FlitZip: Effective Packet Compression for NoC in MultiProcessor System-on-Chip

Dipika Deb, Rohith M.K., and John Jose

Abstract—Applications running on Network on Chip (NoC) based multicore systems demand increased on-chip network bandwidth that can cater to the need for intensive communication among the cores and caches. Due to strict area and power budget, the bandwidth offered by NoC is very limited. Data-intensive and communication-centric applications on encountering a cache miss lead to a considerable burden on the underlying network for transferring blocks from multiple cache hierarchies to the requesting core as packets. This increases the packet transmission latency, thereby slowing down the system performance. Also, NoC being the highest component of power consumption after the cores, an increase in packets increases the dynamic power consumption of NoC. The article proposes *FlitZip* that addresses the problem by reducing on-chip traffic through compressing network packets. Hence, the compressed packet requires less bandwidth during its transfer, reducing the network's power consumption. Experimental analysis shows that *FlitZip* achieves a better compression ratio of 52 percent, reduces packet latency and bandwidth utilization by 19.28 and 27 percent, respectively. It also reduces the area and power consumption of the de/compression units by 53.33 and 62.3 percent, respectively, compared to the state-of-the-art packet compression technique, NoΔ.

Index Terms—Network on chip, packet, delta compression, multicore processor, router

1 Introduction

With the increase in transistor count, MultiProcessor System-on-Chip (MPSoC) provides higher performance than a uniprocessor system [1]. In MPSoCs, Network on Chip (NoC) provides a scalable interconnect as compared to the traditional bus-based interconnect [2]. However, network bandwidth and power consumed by NoC [3], [4], [5], [6] have become a major concern in designing MPSoCs as NoC accounts for around 28 percent of tile-power in Intel Teraflop chip [4] and 36 percent in MIT RAW chip [3]. Hence along with the application's performance, the power consumed by NoC to route request (cache miss) and reply (cache block) packets can no longer be left as a secondary metric for designing a high-performance system. Packet compression [7], [8], [9], [10], [11], [12] provides an alternative to reduce network power by exploiting data redundancy within NoC packets, thereby shrinking its size. Reduction in packet size reduces a) network traffic, b) wastage of link bandwidth, and c) dynamic power consumption in NoC.

Packet compression can be lossless [7], [8], [9], [9], [10], [11], [12] or lossy [13], [14], [15], [16]. Lossy technique aims to attain a higher compression ratio without the need for exactness. Thus, packets before compression and packets after decompression may not be the same. Such techniques are generally faster than lossless techniques and are used in error-tolerant applications like image processing, voice recognition, etc. On the other hand, the lossless technique aims for exactness. Such techniques are used for applications where data loss may change the application execution path. Thus, packets received after decompression are the same as that of the original packet.

Though packet compression is beneficial for enhanced NoC performance, the overhead and latency of compressor and decompressor units add to the average memory access time. This demands for a simple and lightweight compression technique like delta compression [7], [17], [18], [19], [20]. Delta compression is used due to its simplicity, less hardware overhead, power efficiency, and lower de/compression latency. It uses the fact that the relative deviation between data values in a cache block varies a little. Hence, a part of the cache block (X bytes) can be represented as a common base of P bytes and rest of the block is represented as an array of differences: $\{\Delta_1, \Delta_2, \Delta_3, ..., \Delta_n\}$ where $n = X/P$, from the base. It has been applied on caches [18], [21], [22], [23], [24], NoC packets [7], [11] and DRAM [19], [20], [25].

Compression for cache and DRAM increases the space utility, while packet compression in NoC reduces bandwidth utilization, network congestion, and dynamic power consumption. However, cache and DRAM block compression come with extra overhead for space management during block writes and replacements. Also, a compressed block must be decompressed for every access, thereby adding extra latency for each access. In NoC, packet compression is applied on end-to-end, i.e., at Network Interface (NI) only. Caches and DRAM are unaware of any underlying packet compression and functions as a conventional system.

- *Dipika Deb and John Jose are with the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati, Assam 781039, India. E-mail: {d.dipika, john.jose}@iitg.ac.in.*
- *Rohith M.K. is with the Department of Electronics and Communication Engineering, R. V. College of Engineering, Bangalore 560059, India. E-mail: rohithmk97@gmail.com.*

Manuscript received 28 May 2020; revised 31 May 2021; accepted 7 June 2021. Date of publication 17 June 2021; date of current version 2 July 2021.
(Corresponding author: Dipika Deb.)
Recommended for acceptance by R. Ge.
Digital Object Identifier no. 10.1109/TPDS.2021.3090315

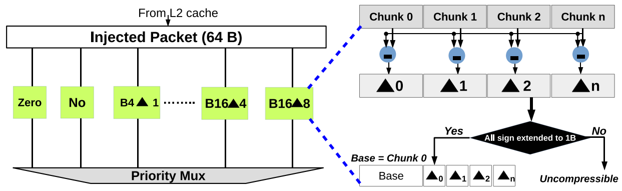


Fig. 1. Compressor unit in existing delta compression in NoC.

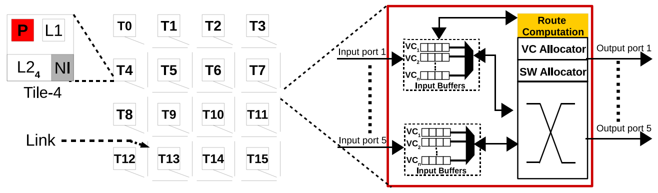


Fig. 2. (a) A 16-core NoC based MPSoC organized as 4x4 2D Mesh. (b) Router architecture. P: Processor/Core; NI: Network Interface.

Very few works have explored NoC packet compression [8], [9], [10] out of which only NoΔ [7] and DISCO [11], [12] uses delta compression on NoC packets.

Fig. 1 shows eleven combinations of de/compressor modules used in existing delta compression in NoC [7], [11], [12]: B16Δ8, B16Δ4, B16Δ2, B16Δ1, B8Δ4, . . . , B4Δ1. The term BxΔy means that the packet is compressed with a base of size x bytes and Δ_i of y bytes. Since a 64 byte packet can be represented as four *16 byte* chunks or eight *8 byte* chunks or sixteen *4 byte* chunks, the base in these cases are of size 16 bytes (B16), 8 bytes (B8) and 4 bytes (B4), respectively. For a particular base say B8, the size of Δ can be represented using 1 byte (B8Δ1), 2 bytes (B8Δ2) or 4 bytes (B8Δ4).

We study the existing delta packet compression techniques [7], [11] in general and uncover a few of its limitations here. 1) The size of base $(4 \leq B \leq 16)$ and Δ $(1 \leq \Delta \leq 8)$ are always represented in bytes. Hence, existing techniques can never compress a packet with a smaller base (<4 bytes) or Δ (<1 byte). Consider a 64 byte packet with data consisting of repeated 0xFF values. The compressed packet would contain a base and Δ_i is 0, which is minimally represented in a byte. But 0 can be represented in less than a byte, which is not possible in the existing techniques. 2) The de/compressor units are bulky and are not area and power efficient. 3) The metadata of a compressed packet is represented using an encoding table containing all combinations of (B, Δ), encoding bits and priority bits. The size of the table is 250 bytes.[1] 4) The base of the compressed packet is appended in the body flits. Hence if $B16$ is used for compression, the minimum packet size is 16 bytes plus Δs. 5) Also, only the encoding scheme of a compressed packet is stored in the head flit. But a head flit has around 50 percent unused bits as shown in Fig. 3. Thus, the remaining bits are underutilized.

The paper proposes a novel and lightweight lossless packet compression technique, FlitZip, that compresses packets at flit-level granularity. It exploits the fact that packets across a wider range may not show any data pattern, thereby making it incompressible. However, if the same packet is divided into smaller parts and the data pattern is individually observed, each part may be compressible at different compression ratios. Thus, the same packet may become compressible part-wise. FlitZip views each flit as a set of one byte chunks where each chunk are compressed in a compact form as a difference (di) from an optimal base. The base and encoding of a compressed packet are stored in the head flit's unused portion, which helps achieve a higher compression ratio. Unlike existing delta compression techniques [7], [11], [12], FlitZip is made power and area efficient by using a single de/compressor of fixed base size of one byte, thereby avoiding multiple de/compressor modules.

The key contributions of the paper are as follows.

(1) We propose FlitZip that identifies data patterns within a flit and compresses each packet on a per-flit basis, thereby achieving a higher compression ratio.
(2) By using a constant base of size 1 byte (B1), FlitZip significantly reduces the size of the subtractor units and avoids multiple de/compression modules for different base sizes. The hardware optimization provides significant savings in area and power.
(3) FlitZip utilizes the unused portion of the head flit to store the compressed packet's metadata.
(4) We perform extensive experiments with 397 workloads from multithreaded (PARSEC 3.0 [26]) and multiprogrammed (SPEC CPU2006 [27]) benchmarks to validate our claims.

Experimentally we have found that in 8x8 NoC, FlitZip achieves an average packet compression ratio of 0.52, reduces packet transmission latency by 19.28 percent, and increases weighted speedup by 12 percent as compared to the baseline (without any compression). The rest of the paper is organized as follows. The literature survey related to NoC packet compression is discussed in the next section. The motivation behind this work is presented in Section 3. Section 4 presents the proposed technique, FlitZip followed by experimental analysis in Section 5. Section 6 presents the hardware aspect of FlitZip and we finally conclude the paper in Section 7.

