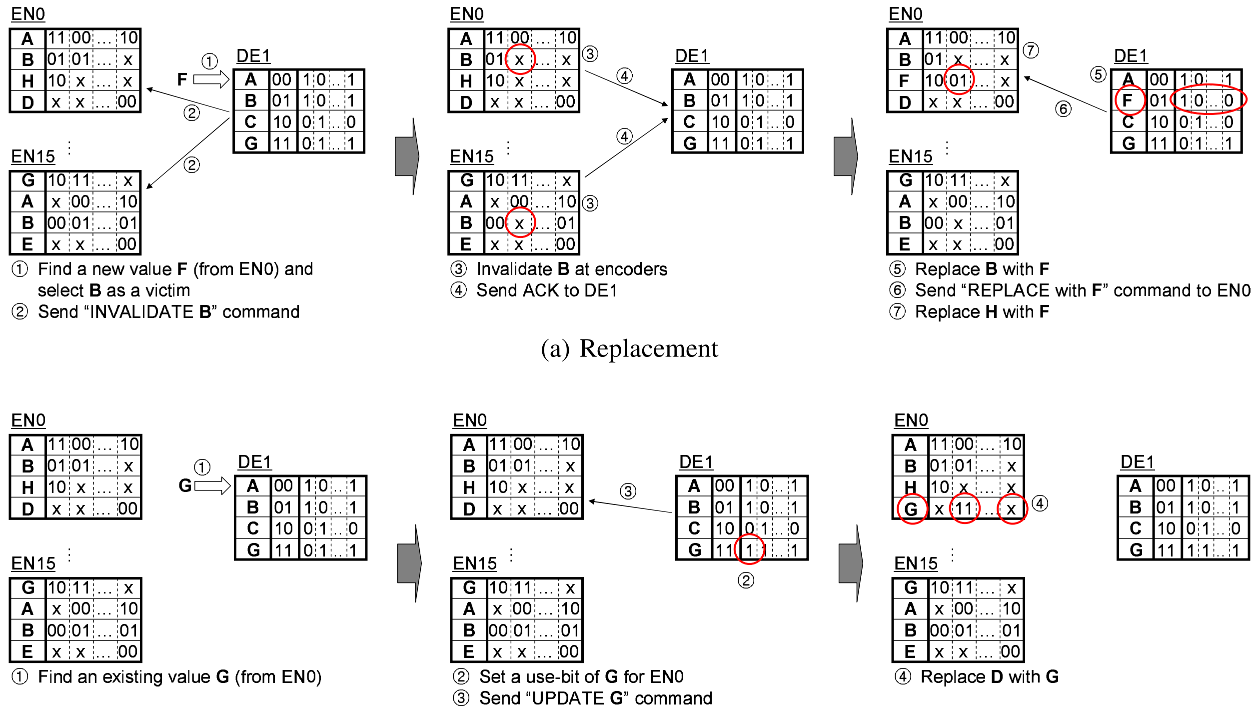
Figure 3. Shared Table Structure: In an N-PE system, an index vector of the encoding table has N indices for each decoding table. A use-bit vector of the decoding table has N bits to indicate the value status of each encoding table.

tells if the corresponding sender transmits the associated value as index. Figure 3 shows the structure of 4-entry tables 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 (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. A sender must not transmit an index from which a receiver cannot restore the correct value in its decoding table. Moreover, an index associated with a value in the decoding table is used for multiple encoding tables. Changing a value associated with an index in the decoding table requires some actions in encoding 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 the 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.

Figure 4 (a) illustrates a replacement example with two



(a) Replacement

(b) Update

Figure 4. Shared Table Management

encoding tables (EN0 for PE0 and EN15 for 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 4 (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. 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.

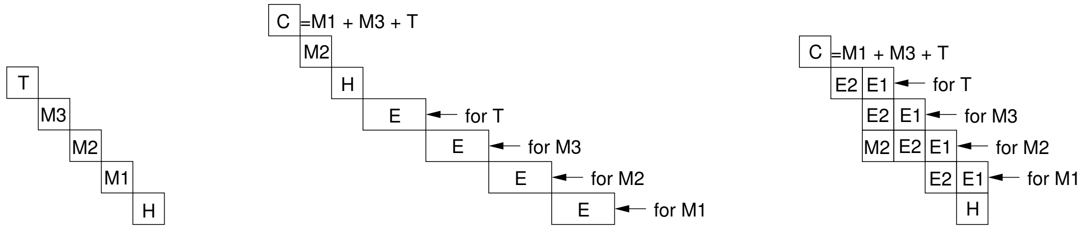
Increasing Compression Effectiveness: Because a single decoding table handles value locality from multiple flows, the shared table may experience many replacement operations, causing a low compression rate. One replacement in a decoding table requires at least three packet transmissions to be consistent 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 hit counter. When a value arrives at a receiver, VLB and the decoding table are accessed together. In case of a hit in the decoding table, no action occurs in VLB. In case of a miss in the decoding table, VLB needs replacement for a miss or increases the counter of a hit entry. When a counter is saturated, the associated value results in replacement for the decoding table. Therefore, VLB is used for confirming temporal locality of new values.

