

Data Compression Through Data Encoding In Network On Chip

V. AnandKumar¹

*¹M. Tech, VLSI Design, School of Computing,
SASTRA University, India.
Email: anandkumarv1992@gmail.com*

R. Saravanan²

*²Assitant Professor, School of Computing,
SASTRA University, India.
Email: saravanan_r@ict.sastra.edu*

Abstract

In Network on Chip (NoC), the number of processing element increases with technology which leads to heavy traffic in links between the cache and the memory accesses. Thus, this paper presents a delta encoding technique to compress the data packets and the objective of the technique to reduce traffic in the networks. The data compression is done before injecting the packets into the network and decompression is done before ejection into the Network Interface (NI). The compression and decompression is done in the Network Interface to curtail the network traffic. Moreover, the proposed compression/decompression is not required any modification in the cache memory design. The results show that the total area and power overhead is reduced considerably.

Keywords: NoC, NI, Delta encoding, Data compression

Introduction

Network on Chip (NoC) [1] has been proposed to overcome the limitations of System on Chip (SoC) like bandwidth, scalability and throughput. Instead of bus based communication architecture in SoC, a new pattern of interconnection between the processing elements has been proposed for on chip communication which provides scalability and bandwidth. Though NoC having limitations like high network power dissipation and long end-to-end latency. Many previous works [2]-[4] include optimization on network topologies, router microarchitecture, routing algorithms, etc. However, these methods may lead to degradation of performance, increase hardware complexity or power.

Instead of going for traditional optimization methods, an orthogonal and complementary approach is data compression which is also known as source coding. Data packets are compressed before injection into the network and decompressed before ejection into the Network Interface (NI). Data compression improves the performance of the system and reduces the power dissipation.

In Network on Chip, point to point links makes the interconnection between routers and between router and network interface. A router consists of buffer, routing logic, arbiter and crossbar switch. The interconnection of routers by links describes the topology of the NoC. Many topologies for NoC have been proposed like mesh, torus, fat tree etc. The most common topology is mesh topology. In this topology every router interconnected by links of equal length. The address of every router is its local XY coordinates. A router has 5 inputs and 5 outputs out of these 4 inputs and 4 outputs for inter router communication and another input and output for making connection with the processing element.

Packet switching techniques in NoC are store and forward, cut through and wormhole switching. The Conventional packet switching approach is Store and Forward (SAF), here the packets are stored in the router and then forwarded to the next router depends on its routing algorithm. Buffer is used to store the packets in the router. SAF helps for flow control in the networks. In this technique, each router waits until a packet is completely received or stored and then the packet is forwarded to the next router. Two resources are needed one is a packet sized buffer in the router to store the packets and other is the exclusive use of the outgoing channel. This technique is having latency and a packet sized buffer space is required at each node. The advantage of SAF is while a router waiting to acquire resources, no channels are being held idle. Pipelining the transfer of SAF switching is the cut through switching here the router does not wait for the whole packets to get arrived at the router, when the header arrives at the router it immediately do routing computation and get routed. When the packet struck because of any congestion in the network this results in holding resources in the router. Wormhole routing is same as cut through switching, here it checks for flit size buffer availability and the packet is forwarded to the next router. Router works based on its routing algorithm.

XY routing algorithm is the common routing algorithm for Mesh topology. XY routing is dimension order routing algorithm and minimal turn algorithm. The packet header contains the routing information like source address, destination address, packet size and other information about the packets. In XY routing algorithm the packets are routed first in X direction and then in Y direction. The advantages of XY routing algorithm is free from deadlock and live-lock.