2 Background and Related Work

Architecture Used: Fig. 2a shows a 16-core NoC based MPSoC where the cores are organized as tiles, and each tile consists of an OoO core, a private L1 cache, and a part of shared (inclusive) L2 (L2 is used as the Last Level Cache) cache called as a bank. A cache block is statically mapped to a bank called the block's home-bank based on its bank-index bits [28], [29], [30], [31], [32]. Each tile is connected to NoC which is arranged as 2D mesh topology using Network Interface (NI). NI consists of an *inject* and *eject* queue that is used to transfer cache blocks to and from the processor. A miss in L1 cache generates a request packet destined to the home-bank of the block. It travels through the underlying NoC and the home-bank forwards cache block (reply packet) to the requesting core.

A NoC consists of routers and links, and the request, reply packets constitute NoC traffic. A request packet is represented using one head flit, and reply packets are further divided into multiple flits as head, body, and tail; with each

1. Encoding table stores base (max 16 bytes), Δ (max 8 bytes), encoding (4bits) and priority (4bits) at each tile.

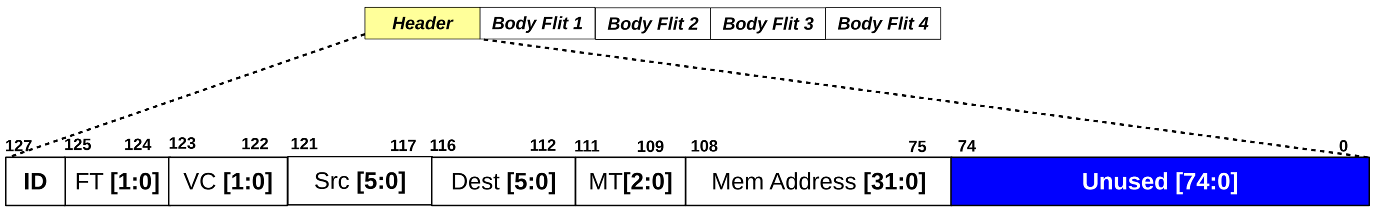


Fig. 3. Content of Head flit for 8x8 NoC framework with link bandwidth of 128 bits, packet size of 64B and main memory size of 4GB.

flit equivalent to the link bandwidth. The packet's control information is contained in the head flit, and the cache block contents in the body flit and tail flit. Fig. 2b shows the router architecture that consists of input port buffers known as Virtual Channels (VC) [33], [34], Route Computation (RC) unit, VC allocator (VA), and Switch allocator (SA). RC determines the output port that a packet takes in the current router to reach its destination. VA determines the VC number in the downstream router. SA resolves port contention and assigns output port through which flits travel out of a router and link carries flit from one router to another.

Related Work: J. Zhan *et al.* [7] proposed NoΔ that uses delta compression and has multiple de/compression modules that execute all combinations of (Base, Δ) to achieve a better packet compression ratio. USBR[9] primarily aims at datasets whose variance is low such that the significant bit changes less frequently and hence becomes a use case to reduce network packets. Das *et al.* [8] proposed ZeroCompr that reduces packet size by encoding consecutive zeroes in the head flit. Frequent Pattern Compression (FPC) [10] uses an encoding table to compress an incoming packet against the patterns stored in the table. The table contains eight frequently occurring patterns and it is shared among all the NIs. Wang *et al.* proposed DISCO router [11], [12], where an incoming packet is selectively compressed at the routers using delta compression [7], [18]. The selection criteria are based on the packet idle time stored in the internal buffers of a router. The selected packets then undergo compression using delta compression (BDI or NoΔ) or FPC.

Boyapati *et al.* introduced a lossy compression, Approx-NoC [13] that approximates frequently occurring data values to reduce the network load for error-tolerant applications. Chen *et al.* [14] also proposed an approximate communication framework that reduces packet size by approximating data values within network packets. Raparti *et al.* proposed DAPPER [35] that uses approximate-computing for GPUGPU architectures. It trades-off computation accuracy for energy savings and uses an overlay circuit prepared dynamically between Memory Controllers (MCs) for a time window. Chen *et al.* proposed DEC-NoC [16] that considers that the NoC is prone to error. Hence, packets are retransmitted to mitigate the errors. To reduce power consumption, the author reduces the amount of error correction and checking during packet transmission. However, FlitZip considers that the NoC is error-free and requires no retransmission of packets.

3 Motivation

We analyse delta compression in NoC [7], [11], [12] and have three motivational observations for proposing FlitZip.

1. *Unused Bits in the Head Flit:* Since each flit is equal to the link bandwidth, the head flit contains control information required by the home-bank to fetch data blocks during a cache miss. The fields are packet number (ID), source tile (Src) that generates the cache miss, destination tile (Dest): the home-bank of the block, flit type (FT): head/body/tail, VC number where the incoming flit is stored, Message Type (MT): REQ/REP/coherence packets and missed block address (MEM-ADDR). Hence, few bits from the head flit remain unused. Fig. 3 shows the structure of a head flit in a 64 core NoC framework with 128-bits link bandwidth. The system uses 4GB main memory with a packet size of 64B.

Out of the total 128 bits in the head flit, 75 bits from the LSB is unused (unused bits in the head flit changes if any of the fields increases or decreases). NoΔ uses only 4 bits from the unused portion to store the encoding information of a compressed packet, and the remaining unused portion is not utilized. Using the unused portion efficiently to store more metadata information (apart from the encoding) may help achieve a better compression ratio than NoΔ. This serves as the first motivation in proposing FlitZip.

2. *Simultaneous (Parallel) Computation in Existing Delta Compression Techniques is Costly:* As shown in Fig. 1, existing delta compression techniques [7], [11], [12], [18] has different compressor modules. All these modules are executed simultaneously on an incoming packet to determine whether a packet is compressible or not. Fig. 4 shows a representative example of simultaneous computation in B8Δ4 and B4Δ1 modules. We assume the packet size as 16 bytes, and each flit is of 4 bytes. From the figure, we can observe that only the last 4 bits from the LSB differ in each flit. As shown in Fig. 4(i), in *B*8Δ4 module, the base is B8 and the

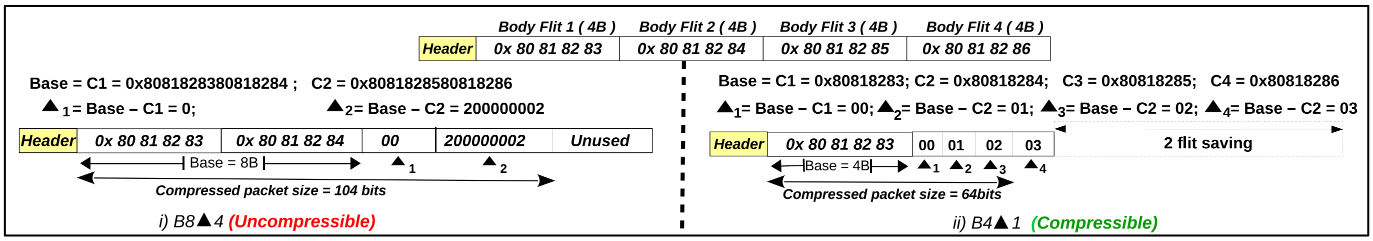


Fig. 4. Representative example showing simultaneous execution in compression modules B8Δ4 and B4Δ1 in existing delta compression techniques.

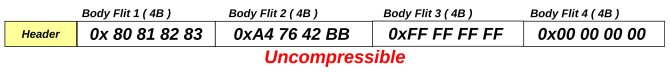


Fig. 5. Example showing delta compression is unable to compress.

packet is divided into chunks of 8 bytes, i.e., C1 and C2. Since NoΔ uses the first chunk (C1) as base, Δ_i is calculated by subtracting each chunk from the base: Δ_1 is 0 and Δ_2 is 200000002. Hence, with $B8\Delta4$, the compressed packet size does not reduce the level to save a flit as Δ_2 cannot be represented in 4 bytes, making the packet incompressible. Similarly, computation for B4Δ1 is shown in Fig. 4(ii). Out of all combinations of (B, Δ), the packet is compressible only with B4Δ1 with two flits saving.

Though simultaneous computation may help to achieve the best compression ratio, multiple modules that include several subtractors, buffers, etc., are a challenge for the area and energy-efficient design. To calculate Δ_i, the bit-width of the subtractors in the de/compressor units are equal to the base size (in bytes), which varies between 16 bytes to 4 bytes and the number of such module is equal to n where $n = \frac{Block\ size}{Base\ size}$. This serves as the second motivation in proposing a lightweight de/compression modules in FlitZip.

3. Existing Delta Compressions Cannot Reduce Redundancy in Packets Effectively: As the size of (B, Δ) is in bytes, it enforces to represent even the smallest value for Δ in a byte. Moreover, the first chunk of a packet is used as a base for compression. Hence, if the first chunk varies widely from rest of the chunks, existing delta compression techniques cannot compress the packet. We call this as *intra-packet* data pattern. This restricts packet compression as explained using Fig. 5. If the base used is either B8 (0x80818283A 47642BB) or B4 (0x80818283), the packet is incompressible. Hence, the packet is incompressible using any combination of (B, Δ).