Table 1. Table Area for a Single PE: N, v, and e/d are the number of PEs, the size of value in bits, the number of entries in encoding/decoding tables.

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

Table Area: The associative search part of a table needs CAM implementation and other parts are constructed as RAM. To compare costs of the private and shared table



(a) No Encoding - 5 cycles (b) Encoding - 11 cycles (c) Streamlined Encoding - 6 cycles

Figure 5. Streamlined Encoding : (a) A 5-flit (H, M1, M2, M3, and T) packet needs 5 cycles for flit injection. (b) 2-cycle encoding operation requires total 8 cycles for encoding except a head flit. (c) A pipeline is constructed as two stages for encoding and one stage for injection, which takes total 6 cycles.

schemes, we estimate the total area of CAM and RAM. Table 1 shows the required number of 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 [26]. It gives us 0.638 μm^2 for RAM (6 transistors) and 1.277 μm^2 for CAM (9 transistors). Figure 6 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 16 (Sh d=16) and 32 (Sh d=32), increased area for the shared table scheme is still scalable.

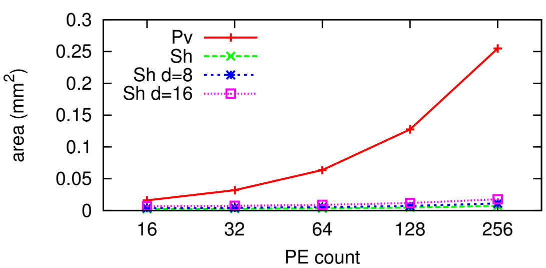


Figure 6. Area Comparison

4.2. Streamlined Encoding

To hide 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 the encoding table and multiple table accesses increase the overall network latency. Furthermore, we exploit the fact that multi-cycle encoding for the table access can be enhanced using a cache pipelining technique [27]. Integrating flit injection and encoding processes into a pipeline can make each process work concurrently.

Figure 5 shows the latency benefit of streamlined encoding for a 5-flit packet, which requires 5 cycles for flit injection (Figure 5 (a)). Figure 5 (b) shows that 8 cycles are required to encode for four flits (M1, M2, M3, T) when the encoding table access needs 2 cycles. Then, injection takes 3 cycles for two uncompressed flits (H, M2) and one compressed flit (C). Figure 5 (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, followings flits can avoid stalls for VC allocation. However, it may cause low buffer utilization due to increased VC reservation time.

4.3. 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 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 in varying 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 alleviate congestion.

Congestion detection at senders: A packet generated by a PE is stored in a buffer of the attached NI. Each packet waits until it is fragmented 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 metric of congestion detection. If the queuing delay is greater than a threshold, compression is applied to packets. 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 has no available VC, it implies that there is a non-zero queuing delay for the current or next packet. If none of three conditions is met, 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 cannot 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. 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. The threshold can be preset at the design time or be adjusted for a specific application. Additionally, a receiver keeps the compression status of each sender and sends a control packet only if it wants to change the status.

Estimating delay: The last k packets are used to estimate the queuing delay and the 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.

5. Methodology

Our evaluation methodology consists of two parts. First, we used Simics [28] full-system simulator configured for UltraSPARCIII+ multiprocessors running Solaris 9 and GEMS [29] that models directory-based cache coherence protocols to obtain real workload traces. Second, we evaluated performance and power consumption of the proposed compression schemes using an interconnection network simulator.

Table 2 shows main parameters of the 16-core design for 45nm technology as shown in Figure 1. We chose the 4GHz frequency, which respects power limitation in future CMPs guided by [3]. All the cache related delay and area parameters are determined by CACTI [30]. The design has a 16MB networked L2 cache and superscalar cores configured similarly with [31]. Tiles in a 4×4 mesh network are connected using $5mm$ long links.