Related Works

Cache compression increases the capacity of cache memory by stuffing in more data blocks in a given space of memory architecture. Zero-Content Augmented cache design [5] proposed by Dusser et al. that cache the null blocks address tag and valid bit in the same address tag. A separate small size memory is allotted to store only the tag address of null values thus reduces the memory traffic and miss rate which

increases the performance of the system with low hardware complexity. Zhang et al [6] observed the frequent value locality, a few values which come out in memory locations very frequently. A value centric frequent value cache approach introduced to cache data locations of frequent values in memory location. The frequent values are compressed and store it in allotted memory locations within the same cache memory. In [7], the two frequent value cache lines are compressed and store it in the same cache line. Alameldeed and Wood [8] discovered some patterns that appear frequently in programs that all words are same in a 4byte word. They adopted adaptive cache compression for L2 8-way set associative cache memory which improves the performance. Das et al. [9] examined cache compression and compression in the network interface, here the data messages are compressed depends on zero bits in a word. Jin et al. [10] proposed table based compression/decompression process for frequent values in the cache memory. Zhou et al. [11] exploits table based frequent value compression to optimize the performance and power of NoC. In this work table based compression/decompression is done with a table management protocol updates the cache memory from destination to source.

Pekhimenko et al. [12] discovered a novel compression algorithm called Base-Delta-Immediate (BDI) compression, which compress and stores the cache lines as a set multiple differences (Δ) to the base values. Though, these techniques of cache compression require necessary modification in the cache memory storage design to support the compression and variable size cache lines. Nowadays modern SoC contains non-flexible hard IP blocks, so the previous methods are not applicable. But on the other hand, simple lightweight compression/decompression modules can be integrated in the network interface (NI). In the proposed approach, cache storage modification is not required. The proposed work is based on the observation that the data stored in the cache memory is in the form of relative differences or many cache lines have low dynamic range of data stored in it.

Proposed Compression and Decompression System

A. Multi-core NoC Architecture

In this work, the delta encoding is applied to cache lines to achieve compression in multi-core NoC by sending data packets as difference (Δ) between the sequential values instead of complete values. Multi-core NoC system in this paper is analysed a 4x4 2D mesh network interconnected by routers as shown in Figure 1. In this design, each node consists of a CPU core with a private L1 instruction and data cache, and a separate shared bank of last level cache (L2). In this work [13] the Static Non-Uniform Cache Access (SNUCA) architectures is implemented for L2 cache. In SNUCA the mapping of data blocks in cache structure is unique and cannot allow moving dynamically. The processing elements are attached with the router through the Network Interface (NI). There are 4 memory controllers (MC) on the four corners of NoC. Every MC attached with the main memory and controls a single memory channel by many Dual In-line Memory Module (DIMM).

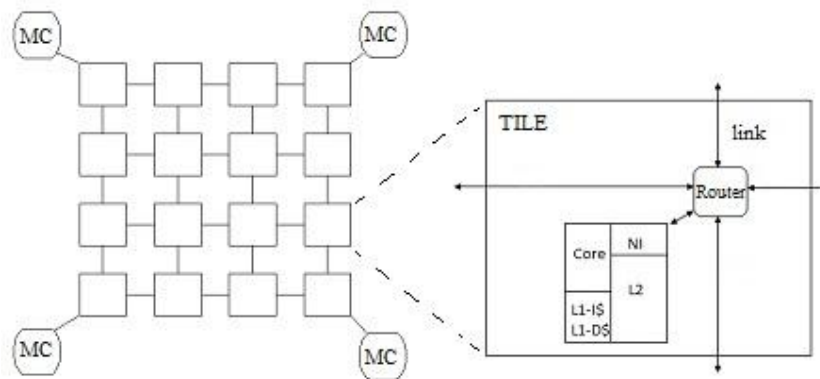


Figure 1: Multi-core 2D 4x4 mesh NoC with 4 Memory Controllers

B. Packet Switching and Memory Access

During a memory request from the core, firstly the private L1 cache is checked to observe the availability of the data. If the data is available the core accesses the data from its private L1 cache. If the data is not available then it forwards the request to the bank L2 through path 1. If again an L2 miss occurred the request is further forwarded to the memory controller through path 2 and off chip memory access is encountered through path 3. The response message from off chip memory reaches the bank L2 via the path 4 and further reaches the respective bank through path 5. L1 cache hit does not create any network traffic, the paths from 2 to 4 used for request/response to off chip memory. Figure 2 shows the off-chip memory for multi-core NoC. Network Interface taken care of data packetization/de-packetization operation. Network Interface makes the data messages into flits that a network can handle. Network latency is an important factor in memory accesses to meet the Quality-of-Service (QoS) requirement.