3.1 Need for a New Compression Technique

In Fig. 5, we can observe that data within flit 1, flit 3, and flit 4 shows low deviations. Hence, if the content of each flit is divided as 1 byte chunks, the variations among the chunks are very less. The chunks of flit 1 (80, 81, 82, 83) differs by 2 bits only while the chunks of flit 3 (FF, FF, FF, FF) and flit 4 (00, 00, 00, 00) are all the same. On the other hand, the chunks of flit 2 ($A4$, 76, 42, BB) differ by 8 bits. Since existing technique searches for intra-packet data patterns, despite having such regular patterns within the flits, the packet is incompressible. This serves as the third motivation behind proposing FlitZip, where data patterns are observed within each flit separately. We call it as *intra-flit* data patterns.

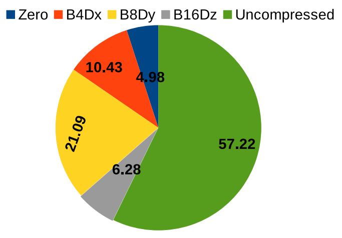


Fig. 7. Data patterns observed within packets (intra-packet patterns) that is used by existing techniques [7], [11], [12] for packet compression. B4Dx, B8Dy and B16Dz are used where x=1/2, y=1/2/4; z=1/2/4/8.

Fig. 6 shows the intra-flit data pattern (marked with different colors) in PARSEC [26] and SPEC CPU2006 [27] benchmarks. 100 percent Comp (blue) indicates that the data within a flit is the same. High Comp (grey), Medium Comp (yellow) and Low Comp (green) indicate that the intra-flit differences (di) can be represented using $1 \leq di \leq 2$ bits, $3 \leq di \leq 4$ bits, and $5 \leq di \leq 6$ bits, respectively. Uncompressed (red) are those flits whose intra-flit difference varies widely ($di \geq 7$). Hence, the lesser bits required to represent di, higher is the compression ratio. From the figure, we can observe that 21 percent flits are uncompressible and 52.2 percent of the flits has same value. Also, 3.2, 4.36 and 5.05 percent of the flits fall in the category of High, Medium, and Low Comp, respectively. Fig. 7 shows the intra-packet data pattern observed for different bases in NoΔ for PARSEC and SPEC 2006. From the figure, we can observe that 57.22 percent of the flits are not compressible while 10.43, 21.09 and 6.28 percent of the packets are compressible with B4, B8 and B16, respectively. Among these packets, only 4.98 percent of them contain only zeroes, which are compressed by the Zero module. It is also used by ZeroCompr [8] for packet compression.

Rationale: Fig. 6 shows the relevance and suitability of compression in flit level granularity as compared to the existing techniques. Thus, in the existing delta compression techniques, there may arise fewer intra-packet data patterns, making it incompressible. However, the same packet may have intra-flit data patterns which is exploitted by FlitZip for packet compression.

4 Proposed Technique: FlitZip

Fig. 8 shows an abstract view of each tile when FlitZip is incorporated. Unlike state-of-the-art techniques [7], [11],

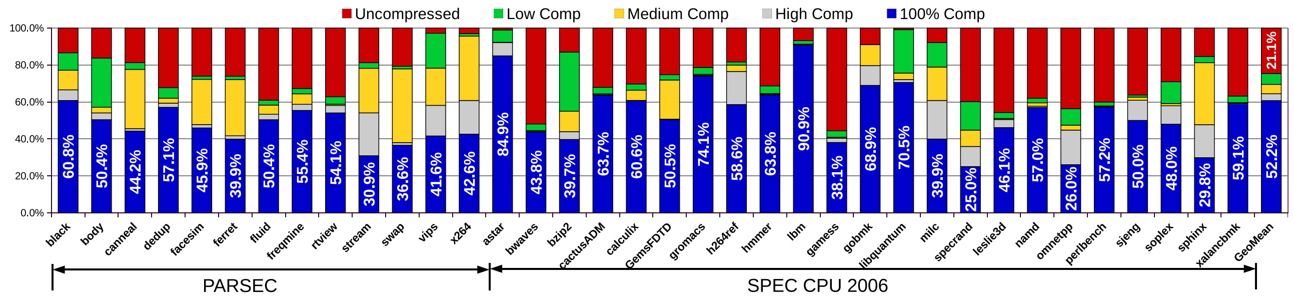


Fig. 6. Data patterns observed within each flits (intra-flit patterns) that becomes the use case for FlitZip in PARSEC and SPEC CPU 2006 benchmark. Each flit is divided into 1 byte chunks and the data among the chunks are used to determine the compressibility rates.

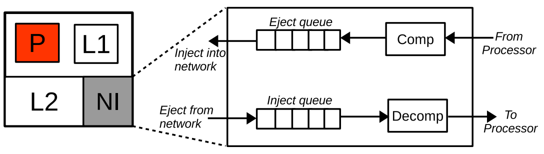


Fig. 8. Abstract view of the compressor and decompressor in a tile.

[12], only a single compressor and decompressor module is added in the NI of each tile. As shown in the figure, whenever a packet is buffered in the eject queue, FlitZip searches for intra-flit data patterns. Upon finding a pattern, each flit is represented as a set of differences, di from an optimal base, B. If the size of di is less than a byte, the flit is compressible. Thus, the proposed technique performs flit wise compression, as the name suggests *FlitZip*. To indicate whether a flit is compressed or not, a three bit encoding, E is used. In FliZip, only the di's of each flit is copied to the compressed packet, and for an uncompressed flit, the original flit content is added together to constitute the compressed packet. The base and encoding of each flit forms the metadata that is used by the decompressor to regenerate the original packet. To achieve a highly compressed packet, the metadata of each flit is stored in the head flit's unused portion. Hence, if the compressed packet consists of fewer flits than the original packet, FlitZip compresses it. Since packets are transferred on a flit-by-flit basis, a compressed packet is flitisized conventionally before injecting into the network.

Similarly, whenever a packet is ejected out of the network and stored in the inject queue, the head flit is investigated. From the head flit, the metadata of each flit is retrieved. For compressed flits, the base is added with the differences to regenerate the original flit which are reorganized to form the original packet. To the best of our knowledge, this is the first work that focuses on packet compression within flit level granularity. Throughout the paper, FlitZip uses di to represent intra-flit differences, while the existing delta compression techniques use Δ to represent the differences. Sections 4.1 and 4.2 explains the compression and decompression of FlitZip in details.

4.1 Compressor

Fig. 9 shows the compressor module in FlitZip. Initially, the compressor statically determines the original flit boundaries, which are multiples of the link bandwidth. Each flit is then divided into 1 byte chunks ($C_1 \ldots C_n$) and the *Base* is calculated as an average of the smallest (C_{small}) and largest (C_{large}) chunk value. This is because the largest and smallest value bounds the chunks. Hence, the range of data values within a flit is determined with a priority encoder that takes C_{small} and C_{large} as input and outputs the encoding bits (001-111). Section 4.3 explains the encoding in details.

If the largest and smallest chunk is the same, it indicates that the intra-flit data pattern is the same and thus 100 percent compressible (encoding $= 000$). For other cases, differences (di) are calculated by subtracting the Base from each chunk. If di can be represented with a maximum of 6 bits, the flit is compressible. This is determined using Equation (1); otherwise, the flit is uncompressible. A total of n subtractors are used for this purpose that takes the chunks and either *Base* or 0 as inputs to calculate the differences, di

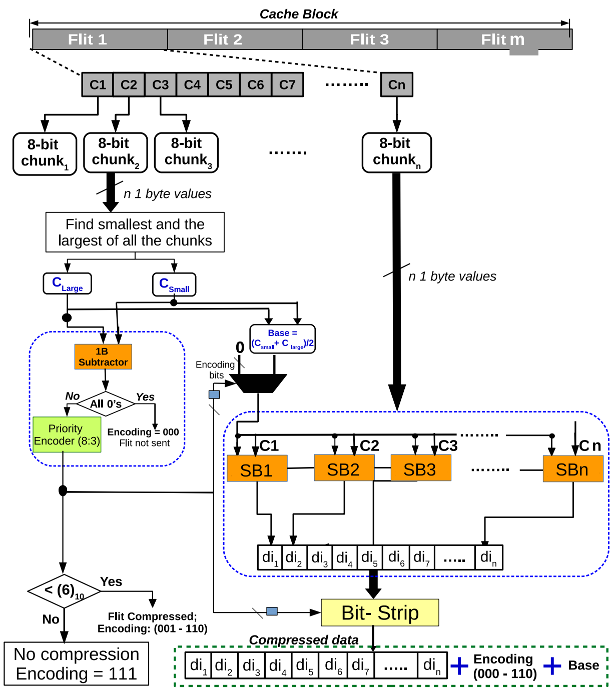


Fig. 9. Internal structure of compressor in FlitZip [A cache block consists of m number of flits and each flit is of size n bytes.]