Table 2. CMP System Parameters

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

The network simulator models the detailed timing and power behaviors of routers, links, encoders, and decoders. The router is configured to fit a pipeline delay for one cycle clock using the logical effort model [25]. Routers have 4-flit buffers for a VC and one flit contains 8B data. We estimate router power consumption from Orion [32]. Table 3 (a) shows the router pipeline delay and the energy consumption of each router component. The first column specifies the channel property as the number of physical channels (p), the number of VCs (v), and the 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 the length. With the optimal number (k_{opt}) and size (h_{opt}) of repeaters, the delay (T_w) for the given length (L) is determined as the following equations [33].

$$h_{opt} = \sqrt{\frac{2r_s(c_0+c_p)}{r_w c_w}}, \qquad k_{opt} = \sqrt{\frac{r_s c_w}{r_w c_0}},$$
$$T_w = 2log2\sqrt{r_s c_0 r_w c_w}\left(1 + \sqrt{\tfrac{1}{2}(1+\tfrac{c_p}{c_0})}\right)\frac{L}{h_{opt}},$$

where c_0, c_p, and r_s are the input capacitance, the output capacitance, and the output resistance of the minimum size repeater, respectively, and c_w and r_w are the unit length capacitance and resistance of the wire. Wire power consumption comes from charging and discharging both wire and repeater capacitances. Also the link power behavior is sensitive to value patterns. For accurate estimation, we consider the actual bit pattern crossing a link and the coupling effect on adjacent wires [21]. 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 [23]. Thus we can drive energy drawn in a multi-wire link (E_{link}).

$$E_{link} = 0.5V_{dd}^2\left(\alpha(\frac{k_{opt}}{h_{opt}}(c_0+c_p)+c_s)+\beta c_i\right)L, \quad (1)$$

where α and β are the transition counts for wire-substrate and inter-wire, respectively. We obtain the wire property from ITRS [5] and PTM model [34], and the repeater property from [35]. 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 3 (b) shows delay and power consumption of the global wire.

Table 3. Delay and Power Characteristics for Interconnects (4GHz and 1V used for 45nm)

Router (p, v, d)	Delay (ns)	Buffer (pJ)	Switch (pJ)	Arbiter (pJ)	Static (pJ)
6, 3, 4	0.250	11.48	34.94	0.22	9.05

(a) Router

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

(b) Link

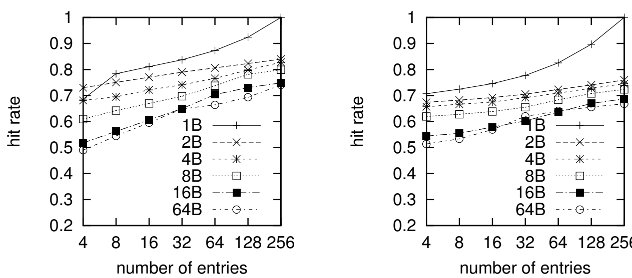
The benchmarks considered in this paper are six parallel scientific (SPEComp) and two server (SPECjbb2000, SPECweb99) workloads.

6. Experimental Evaluation

We conducted experiments to examine how communication compression affects performance and power consumption of on-chip interconnection networks. In a 64B cache block and an 8B-wide channel network, 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 cache block data starting from the most significant bit position. We apply compression only to cache block data, because address compression requires more tables and does not give a high return for packet length reduction.

6.1. Compressibility and Value Pattern

Since one cache block contains 16 4B words, words or sub-words can have the identical values. 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 such as 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.



(a) LRU Replacement (b) LFU Replacement

Figure 7. Communication Data Compressibility

Figure 7 shows the trend of average hit rates for two replacement policies. For a fixed size table, making the entry size smaller increases a hit rate more than providing more entries due to partial compression of cache blocks. For the total 128B capacity with LFU, 2B×64 (entry size × number of entries) table has 5%, 13%, and 30% higher hit rates than 4B×32, 8B×16, and 16B×8, respectively. Though it is not easy to show which replacement policy is better in our experiments, LFU exhibits less sensitive hit rates across the different number of entries of the table, partly because there is a set of frequent values in multi-threaded programs such as single-threaded programs [20]. This result shows high compressibility in cache traffic even with small tables.

Next, we examine value sharing property by analyzing values in flows that have the same 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.

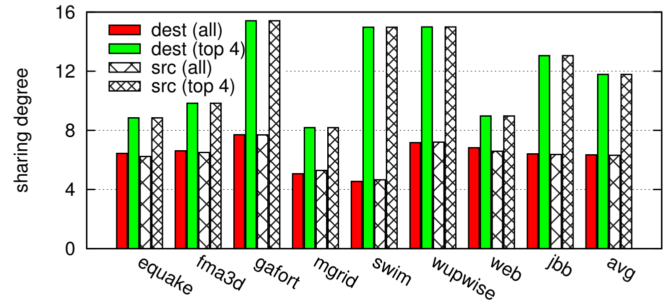


Figure 8. Value Sharing in Flows: The destination sharing degree (dest) is the average number of destinations for each value. The source sharing degree (src) is the average number of sources for each value.