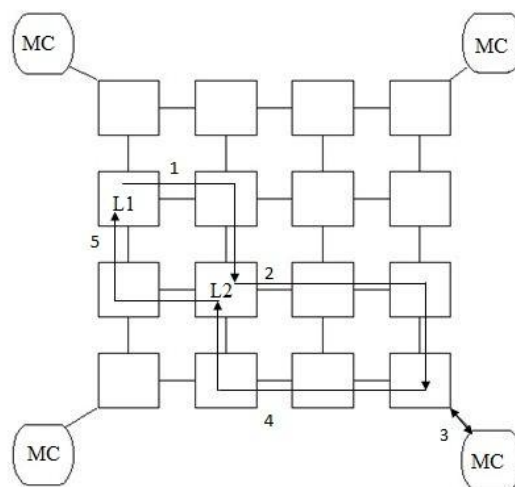


Figure 2: Memory Access in multi-core NoC

C. Packet Format

In the processor cache, when there is a miss in load and store instruction, these misses are transformed into read request and write request packets and injected into the network through the Network Interface. Figure 3 shows the request/response packets for NoC. The read request packet contains only the flit size packet that contains the address information of the data. The read reply and write request message contains header and body flits. The write request also contains the information of where the data to get stored in the cache memory.

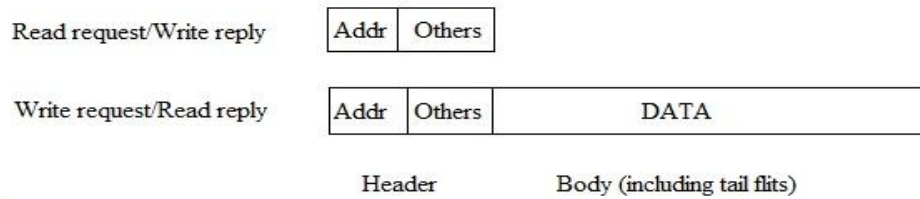


Figure 3: Request and response packets for memory accesses

D. Compression Module

The input to the compression module is a data packet with body flit. The body flits are subjected to compression. The first flit of the body is taken as a base and delta encoding is done for the remaining body flits with respect to the base. Taking the first flit as the base and then finding the differences (Δ) of the remaining body flits. For N number of body flits, N-1 subtractors are required for the compression module. In this work, the body flits is of 16 byte with 4 bytes for each flit. The input and output is of 32 bits wide. A compression module diagram is shown in the figure 4. The difference values are checked for sign extension by observing the most significant bits (MSB). If any one of the Δ s is sign extended then the packet can be compressed. The sign extensions are stored and transmitted in the header which is used for enabling operation in the decompression module. All the compression modules are getting the 16 byte cache line data packet and compression operation takes place. After the compression the module with the less data bytes are injected into the network. Depends on the compression encoding values are written in the header for the reproduction of the original packet during decompression operation.