($di = Base - C$). Since di can be either positive or negative, an extra bit is used to indicate the sign as shown in Equation (2). When compression is possible (encoding $= 001$ to 110), one of the inputs to the subtractors (SB1, ..., SBn) is always the Base. However, when compression is not possible (encoding = 111), the multiplexer sends all zeroes instead of the Base. This is done such that the chunks remain the same for an uncompressed flit. To select either Base or 0, the select line of MUX is determined using the output of the priority encoder. The signal is high when the priority encoder output is within 001 to 110 and for output 111, the signal is low. A small combinational circuit (shown as a blue rectangle near the MUX) is used to convert the encoder output into a select line.

$$max_{di} = max\{|di_1|, |di_2|, \ldots, |di_n|\}; n = \frac{flit\ size}{1byte}. \tag{1}$$

$$sizeof(max_{di}) = \lceil log_2(max_{di}) \rceil + 1. \tag{2}$$

$$\begin{aligned} ComPS = Chunks \times \{(max(|di_1|) + 1) + \ldots + \\ (max(|di_F|) + 1)\}; \\ F = number\ of\ flits; Chunks = n. \end{aligned} \tag{3}$$

Also, to represent di in a minimum number of bits, a *Bit-Strip* module is used that strips off the extra bits from each of the di's. Therefore, the priority encoder output is also required to enable or disable the Bit-Strip module. It is enabled when the signal is high (001-110) and disabled when it is low (111). When the module is disabled, no bits are stripped from the chunks, thereby avoiding flit compression. The compressor finally outputs a Base, Encoding bits, and an array of differences, di's for a compressed flit and original

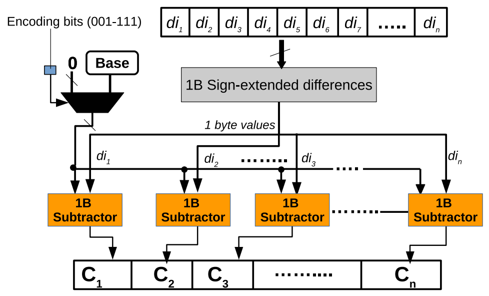


Fig. 10. Internal structure of decompressor where n is flit size in bytes.

data values for an uncompressed flit. Equation (3) determines the size of a compressed packet (ComPS).

4.2 Decompression

Fig. 10 shows the decompressor that consists of a MUX and n-subtractors. Since the head flit reaches the destination first, the decompressor decodes the metadata. The base and encoding of each flit are extracted to determine the flit boundaries. Using Equation (4), the boundary of each flit is obtained in the compressed packet. Flits with encoding 000 are regenerated by copying the base by the number of chunks. For other cases, the module takes $di_1, \ldots, di_n$ from the incoming flit and the MUX sends either base (for encoding 001 to 110) or 0's (for encoding 111, the flits are uncompressed) as inputs. Hence, the select line of the MUX is determined from the encoding bits. The size of each di is obtained from the encoding bits and is sign-extended to 1 byte values. Example: For a flit of size 4B ($Chunks = 4$), if flit 1 has an encoding of 011, each di is of 3 bits. Thus, flit 1 consists of first 12 bits ($3 \times Chunks \times 1$) where bits [0,1,2] corresponds to di_1, [3,4,5] corresponds to di_2 and so on in the compressed packet. The subtractors take the sign-extended di's and the base as inputs and finally output the original flit, which is reorganized to generate the original packet. Since in FlitZip packet received after decompression is the same as that of the original packet, it is classified as a lossless compression technique. Compression and decompression of network packets using FlitZip are explained using Fig. 11.

$$flit\ boundary_i = Encoding\ bits \times Chunks \times i; \qquad i = flit\ number, 1 \le i \le \frac{packet\ size}{link\ bandwidth}. \tag{4}$$

TABLE 1
Encoding Table Used at each NI in FlitZip

Number of bits for di	Encoding bits	Compressed-flit size (in bits)
≥7	**111**	-No-
6	110	(6+1)*16 = 112
5	101	(5+1)*16 = 96
4	100	(4+1)*16 = 80
3	011	(3+1)*16 = 64
2	010	(2+1)*16 = 48
1	001	(1+1)*16 = 32
0	000	0 (All Same; including zeroes)

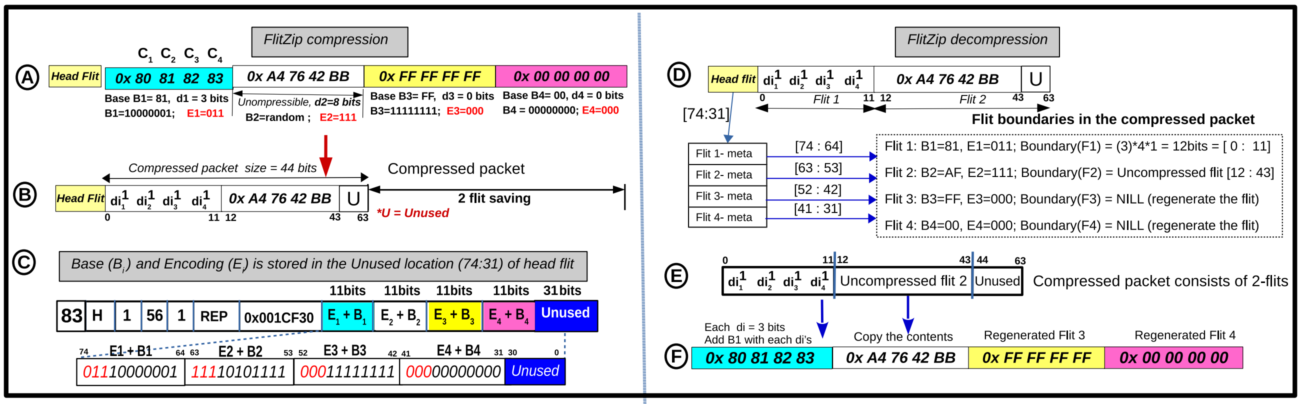
The third column is not a part of the encoding table and the compressed flit size is calculated using the original flit size and link bandwidth as 16 bytes each.

4.3 Encoding Table

Table 1 shows the encoding table present at the NI of each tile. The table consists of eight entries where each entry is a 2-tuple: $<$ di-bits, encoding bits $>$. Encoding bits store the size of di of each compressed flit that is used by the decompressor to determine the flit boundaries in a compressed packet. Three encoding bits can sufficiently represent a total of 2^3 encodings, out of which 111 and 000 have a special meaning. 111 indicates that the flit is uncompressed, and 000 means that the flit chunks are the same. Rest combination indicates various encodings used for different di-bits. For example: if di requires 5 bits for a compressed flit, the encoding is 101.

4.4 Metadata Storage in Head Flit

Another property of FlitZip is that it reduces the packet size further by storing the metadata of a compressed packet in the corresponding head flit's unused location. As shown in Fig. 3, for 128 bit link bandwidth, 75 bits from the head flit is unused. For a 64B packet (4 body flits), each flit requires a metadata of 11 bits (Base $= 8$ bits, Encoding $= 3$ bits). Hence, a total of 44 $(11 * 4)$ bits from the head flit with bit location [74:31] is used to store all the flits' metadata. Bits [74:64] contain the encoding bits and base of flit 1 out of which bits [74:72] are the encoding bits, and the next 8 bit is the base. Similarly, bits ranging from [63:53], [52:42] and [41:31] contains the metadata for flit 2, 3 and 4, respectively.



FlitZip compression
A
Head Flit
0x 80 81 82 83
0x A4 76 42 BB
0x FF FF FF FF
0x 00 00 00 00
Base B1= 81, d1 = 3 bits B1=10000001; E1=011
Unpromissible, d2=8 bits B2=random ; E2=111
Base B3= FF, d3 = 0 bits B3=11111111; E3=000
Base B4= 00, d4 = 0 bits B4 = 00000000; E4=000
Compressed packet size = 44 bits
Compressed packet
B
2 flit saving
*U = Unused
C
Base (Bi) and Encoding (E) is stored in the Unused location (74:31) of head flit
83 H 1 56 1 REP 0x001CF30
E1 + B1
E2 + B2
E3 + B3
E4 + B4
Unused
01110000001
11110101111
00011111111
00000000000
FlitZip decompression
D
Flit 1
Flit 2
[74:31]
Flit boundaries in the compressed packet
Flit 1- meta
Flit 2- meta
Flit 3- meta
Flit 4- meta
[74 : 64]
[63 : 53]
[52 : 42]
[41 : 31]
Flit 1: B1=81, E1=011; Boundary(F1) = (3)*4*1 = 12bits = [0 : 11]
Flit 2: B2=AF, E2=111; Boundary(F2) = Uncompressed flit [12 : 43]
Flit 3: B3=FF, E3=000; Boundary(F3) = NILL (regenerate the flit)
Flit 4: B4=00, E4=000; Boundary(F4) = NILL (regenerate the flit)
E
Uncompressed flit 2
Unused
Compressed packet consists of 2-flits
Each di = 3 bits Add B1 with each di's
Copy the contents
Regenerated Flit 3
Regenerated Flit 4
F
0x 80 81 82 83
0x A4 76 42 BB
0x FF FF FF FF
0x 00 00 00 00

Fig. 11. Example of FlitZip compression (left half) and decompression (right half). The uncompressed packet size and flit size is 16 bytes and 4 bytes.

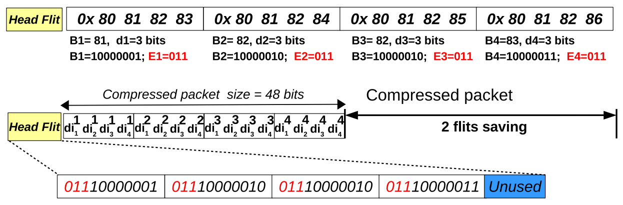


Fig. 12. FlitZip can compress all packets that existing technique does.