Figure 8 shows sharing degrees for 2B values in each benchmark. Regarding top (frequently accessed) four values shows much higher sharing degree than taking all values. Particularly, top four values are involved with 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 2B×8 tables and the LFU policy, which is fairly small but attains a high hit rate, to evaluate our compression techniques. Furthermore, we use four tables for each tile to compress the corresponding part of 8B flit data concurrently.

6.2. Effect on Latency

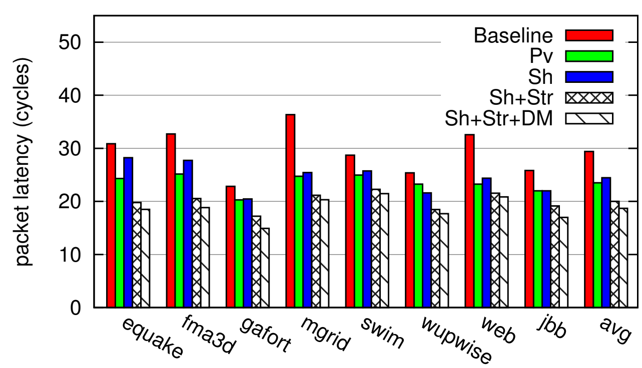


Figure 9. Latency Comparison

Figure 9 shows average packet latencies 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**. For shared table, VLB has 8 entries with 3-bit saturation counters. In most benchmarks, private table (second bar) achieves 20% latency reduction. Especially, compression improves latency dramatically by resolving high congestion in some benchmarks (*equake*, *fma3d*, *mgrid*, *web*).

Shared table (third bar) achieves almost similar improvement, 17% due to high value sharing. It penalizes latency by only 4% over private table (second bar) from a lower hit rate in the encoding table. The average hit rate for shared table decreases by 4.2%. The hit rates are listed in Table 4. Compared with the previous proposal based on runs of zero bits in a word [4], our table-based compression achieves 14% better hit rate (up to 25%). Due to a large compression gain from zeros, a special treatment such as pinning a zero value in the table would be desirable. We found that management traffic for shared table increases traffic by less than 1%.

In Figure 9, streamlined encoding (fourth bar) reduces latency for shared table (third bar) on average by 18% (4.4 cycles). Dynamic compression management (fifth bar) further reduces latency by 6% (1.3 cycles). In summary, the design using all the techniques (fifth bar) improves latency up to 44% with an average of 36% compared with the baseline.

Figure 10 shows the runtime latency behavior in *wupwise*. As expected, compression with shared table increases latency compared to the baseline in a light load, but it significantly decreases latency by activating packet compression for congested paths. In contrast, dynamic compression management applies compression on-demand and shows the finely tuned behavior. It follows low latency in the baseline architecture by shutting off compression overhead in a light load, and lowers down high latency in a high load by eliminating congestion.

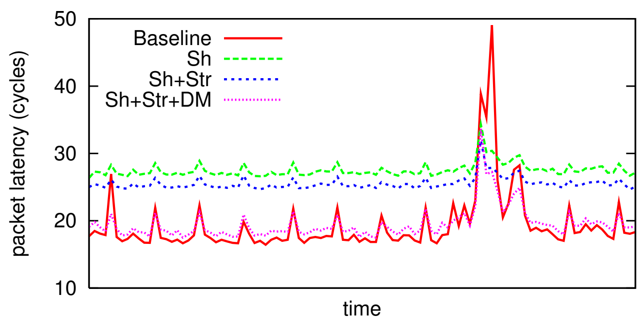


Figure 10. Behavior of Dynamic Compression

6.3. Effect on Network Power Consumption

Figure 11 shows energy reduction relative to the baseline [2]. Private table (second bar) and shared table (third bar) saves energy over the baseline (first bar) by 36% and 23% on average. As long-distance packets are more involved with compression, energy saving is more effective. For example, fma3d exhibits the hop count as 3.16 and 42% energy saving for private table, while mgrid exhibits the hop count as