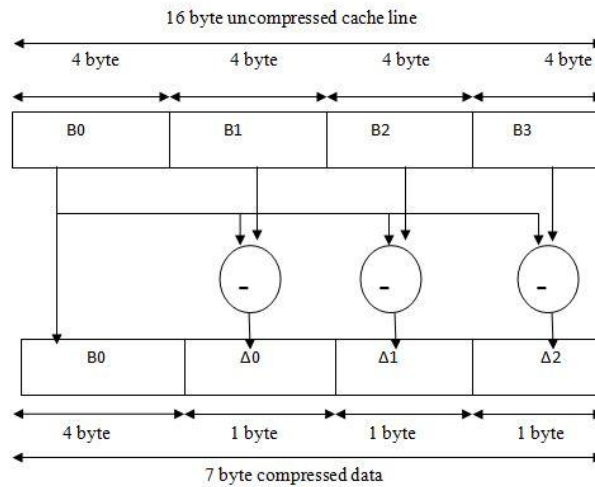


Figure 4: A single proposed compression Module

E. Decompression Module

Decompression module consists of adder/subtractor circuit for reproducing the original data packets. A single decompression module is shown in the figure 5. If sign extension is 1, subtractor is enabled; the original data packets are reproduced by subtracting the deltas from the base value. When sign extension is 0, adder is enabled; the deltas are added with the base flit for the reproduction of original data packet. When there is no compression value of zero is placed in the base and the output is the original uncompressed data. In the decompression module the subtractor input and output width requires the size of delta. The base value is given directly to the output.

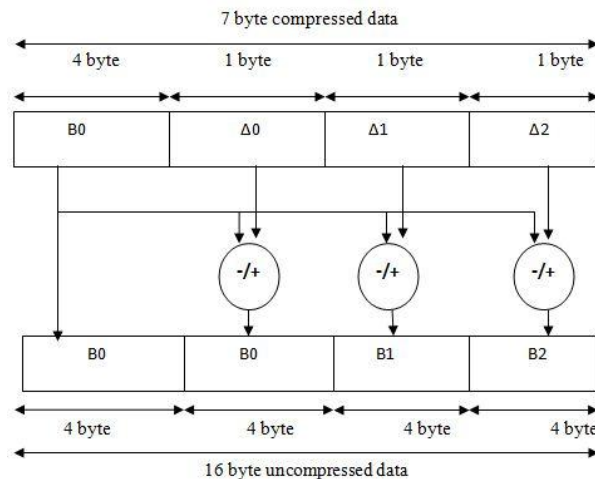


Figure 5: A single proposed decompression Module

In order to support different (Δ) sizes, various compression modules are shown in Table I and organized in parallel as shown in Figure 6. The compression modules

simultaneously compress the data packets from the uncompressed cache line and the priority multiplexer selects the body flits with smallest number of body flits. The Table I show the priority given for the body flits depends on the value in it. The priority in this table is for choosing the compression data packets. The lowest priority is given for no compression data. If the all body flits is zero then priority for this type of body packet is one, instead of send body flits the encoding data is stored in the header and injected into the network. For repeated body flit value the encoding data is stored in the header and packet consist of header and first body flit is injected into the network. For body flit with one byte variation comes under B4 Δ 1 and base of 4 byte and 2 byte comes under B4 Δ 2.

The respective encoding data is stored in the header before injection in to network which is helpful in reproduction of the original data during decompression.

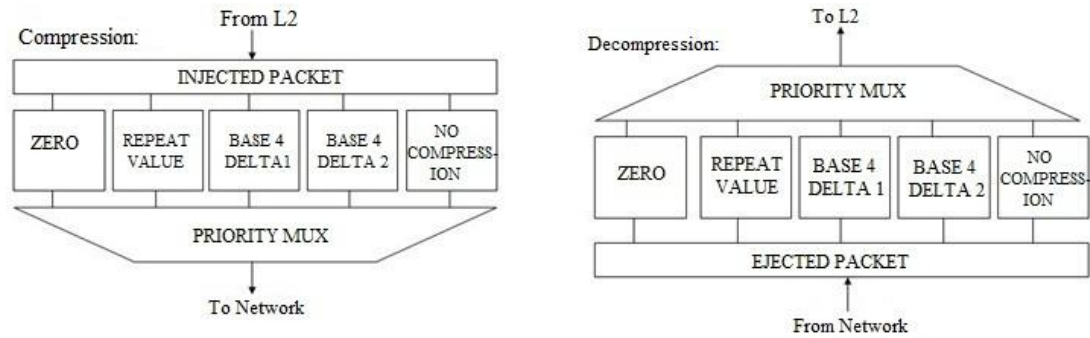


Figure 6: Proposed architecture with multiple compression and decompression modules