4.5 Illustrative Example of FlitZip De/Compression

Fig. 11 shows the working methodology of FlitZip. The left half of the figure shows the same flit as in Fig. 5 that NoΔ could not compress. Flit 1 has four chunks: $C1 = 80$, $C2 = 81$, $C3 = 82$ and $C4 = 83$ out of which C_{small} is 0x80 and C_{large} is 0x83. Therefore, base B1 is 0x81 (10000001) and di requires 3 bits (2+1 sign bit). Thus, flit 1 is compressible and the encoding is *011*. Flit 2 is not compressible as di requires 8 bits. Therefore, the encoding for flit 2 is *111*. For flits 3 and 4, each flit has same data values. These flits are not transmitted and the (encoding, base) used for flit 3, 4 is (*000*, *11111111*) and (*000*, *00000000*), respectively. Fig. 11 B shows the content of the compressed packet using FlitZip where di^i_j is the *jth* difference of flit *i*. The encoding and base of each compressed flit is stored in the head flit as shown in Fig. 11 C. Thus, FlitZip compresses the packet in Fig. 5 by 50 percent which NoΔ is unable to compress.

The right half of Fig. 11, shows FlitZip decompression. From the head flit of Fig. 11 D, the metadata of the compressed packet is extracted. Bits [74, 73, 72] is 011 which means that the number of bits used for di in flit 1 is 3. Thus, flit 1 is of 12 bits ($3 \times Chunks \times 1$). As shown in Fig. 11 E, in the compressed packet the flit ranges from [0:11]. The base extracted from the head flit is added with di^1_i to generate the original flit. For flit 2, bits [65:63] from the head flit is 111. Hence the flit boundary is [12:43] (original flit size is 32 bits). On the other hand, for flit 3 and 4, the encoding bits are 000. Both the flits are regenerated by copying the bases by the number of chunks. The decompressed flit is shown in Fig. 11 F which is same as that of Fig. 11 A. Thus, FlitZip is a lossless compression that can effectively reconstruct the original packet from a compressed packet.

Advantage of FlitZip. Unlike the existing delta compression techniques [7], [11], [12], FlitZip avoids the requirement of multiple de/compression units with varying bases. This addresses the second problem of simultaneous computation, as mentioned in Section 3. Also, FlitZip can compress all packets that NoΔ and DISCO can, in addition to that, FlitZip compresses some extra packets that NoΔ cannot. Hence, it can compress more packets than the existing techniques, as shown in Fig. 12. The figure is the same as that of Fig. 4 where existing delta compression techniques compresses the packet using B4Δ1 with the same compression ratio of 50 percent. Experimentally, we found that FlitZip can compress an extra 27 percent of packets compared to the existing techniques across all the benchmarks.

5 Experimental Analysis

For experimental purposes, we use gem5 [36] a cycle-accurate full system simulator with ruby memory module and GARNET [37] in gem5 to model the NoC. The baseline configuration used is mentioned in Table 2 without packet compression. We simulate different multicore system (2x2, 4x4 and 8x8) and have performed extensive analysis using 13 multithreaded PARSEC [26] benchmarks and 384 workload mixes for 64 cores from SPEC 2006 benchmarks [27]. For performance metrics, we have used flit count, weighted speedup [38], compression ratio, packet latency, bandwidth utilization and packet queuing latency. We analyze the sensitivity of FlitZip to various parameters: network size, base size, block size and NoC bandwidth. Among state-of-the-art delta compression techniques, we have compared FlitZip with NoΔ initially. We have also compared FlitZip with other packet compression techniques: ZeroCompr [8], FPC [10] and DISCO [11] in Section 5.2.

TABLE 2
Simulation Parameters

Processor	4/16/64, x86 cores OoO superscalar
Processor frequency	3 GHz
NoC operating frequency	1GHz
L1 I & D cache/core	32KB, 4-way associative, 64B block
L2 cache (shared)	1MB, 8-way associative, 64B block
DRAM configuration	DDR3, 4GB, 64-bits channel, 2 ranks/channel, 8 banks/rank
L1 and L2 cache access time	2, 10 cycles
DRAM access time	100 cycles
Coherence	MESI CMP directory protocol
NoC topology	8x8 2D mesh with XY routing 1 cycle - link delay, 2 cycle-router delay
NoC Link Bandwidth	128 bits (= flit size)
Packet size	1-flit request and 4-flit reply
Number of VCs/ router	5, buffer-depth = 4

Workload Characterization: We have classified the benchmarks of SPEC CPU 2006 suite into three categories ($B1$, $B2$ and $B3$) based on Misses Per Kilo Instructions (MPKI) from L1 cache. B1 (MPKI < 0.25) contains *gromacs*, *sjeng*, *bwaves*, *hmmer*, *calculix*, *gobmk*, *cactusADM*, *namd*, *sphinx* and *h264ref*. B2 ($0.25 \leq$ MPKI < 0.5) consists of *astar*, *specrand*, *mcf*, *milc*, *omnetpp* and *gamess*, and B3 (MPKI ≥ 0.5) consists of benchmarks *bzip2*, *lbm*, *leslie3d*, *soplex*, *GemsFDTD*, *perlbench*, *xalancbmk* and *libquantum*. Since SPEC is a multiprogrammed benchmark, a single benchmark runs on a single core. Hence, we create a workload mix combining benchmarks from various categories ($B1$, $B2$ and $B3$) to extensively evaluate the system performance.

Table 3 shows 16 different workload mixes ($M1$, ..., $M16$) generated by mixing B1, B2 and B3 in different ratios. The benchmarks in each workload mixes are mapped to 64 cores in 24 different ways as shown in Fig. 13. For example, in the first mapping, the first benchmark is mapped to core [0-15], the second benchmark is mapped to core [16-31], the third benchmark is mapped from core [32-47], and the fourth benchmark is mapped from core [48-63]. Hence, a total of 384 different workloads for 64 core and 128 for 16 core are created to evaluate the effectiveness of FlitZip. The workloads are *fast forwarded* for 10 billion instructions, *warm up* for next 1 billion instructions and then *executes* for next 2 billion instructions for collecting the statistics. The results plotted for each Mi is the geometric mean of 24 different ways

TABLE 3
Workload Mixes Created From SPEC 2006 Benchmark

M1, M2, M3	B1 (all), B2 (all), B3 (all)
M4 , M5 M6, M7 M8, M9	0.75B1 - 0.25B2, 0.75B1 - 0.25B3 0.75B2 - 0.25B3, 0.75B3 - 0.25B2 0.75B3 - 0.25B1, 0.75B2 - 0.25 B1
M10, M11 M12	0.5B1 - 0.5B2, 0.5B1 - 0.5B3 0.5B2 - 0.5B3
M13, M14, M15	0.25B1 - 0.5B2 - 0.25B3 0.25B1 - 0.5B3 - 0.25B2 0.25B2 - 0.5B1 - 0.25B3
M16	Random combination

xB_i - yB_j constitutes a mix where x% of B_i and y% of B_j are used where $1 \leq i,j \leq 3$. B1: (MPKI $\leq$ 0.25), B2: (0.25 $\leq$ MPKI $<$ 0.5) and B3: (MPKI $\geq$ 0.5).

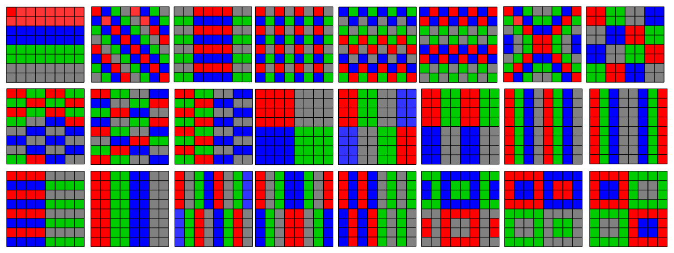


Fig. 13. 24 different ways of mapping SPEC 2006 benchmarks to each core in 8x8 NoC based MPSoC. Different colors indicate a separate benchmark application running on a core from B1, B2, and B3 categories.

of workload mapping. For PARSEC benchmarks, we have used *blackscholes (black)*, *body*, *canneal*, *dedup*, *facesim*, *ferret*, *fluidanimate (fluid)*, *freqmine*, *rtview*, *streamcluster (stream)*, *swaptions (swap)*, *vips* and x264. In PARSEC, each core runs a separate thread of the same benchmark.

5.1 Performance Evaluation for 8x8 NoC

Compression Ratio (CR): CR is measured as the ratio of compressed flits to uncompressed flits in the network. The higher the compression ratio, the better is the technique. Fig. 14 shows the compression ratio achieved by FlitZip and NoΔ. On average (geometric mean), FlitZip achieves a compression ratio of 0.52, whereas the compression ratio for NoΔ is 0.30 only. Kindly note that the packet count in both the technique remains the same. It only reduces the number of flits within a packet, thereby reducing the packet size. Fig. 15 shows the normalized flit count after applying FlitZip compression. From the figure, we can observe that FlitZip effectively reduces flit count in the network by 42.53 percent whereas NoΔ reduces the flit count by 16 percent as compared to the baseline. FlitZip achieves higher compression ratio than NoΔ as shown in the figure with the highest/lowest compression ratio of 0.59/0.44 in *x264*/*fluid* for PARSEC benchmark, and 0.52/0.19 in *M3*/*M14*/*M16* for SPEC CPU 2006 workload mixes. *x264* is a multimedia application where adjacent frames of the application are very much alike. When the frames are transferred across the network as packets, the similarity among the adjacent frames made it possible to achieve a higher compression ratio among all the PARSEC benchmarks. *vips* is also a multimedia application and achieves a fairly higher

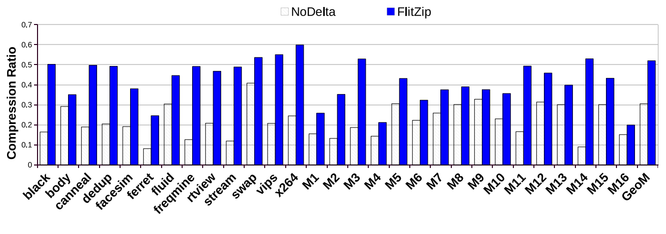


Fig. 14. Compression ratio in PARSEC and SPEC CPU 2006 benchmark in 8x8 NoC.