2. We omit **Sh+Str** that has almost the same result as **Sh**.

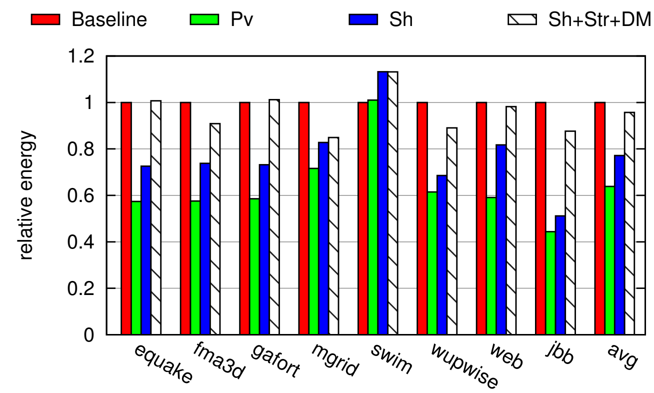


Figure 11. Energy Consumption Comparison

2.76 and 28% energy saving. Our examination in swim for increasing energy consumption exhibits that it has a huge set of different values (a low hit rate), Moreover, encoding indices from compression introduces a new pattern that is not in the original workload and does not trade off the effect of packet length reduction.

Dynamic compression management (fourth bar) diminishes energy saving in shared table (third bar), resulting in only 4% energy saving over the baseline. Also additional control packets for congestion detection at receivers are required.

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 11% (up to 24%) while bit pattern analysis for intra-wire switching shows an average of 2% activity factor (up to 7%).

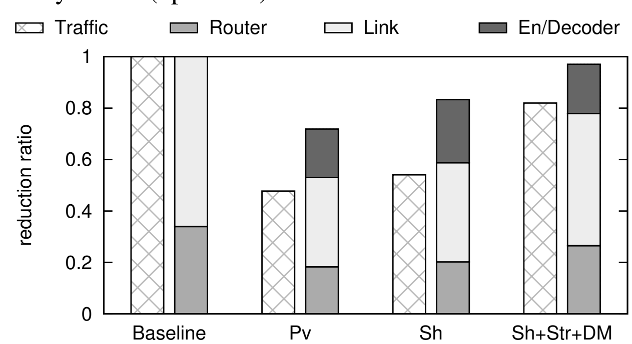


Figure 12. Traffic and Energy Relationship

Figure 12 shows a relationship between traffic (left bar) and energy (right bar). It further breaks down energy consumption into routers, links, and encoding/decoding tables. We observe that energy reduction is less than traffic reduction, because more repeated values in traffic implies less switching activities. Routers consume 34% of the total network energy in baseline configurations. 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). For shared table, encoders

scheme	equake	fma3d	gafort	mgrid	swim	wupwise	web	jbb	avg.
private	0.877	0.865	0.893	0.706	0.345	0.774	0.856	0.981	0.787
shared	0.804	0.801	0.871	0.694	0.337	0.762	0.788	0.973	0.754

Table 4. Encoding Table Hit Rates for Private and Shared Tables

and decoders consume 25% of the total energy.

7. Conclusion

The current design trend towards many-core architectures indicates that on-chip interconnect is becoming a performance bottleneck. We introduced a table-based data compression framework to maximize the bandwidth of the existing on-chip network. Compression enables the network to perform low latency packet delivery even at the saturation point. Another benefit is energy saving by reducing router operations and link traversals.

In this work, we presented optimization techniques to overcome the huge table cost and the long compression latency. First, we proposed the shared table scheme, which stores identical values into a single entry from different sources or destinations, and thus provides area-scalable implementation. We also presented an efficient table management protocol for consistency. We proposed the performance improvement schemes; Streamlined encoding reduces the encoding latency by overlapping encoding with flit injection. Dynamic compression management encodes packets on-demand in varying workloads to maximize performance. Our simulation results show that compression using shared tables improves the packet latency by 31% and saves energy by 36%. Moreover, dynamic compression management further achieves additional 5% latency improvement.

Due to limitations of the current simulation environment, we only showed the network performance. We are currently integrating our design in a full-system simulator to evaluate overall system performance.

References

[1] R. Kumar, V. V. Zyuban, and D. M. Tullsen, "Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads and Scaling," in *Proceedings of ISCA*, pp. 408–419, 2005.

[2] 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.

[3] 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.

[4] 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*, 2008.

[5] "International Technology Roadmap for Semiconductors," 2005.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

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

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

[18] M. Thuresson, L. Spracklen, and P. Stenström, "Memory-Link Compression Schemes: A Value Locality Perspective," *IEEE Trans. Computers*, vol. 57, no. 7, pp. 916–927, 2008.

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

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

[21] 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.

[22] 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.

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

[24] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

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

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

[27] 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.

[28] 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.

[29] 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*, 2005.

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

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

[32] 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.

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

[34] "Predictive Technology Model." `http://www.eas.asu.edu/~ptm`.

[35] 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.