Table 1: Encodings For Different Types of Compression

Name	Base (bytes)	Δ (bytes)	Packet Size in bytes (flits)	Priority	Encoding
Zero	0	0	4(1)	1	000
Repeat value	4	0	8(2)	2	001
B4 Δ 1	4	1	11(3)	3	010
B4 Δ 2	4	2	14(4)	4	011
No Compression	4	0	20(5)	5	111

Experimental Setup

For experiments, a 4x4 2D Mesh NoC with dimension order routing algorithm with store and forward flow control is taken for consideration. The network flit size is of 4 bytes long and the router use the conventional five-stage pipelined architecture. Each router is having two Virtual Channels (VC) per port and each VC is of 4 flit depth. The full system configuration is shown in Table II.

Table 2: System Configurations

PARAMETERS	DETAILS
Core count	16
L1 I & D cache	2-way set associative, 16KBx16, 1cycle
L2 Cache	8-way set associative, 1MBx16, 5 cycle per bank
Bank count	16
Memory controllers	4 on the corners
Network Topology	4x4 2D mesh
Routing	Dimension order routing
Router pipeline	5 stage pipeline
Buffer size	2 Virtual Channel (VC) per port, 4-flit depth per VC
Packet length	5 flits
Flit length	4 bytes

Results and Analysis

Multi-core NoC with 16-cores is modelled for evaluation of the proposed work. Each core has a private 16KB L1 cache memory which is a split cache for instruction and data values (8KB for instruction and data cache separately). In addition, each core has a shared bank of L2 cache with each 1MB size following SNUCA. The CACTI tool produces perfect cache memory designs like power consumption, area, access time, dynamic power, leakage power, etc based on the input parameters like cache memory size, technology node, banks, line size and so on. The CACTI tool [14] is used to analyze the cache memory for different cache sizes from 16 to 1024 KB in 16 banks with line size of 16bytes, 2-way set associative with a technology node of 45nm are tabulated in Table III. .

TABLE 3: Cache Sizes Comparison of Different L1 2-Way Set Associative Cache

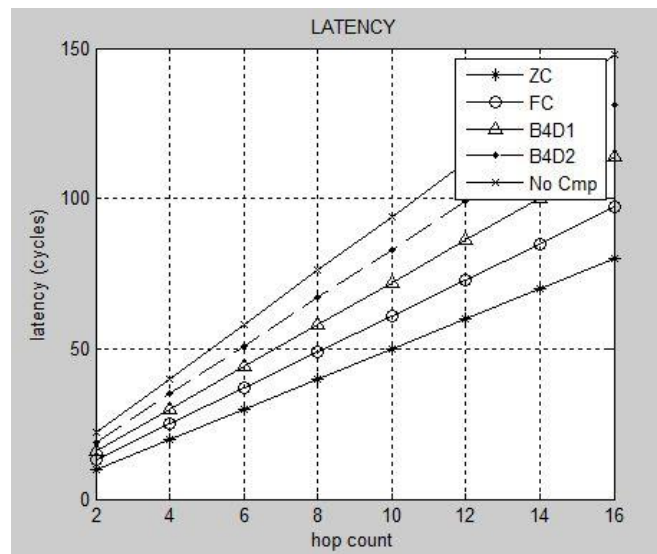
Cache size (KB)	Access time (ns)	Random cycle time(ns)	Total area (mm ²)	Total read dynamic power (W)	Total standby leakage power (W)
16	0.795	0.1495	2.2172	0.2636	0.004806
32	0.8203	0.1576	2.3516	0.2631	0.005534
64	0.8284	0.2053	2.4693	0.188	0.006083
128	1.0151	0.191	3.1325	0.306	0.011
256	1.2328	0.2372	4.7063	0.3318	0.018041
512	1.4653	0.2934	6.8687	0.3497	0.031972
1024	1.8888	0.3744	10.406	0.4072	0.060508

For the L2 cache the size is varied from 1MB to 16MB over 16 banks with a line size of 16 byte, 8-way set associative with 45nm technology is shown in Table IV.