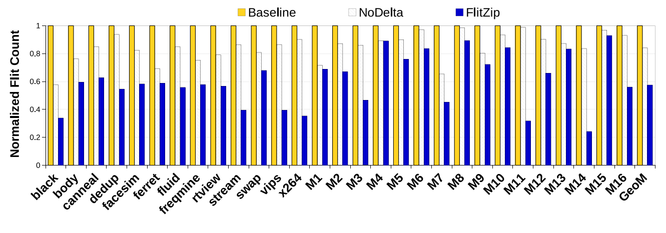


Fig. 15. Normalized flit count with respect to baseline (=1) in PARSEC and SPEC 2006 benchmark in 8x8 NoC.

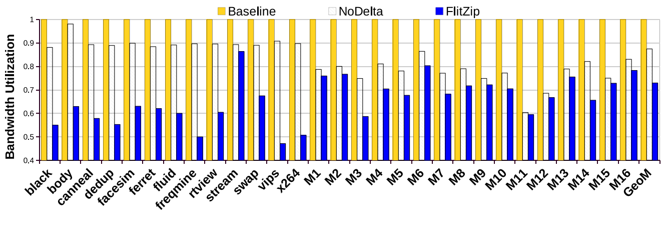


Fig. 16. Normalized bandwidth utilization with respect to baseline (=1) in PARSEC and SPEC 2006 benchmark in 8x8 NoC.

compression ratio than the other benchmarks. Since FlitZip stores the base of compressed flits in the unused location of the head flit, the packets are highly compressed as compared to NoΔ.

Bandwidth Utilization (BU): BU is the average number of flits passing through a link per cycle. Fig. 16 shows a comparison of normalized bandwidth utilization in NoΔ and FlitZip w.r.t. baseline. On average, FlitZip and NoΔ reduce the bandwidth utilization by 27 and 12.5 percent, respectively, to transfer flits across the tiles. In NoC, on-chip bandwidth is an essential component used for carrying flits from one router to the other. An increase in the compression ratio of network packets reduces the number of flits transferred between the tiles. Thus, the usage of on-chip bandwidth also reduces. Among the PARSEC and SPEC 2006 benchmarks, *x264* (50 percent) and M3/M14 (41 percent) workload mixes have the lowest bandwidth utilization rate, respectively. Workload mixes M3 and M14 consists of B3 benchmark mixes (c.f. Table 3) which is a combination of *lbm-libquantum-soplex-GemsFDTD-bzip2-xalancbmk* benchmarks. Since these combinations achieve a higher compression ratio than NoΔ, it helps in compressing packets efficiently, thereby generating less flit count.

Weighted Speedup (WS): Since we use a multicore system, weighted speedup gives a better view of the throughput achieved by the system. It is calculated as $WS = \sum_{i=0}^{P-1} \frac{IPC_i^{shared}}{IPC_i^{alone}}$ where IPC_i^{shared} is the Instructions Per Cycle (IPC) of core i shared with other (P-1) applications running on a MPSoC of P cores and IPC_i^{alone} is the IPC of core i running alone in a MPSoC of P cores. As shown in Fig. 17, FlitZip provides a

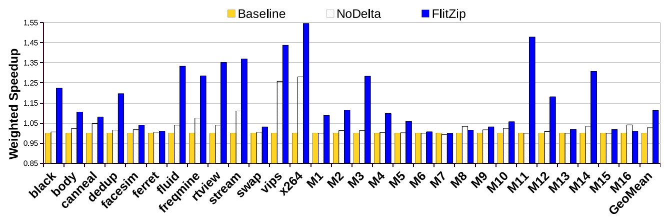


Fig. 17. Normalized weighted Speedup with respect to baseline (=1) in PARSEC and SPEC 2006 benchmark in 8x8 NoC.

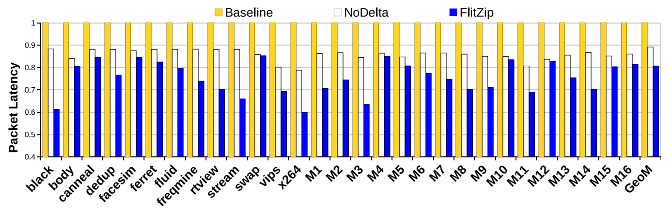


Fig. 18. Normalized Packet Latency with respect to baseline (=1) in PARSEC and SPEC 2006 benchmark in 8x8 NoC.

better trade-off between performance (speedup) and bandwidth utilization as it provides 12 percent performance improvement with a 39.87 percent reduction in flit count compared to the baseline. Also, when compared with NoΔ, the weighted speedup of FlitZip increases by 8.35 percent. The primary goal of FlitZip is to reduce redundancies within packets such that the request and reply packets are transmitted faster to the requesting tile. Since FlitZip achieves a higher compression ratio than NoΔ, a packet takes less time to reach the core. It results in faster instruction execution, thereby increasing the system throughput. Among the PARSEC and SPEC 2006, *x264*, and the mix *M11/M3/M14* are the major performance gainers as compared to the baseline and NoΔ.

Average Packet Latency (PL): Fig. 18 shows that FlitZip reduces packet latency in PARSEC and SPEC CPU 2006 mixes by 13.57 and 14.4 percent, respectively. Since FlitZip reduces network traffic, the time taken by a packet to reach the requesting tile decreases. Hence, on average, FlitZip reduces the packet latency by 19.28 percent as compared to the baseline, whereas, NoΔ reduces packet latency by 11 percent, only.

Average Packet Queuing Latency (PQL): Packet queuing latency is the average amount of time when the flits of a packet are queued in the internal buffers of a router, i.e.,

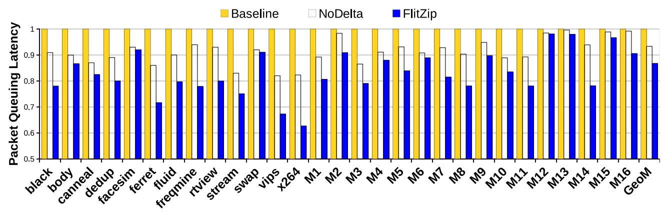


Fig. 19. Normalized Packet Queuing Latency with respect to baseline (=1) in PARSEC and SPEC CPU 2006 benchmark in 8x8 NoC.

Virtual Channel (VC) during its transmission from a source tile to a destination tile. Fig. 19 shows that FlitZip reduces the queuing latency by an average of 13.3 and 7.15 percent (geometric mean) compared to baseline and NoΔ, respectively. Since FlitZip efficiently reduces the flit count within a packet, the queuing time of the packet reduces.

5.2 Sensitivity Analysis

5.2.1 Change in Network Size

Table 4 shows the performance of FlitZip as compared to NoΔ and baseline with varying network size between 4 cores (2x2 NoC) to 64 cores (8x8 NoC). Each entry is in the form of *A*% (B%/C%) where A% is the geometric mean over all the workloads and B%, C% is the highest and lowest respective values. We can observe that on average FlitZip performs better across different NoC sizes as compared to NoΔ and baseline. It is also observed that the performance of 8x8 NoC is better than 4x4, which is better than 2x2. This is because the bigger the network diameter, the larger is the time taken by a flit to reach its destination (as end to end hop count increases). Also, the number of applications running in 8x8 NoC is 4 times more than that of a 4x4 NoC, which generates more traffic to the network. As network size increases beyond 8x8, the main memory supported by the system reduces as discussed next.

5.2.2 Change in Block Size, Link Bandwidth and Memory Size

Table 5 shows the sensitivity of FlitZip for various packet sizes when the NoC link bandwidth is 128 bits and 256 bits, respectively. From the table, we can observe that for 128 bit link bandwidth, FlitZip can support up to a maximum of 96 byte block size (6 body flits) only. However, for 256 link bandwidth, no

TABLE 4
Sensitivity Analysis of FlitZip With Varying Network Size Normalized With Respect to Baseline and NoΔ

	WS	PL	BU
2x2 NoC (4 cores : 64 SPEC workloads + 13 PARSEC benchmarks)			
NoΔ	5.88%(20.45%/0.66%)	8.65%(18.2%/9.62%)	12.9%(25%/1.5%)
Baseline	9.86%(45.54% / 0.01%)	13.21%(27.4% / 10%)	21%(39.1% / 11.2%)
4x4 NoC (16 cores:128 SPEC workloads + 13 PARSEC benchmarks)			
NoΔ	7.08%(25%/1.01%)	10%(20%/12%)	15.2%(34.5%/1.18%)
Baseline	11.05%(49.76% / 0.086%)	15.76%(31.3% / 10.2%)	22.5%(45.8% / 11.27%)
8x8 NoC (64 cores : 384 SPEC workloads + 13 PARSEC benchmarks)			
NoΔ	8.35%(28.03%/1.12%)	9.46%(21.82%/11.78%)	16.56%(48.02%/1.52%)
Baseline	12%(54.56%/0.1%)	19.28%(38.8%/14.4%)	27%(50%/13.6%)

WS: Weighted Speedup, PL: Packet Latency, BU: Bandwidth Utilization.

TABLE 5
Sensitivity Analysis With Varying Block Size and NoC Link Bandwidth

Link bandwidth	Block/packet size	#Flits	Metadata bits
128 bits (16B), 75 unused bits	16 B	1	11
	32 B	2	22
	48 B	3	33
	64 B	4	44
	80 B	5	55
	96 B	6	66
256 bits (32B), 203 unused bits	32 B	1	11
	64 B	2	22
	96 B	3	33
	128 B	4	44
	256 B	8	88
	512 B	16	176

such issue arises. Thus, FlitZip can compress a packet if it contains a maximum of $(Unused\_Bits)/11$ number of flits.

Also, it might seem that the head flit may not be sufficient enough to store the metadata when either the address space increases or the main memory size increases. Since FlitZip is applied on NoC packets and NoC communication occurs while accessing the Last Level Cache (LLC), all addresses generated for LLC accesses are physical addresses[2] and not virtual. Thus, the virtual address generated by the CPU is translated into a physical address that depends on the main memory size. In the case of 64B cache block size and 128 bit link bandwidth, FlitZip requires 44 bits (11 bits per flit) for metadata, and a total of 21 bits is used for various fields such as Flit Type, Source, Destination, etc. as shown in Fig. 3. Hence, 63 bits (128-21-44 = 63) are left for the MEM-ADDR field, thereby supporting a main memory size of 2^{63} (8 Exabytes) which is sufficient for present-day architecture.

However, if we anticipate the growth in main memory size beyond 8 Exabytes in the future, the extra memory support can be provided by not sending the redundant block offset bits in the MEM-ADDR field. Since data requested by a CPU is transferred from the LLC in block-level granularity, only the index and tag bits are required to fetch a block. This makes the offset bits in the MEM-ADDR redundant. Also, for fixed address space, the set index potion decreases with the increase in the block size. Thus, if the block bits are not sent as a part of MEM-ADDR, the head flit's unused portion increases. Example: For a 64B block size, if the 6 offset bits are not sent as a part of MEM-ADDR, the maximum memory supported is 2^{69} ($2^{63} \times 2^{6}$ = 512 Exabytes). Also, when the network size increases beyond 8x8, it can be accommodated by FlitZip using a few bits from the MEM-ADDR field. In such a case, the supported main memory for 128 bit link bandwidth and network size of 16x16 and 32x32 is 32 Exabytes and 2 Exabytes, respectively.

5.2.3 Change in Base Size

Fig. 20 shows the normalized count of packets that FlitZip can compress when the base size is varied between 1 byte to 8 bytes for de/compression. From the figure, we can observe that FlitZip can compress maximum packets across all the benchmarks when the base size is 1 byte. Since integer/floating-point data are in multiples of bytes, considering the lower byte granularity helps FlitZip find the maximum patterns within a flit. On the other hand, NoΔ uses varying bases whose size is in multiple of bytes. The benefit of using constant base size is to avoid multiple de/compressor modules. The figure also indicates that the probability of finding patterns on a larger base size ($>1 byte$) is less than that of 1 byte, which helps FlitZip achieve a higher packet compression ratio than NoΔ.

5.2.4 Comparison of FlitZip With Other Techniques

We have also implemented ZeroCompr [8], FPC [10] and DISCO [11] in 8x8 NoC for evaluating the performance of FlitZip. Fig. 21 shows the average performance of FlitZip with respect to the existing techniques. It can be observed from the figure that FlitZip, on average, performs better than all the techniques. This is well explained because only 4.98 percent of all packets contains only zeroes averaged across all the benchmarks. Hence, ZeroCompr compresses only a small number of packets inspite of having different patterns within a packet (Fig. 6). On the other hand, in FPC, the encoding table has a restricted count of eight patterns only. Also, packet de/compression in FPC requires indexing into the encoding and decoding table, which is time consuming (5 cycles). This degrades the performance of FPC as compared to FlitZip. DISCO router with NoΔ/BDI compression for NoC packets, on the other hand, performs better than ZeroCompr and FPC, but the performance degrades as compared to FlitZip. This is because DISCO router selectively compresses packets that experience a larger waiting time in the router's internal buffers. So it avoids compressing those packets with the least waiting time even if the packet content has a well-defined pattern. Though the selection criteria are advantageous in avoiding the extra de/compression latency for the compressed packets, the count of compressed packets is limited compared to FlitZip.

6 Hardware and Power Analysis

We have also implemented FlitZip in hardware using $45nm$ technology. The de/compressor units have been tested for power, area, and timing estimations by Verilog post-synthesis simulation, and we synthesize the Verilog implementation in Synopsys Design Compiler for both FlitZip and NoΔ. Also, DSENT [39] is used to analyze the dynamic power consumption of NoC at $45nm$ technology.

6.1 Timing Aspects of FlitZip De/Compressors

In FlitZip, a compressor circuit requires two cycles and a decompressor circuit requires one cycle. Since the head flit is the first to reach the requesting tile, the body flits can be decompressed as soon as they arrive at the network interface of the requesting tile. In zero-network load, consecutive flits arrive at a router after every cycle, and the router requires two additional cycles to process it, thereby requiring a total of three cycles per body flit (1 cycle-link traversal and 2 cycle-processing at the local router). Hence, the decompression latency for all body flits is hidden by the flits' queuing latency except for the last flit. The last flit requires one cycle to

2. LLC is always Physically Indexed and Physically Tagged in all types of caches i.e., PIPT or VIPT.

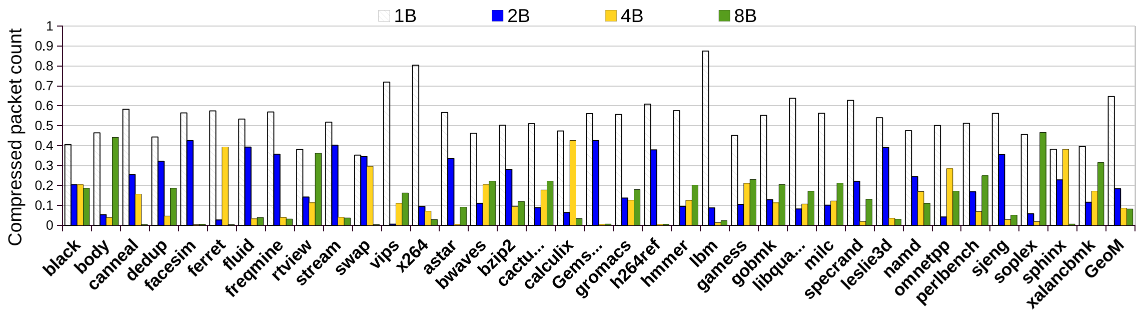


Fig. 20. Sensitivity analysis of FlitZip with respect to different base sizes: 1B, 2B, 4B and 8B for 8x8 NoC [Packet size: 64 bytes and flit size: 16 bytes.]

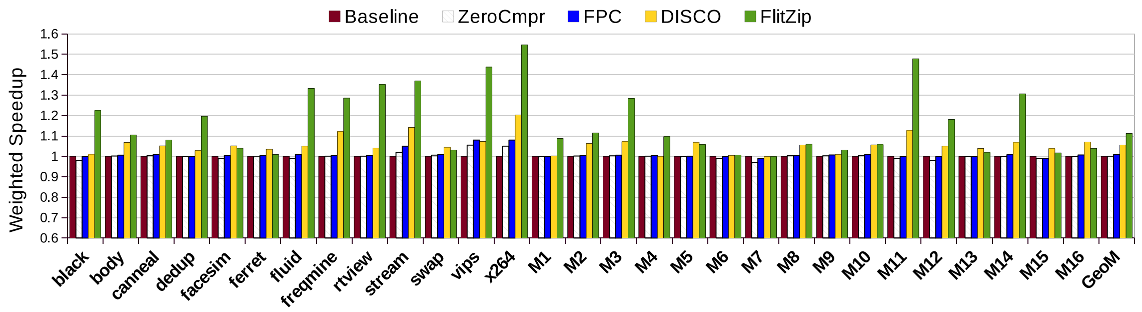


Fig. 21. Performance of FlitZip compared to ZeroCompr, FPC and DISCO normalized to baseline in 8x8 NoC.

decompress, thereby reducing the decompression latency to a single clock cycle with a single decompression circuit. The number of compressors that FlitZip can use is a design choice that depends on whether it is used for a low-power or a low-latency system. A single compressor can be used for a low power system where the flits are compressed serially one after another. In such a scenario, compression may require more cycles. On the other hand, for a system with low latency, multiple compressors can be used that can compress all the flits in parallel [18]. In our experiment, we have used four compressors to compress a four flit packet, in parallel.