Table 4: Cache Sizes Comparison of Different L2 8-Way Set Associative Cache

Cache size (MB)	Access time (ns)	Random cycle time(ns)	Total area (mm ²)	Total dynamic read power (W)	Total standby leakage power (W)
1	1.9114	0.4056	11.3692	0.3919	0.05633
2	2.2381	0.417	16.2781	0.5364	0.1052
4	3.1351	0.6812	27.9681	0.4599	0.20349
8	4.1542	0.6921	48.11	0.773	0.3769
16	5.7261	1.4452	88.01	0.4618	0.7177

The latency of the data packets in the network before and after compression is analysed for the proposed work. The latency in cycles for different compression modules of the proposed system with respect to the number of router hops in the network is shown in the Figure 7. ZC-Zero compression, FC-Frequent compression, B4D1-Base 4 bytes of delta 1bytes, B4D2-Base 4bytes of delta 2bytes and No Compression.

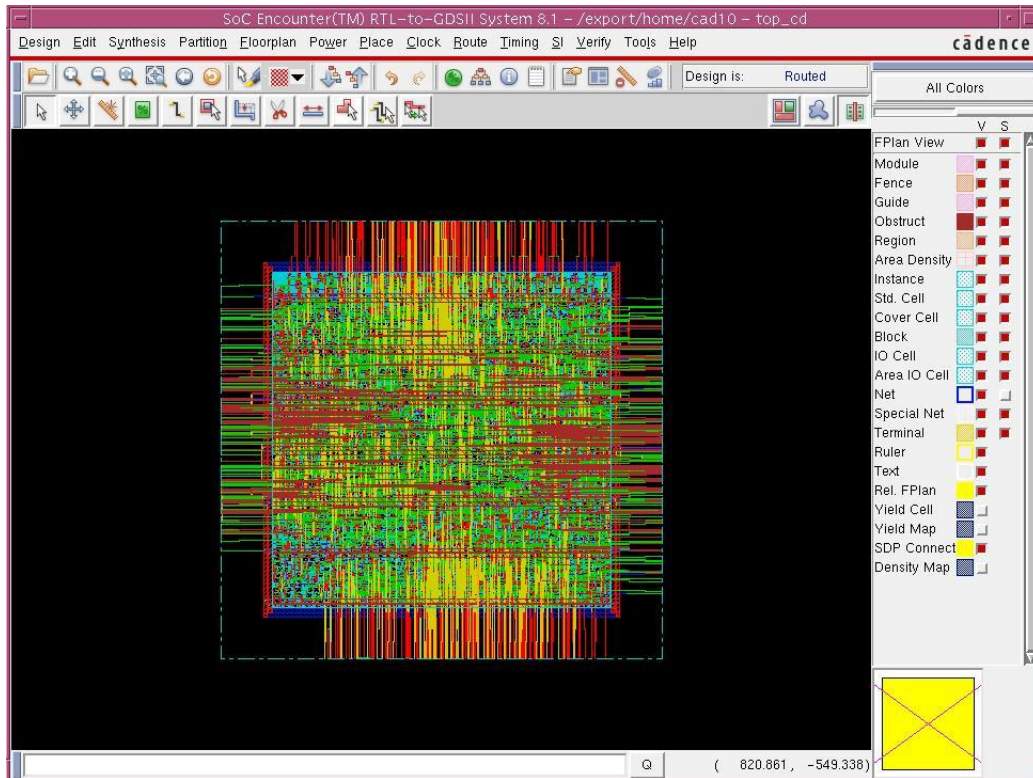
**Figure 7:** Latency In Cycles For Different Modules of Proposed Compression