6.2 Power and Area Aspects of FlitZip De/Compressors

FlitZip does not change the router microarchitecture and it is similar to that of Fig. 2b. The de/compressor units are added to the NI. Since FlitZip uses a single static base size of 1 byte, it avoids multiple compression modules, unlike NoΔ. The chunks, C_i and base being of equal size (1 byte), FlitZip avoids extra bit padding for computing the differences, di_i. In addition to these, the size of the base and difference (di) is $1B$ each. Hence, the bit-width of the subtractor units reduces from $16B$ to $1B$ in FlitZip. The power required for sign-extending, i.e., fan-in AND and OR gates, is also avoided in the FlitZip compressor as the chunks and base are of equal size, unlike NoΔ. It greatly reduces the size and dynamic power consumption of the de/compressor unit, thereby making compression and decompression energy efficient. Also, the encoding table at each tile requires 48 bits (6 bits*8 = 6B) which is nearly 97.6 percent less storage than the encoding table used in NoΔ per tile.

The de/compressor unit in FlitZip, being an additional component added in the NI, requires an extra power of $13.75\,mW$ while NoΔ requires $36.48\,mW$ of power consumption. Hence, FlitZip reduces power by 62.3 percent as compared to NoΔ. Also, the combined area of the de/compressor module in FlitZip and NoΔ is $0.0028\,mm^2$ and $0.006\,mm^2$, respectively. Thus the area of the de/compressor module in FlitZip is also reduced by 53.33 percent than that of NoΔ. The reduction in area and power consumption in FlitZip is obtained because NoΔ uses multiple de/compression modules for various (Base, Δ) sizes executed in parallel on a packet, and each module consumes additional power, which is completely avoided in FlitZip. Moreover, efficient reduction in the flit count resulted in carrying fewer flits in the NoC. Hence, FlitZip reduces the dynamic power consumption in the network by 16.68 percent as compared to NoΔ. The de/compression circuit in FlitZip being very simple and power efficient, FlitZip can become a better packet compression technique for NoC.

7 Conclusion

Present-day application generates large volume of data that demands large on-chip caches and network bandwidth. Rather than increasing the cache size or network bandwidth, squeezing data together may help satisfy modern-day applications' needs. *FlitZip* proposed in the paper compresses NoC packets on per flit basis more efficiently than the state-of-the-art techniques. It utilizes the unused bits of the head flit to store the bases of each compressed flit, thereby providing better compression of 52 percent. It is also efficient in terms of power consumption, thereby reducing the power consumption of de/compressor units by 62.3 percent with reduced bandwidth utilization of 27 percent across all the workloads.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. 38th Des. Automat. Conf.*, 2001, pp. 684–689.

[2] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[3] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar operand networks: On-chip interconnect for ILP in partitioned architectures," in *Proc. 9th Int. Symp. High-Perform. Comput. Archit.*, 2003, pp. 341–353.

[4] 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, Sep./Oct. 2007.

[5] M. Buckler, W. Burleson, and G. Sadowski, "Low-power networks-on-chip: Progress and remaining challenges," in *Proc. Int. Symp. Low Power Electron. Des.*, 2013, pp. 132–134.

[6] A. BenAchballah, S. BenOthman, and S. BenSaoud, "Problems and challenges of emerging technology networksonChip: A review," *Microprocessors Microsystems*, vol. 53, pp. 1–20, 2017.

[7] J. Zhan, M. Poremba, Y. Xu, and Y. Xie, "NoΔ: Leveraging delta compression for end-to-end memory access in NoC based multicores," in *Proc. 19th Asia South Pacific Des. Automat. Conf.*, 2014, pp. 586–591.

[8] R. Das *et al.*, "Performance and power optimization through data compression in network-on-chip architectures," in *Proc. IEEE 14th Int. Symp. High Perform. Comput. Archit.*, 2008, pp. 215–225.

[9] S. Ogg and B. Al-Hashimi , "Improved data compression for serial interconnected network on chip through unused significant bit removal," in *Proc. 19th Int. Conf. VLSI Des. held jointly with 5th Int. Conf. Embedded Syst. Des.*, 2006, p. 5.

[10] P. Zhou *et al.*, "Frequent value compression in packet-based NoC architectures," in *Proc. Asia South Pacific Des. Automat. Conf.*, 2009, p. 13–18.

[11] Y. Wang, Y. Han, J. Zhou, H. Li, and X. Li, "DISCO: A low overhead in-network data compressor for energy-efficient chip multi-processors," in *Proc. Des. Automat. Conf.*, 2016, pp. 1–6.

[12] Y. Wang, H. Li, Y. Han, and X. Li, "A low overhead in-network data compressor for the memory hierarchy of chip multiprocessors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 6, pp. 1265–1277, Jun. 2018.

[13] R. Boyapati, J. Huang, P. Majumder, K. H. Yum, and E. J. Kim, "APPROX-NoC: A data approximation framework for network-on-chip architectures," in *Proc. 44th Annu. Int. Symp. Comput. Architecture*, 2017, pp. 666–677.

[14] Y. Chen and A. Louri, "An approximate communication framework for network-on-chips," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1434–1446, Jun. 2020.

[15] M. F. Reza and P. Ampadu, "Approximate communication strategies for energy-efficient and high performance NoC: Opportunities and challenges," in *Proc. Great Lakes Symp. VLSI*, 2019, pp. 399–404.

[16] Y. Chen, M. F. Reza, and A. Louri, "DEC-NoC: An approximate framework based on dynamic error control with applications to energy-efficient NoCs," in *Proc. IEEE 36th Int. Conf. Comput. Des.*, 2018, pp. 480–487.

[17] Y. Zhang *et al.*, "Improving restore performance for in-line backup system combining deduplication and delta compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, pp. 2302–2314, Oct. 2020.

[18] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn.*, 2012, pp. 377–388.

[19] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, "MemZip: Exploring unconventional benefits from memory compression," in *Proc. 20th Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 638–649.

[20] R. Kanakagiri, B. Panda, and M. Mutyam, "MBZip: Multiblock data compression," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 4, pp. 42:1–42:29, 2017.

[21] S. Mittal and J. S. Vetter, "A survey of architectural approaches for data compression in cache and main memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1524–1536, May 2016.

[22] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high-performance processors," in *Proc. 31st Annu. Int. Symp. Comput. Archit.*, 2004, p. 212.

[23] S. Baek, H. G. Lee, C. Nicopoulos, J. Lee, and J. Kim, "Size-aware cache management for compressed cache architectures," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2337–2352, Aug. 2015.

[24] A. Ghasemazar, P. Nair, and M. Lis, "Thesaurus: Efficient cache compression via dynamic clustering," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, p. 527–540.

[25] G. Pekhimenko *et al.*, "Linearly compressed pages: A low-complexity, low-latency main memory compression framework," in *Proc. 46th IEEE/ACM Int. Symp. Microarchit.*, 2013, p. 172–184.

[26] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, Princeton, NJ, USA, Jan. 2011.

[27] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, pp. 1–17, 2006.

[28] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," *SIGARCH Comput. Archit. News*, vol. 30, no. 5, pp. 211–222, 2002.

[29] J. Lira, C. Molina, and A. Gonzalez, "HK-NUCA: Boosting data searches in dynamic non-uniform cache architectures for chip multiprocessors," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2011, pp. 419–430.

[30] S. R. Vangal *et al.*"An 80-tile sub-100-W teraFLOPS processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[31] S. Bell *et al.*, "TILE64 - Processor: A 64-Core SoC with mesh interconnect," in *Proc. Int. Solid-State Circuits Conf. - Dig. Tech. Papers*, 2008, pp. 88–598.

[32] A. Sodani *et al.*, "Knights landing: Second-generation Intel Xeon Phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, Mar./Apr. 2016.

[33] W. Dally, "Virtual-channel flow control," *Trans. Parallel Distrib. Syst.*, vol. 3, no. 2, pp. 194–205, 1992.

[34] Y. J. Yoon, N. Concer, M. Petracca, and L. Carloni, "Virtual channels vs. multiple physical networks: A comparative analysis," in *Proc. 47th Des. Automat. conf.*, 2010, pp. 162–165.

[35] V. Y. Raparti and S. Pasricha, "DAPPER: Data aware approximate NoC for GPGPU architectures," in *Proc. Int. Symp. Networks-on-Chip*, 2018, pp. 1–8.

[36] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Architecture News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[37] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha, "GARNET: A Detailed on-chip network model inside a full-system simulator," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, 2009, pp. 33–42.

[38] Kun Luo, J. Gummaraju, and M. Franklin, "Balancing thoughput and fairness in SMT processors," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, 2001, pp. 164–171.

[39] C. Sun *et al.*, "DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proc. Int. Symp. Networks-on-Chip*, 2012, pp. 201–210.

Dipika Deb received the BTech degree in Computer Science and Engineering (CSE) from Tezpur University, Assam, India, and the MTech degree in CSE from the Indian Institute of Technology Guwahati, India. She is currently a research scholar with the Department of CSE, Indian Institute of Technology Guwahati, India. Her research interests include multicore computer architecture, prefetching, packet compression, network-on-chip, and machine learning.

Rohith M.K. received the BTech degree in Electronics and Communication Engineering from the RV College of Engineering, Banaglore, India. He is currently an ASIC engineer with NVIDIA, Bangalore, India. His research interests include multicore architecture, on-chip networks, and memory technologies.

John Jose received the PhD degree in CSE from the Indian Institute of Technology Madras, India, in 2014, and the MTech degree from the Vellore Institute of Technology, India. He is currently an assistant professor with the Department of CSE, Indian Institute of Technology Guwahati, India. His research interests include on-chip memory and communication aspects of multi/many core processors.