We implemented our compression and decompression modules in behavioral Verilog HDL and functionally verified and simulated in Mentor Graphics Modelsim. The design is checked for all the patterns from the cache memory using test bench. The design is further implemented in Cadence RTL Compiler for a technology library of 45nm. The observed result for the compression and decompression modules is shown in Table V.

Table 5: Comparison of Power and Area overhead for different schemes

Area(mm ²)	Power(mW)			Power(mW)		
	Encoder	Decoder	Total	Encoder	Decoder	Total
ZeroCmpr [9]	NA	NA	0.183	NA	NA	273
FreqCmpr_p [10]	0.00526	0.00523	0.17	NA	NA	1626.1
FreqCmpr_s [10]	NA	NA	0.023	NA	NA	220
FreqCmpr_S [11]	0.00443	0.00432	0.14	4.26	3.95	131.36
Proposed	0.00401	0.00196	0.007	4.07	1.35	19.505

Table V shows the comparison of the proposed work with the previous studies under 45nm technology. The design is further extended in the Cadence SoC encounter and the final layout for the compression and decompression is shown in the Figure 8.

**Figure 8:** SoC Encounter Cadence Layout of Compression and Decompression Modules

Conclusion

In this work, a new and simple compression/decompression which conducts data encoding/decoding before injection/ejection of packets in to networks. The proposed encoder/decoder module offers minimal area and power overhead as from the Table V. The proposed compression achieves the average latency improvement by 17% in

the network. The proposed work is compared with the previous related works and the experimental results show that the proposed work offers a minimal overhead in both area and power.

References

- [1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *DAC*, 2001, pp. 684–689.
- [2] J. Kim *et al.*, "A novel dimensionally-decomposed router for on chip communication in 3D architectures," *CM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 138-149, 2007.
- [3] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Express cube topologies for on-chip interconnects," in *HPCA*, 2009, pp. 163–174.
- [4] S. Ma, N. E. Jerger, and Z. Wang, "Whole packet forwarding: Efficient design of fully adaptive routing algorithms for networks-on-chip," in *HPCA*, 2012, pp. 1–12.
- [5] J. Dusser, T. Piquet, and A. Sez nec, "Zero-content augmented caches," in *ICS*, 2009, pp. 46–55.
- [6] Y. Zhang, J. Yang, and R. Gupta, "Frequent value locality and value centric data cache design," in *ACM SIGOPS Operating Systems Review*, vol. 34, no. 5, 2000, pp. 150–159.
- [7] J. Yang, Y. Zhang, and R. Gupta, "Frequent value compression in data caches," in *MICRO*, 2000, pp. 258–265.
- [8] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high-performance processors," in *ISCA*, 2004, pp. 212–223.
- [9] R. Das *et al.*, "Performance and power optimization through data compression in network-on-chip architectures," in *HPCA*, 2008, pp. 215–225.
- [10] Y. Jin, K. H. Yum, and E. J. Kim, "Adaptive data compression for high-performance low-power on-chip networks," in *MICRO*, 2008, pp.354–363.
- [11] P. Zhou *et al.*, "Frequent value compression in packet-based NoC architectures," in *ASPDAC*, 2009, pp. 13–18.
- [12] G. Pekhimenko *et al.*, "Base-delta-immediate compression: practical data compression for on-chip caches," in *PACT*, 2012, pp. 377–388.
- [13] J. Huh *et al.*, "A NUCA substrate for flexible CMP cache sharing," in *ICS*, 2005, pp. 31-40.
- [14] CACTI tool v5.3, <http://quid.hpl.hp.com:9081/cacti/>

