International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 / https://doi.org/10.14445/22315381/IJETT-V70I6P222

Volume 70 Issue 6, 195-208, June 2022

Original Article

ABΔZip: An Approximate Base Delta End-to-End Packet Compression Framework for Network-on-Chips

T. Pullaiah[1], K. Manjunatha Chari[2], B.L. Malleswari[3]

[1]*ECE Department, JNTUH, Hyderabad*
[2]*Department of ECE, GITAM University, Hyderabad, Telangana.*
[3]*Sri Devi Women's Engineering College, Hyderabad, Telangana.*

thanam.tp@gmail.com

Received: 03 April 2022 Revised: 23 May 2022 Accepted: 09 June 2022 Published: 29 June 2022

Abstract *- Approximate communication methods can be applied in different domains, including pattern recognition and data mining, to enhance transmission efficiency in power and delay while guaranteeing tolerable output quality. This paper proposes a new Approximate Base delta (ABΔZip) packet compression framework. This approach initially truncates the integer or floating-point data based on the error threshold for the reduction of power consumption and network latency. This model uses an approximate level configuration framework to compute the optimal error threshold based on error tolerance. After forming the approximate pattern, a BΔ compression method is used to compress the approximate pattern. Here, the BΔ compression method is modified by identifying the data patterns within a flit and is used to minimize the size of the subtractor components. Also, the proposed method uses frequent pattern compression (FPC) scheme to replace the frequent pattern with the codeword for uncompressible chunks of the delta compression method. It avoids the transmission of a small valued floating-point and integer with a larger number of Most significant bits (MSBs). The simulation results illustrate that the proposed ABΔZip increases the compression ratio to 3.7% at a 10% error threshold and minimizes the network latency, area, and power consumption to 42.1%, 4.8%, and 11.76%, respectively to the most recent existing approximate communication method.*

Keywords *- Approximate communication, network on chip (NoC), Base-delta (BΔ) compression, error tolerance, and latency.*

1. Introduction

Multicore systems integrate thousands of cores on a single chip for accomplishing evolving solicitations via lower power and parallel computations. Network-on-Chip (NoC) has been identified as a feasible candidate for efficiently managing the complex interconnection and energy in a multiple-core system [1-3]. The NoC structure is usually defined using two units: a group of 'tiles' and a network interface. The hardware component that transmits data over the NoC is named the 'tiles.' The power consumption of the NoC is increased gradually while increasing the number of cores on the integrated chip. [4, 5]. Nowadays, approximate computing applications (e.g., pattern recognition, image processing, and scientific computing) tolerate moderate errors by providing tolerable outputs [6-8]. But, the traditional NoC structures transfer all information with absolute accuracy, and this transmission is not required for approximated applications. Transmitting packets with high accuracy requires more power and intensifies the network latency.

This research work focuses on approximation communication to reduce communication overhead. The approximation communication framework is used to lower power and increase the throughput of NoC. If the endpoint node is not requiring an exact packet for calculation, the approximation communication framework gives a chance to decrease the data movement via NoC. This framework uses the approximated computations' error tolerance to enhance communication efficiency in multicore systems [9]. However, there is a requirement for a quality control method that provides approximation data for controlling the accuracy of transmitted data [10, 11]. The approximation data includes an approximation pointer and approximate level. The error-resilient data of the communication traffic can be identified by the approximation pointer, whereas the error margin to approximate the data can be specified by the approximate level [12].

The compression methods in approximate communication are usually categorized as memory and compression at interconnects. The methods utilized in approximate memory compression compress the data at various levels of the memory hierarchy (e.g., processor queues, on-chip cache) [13-15]. These methods reduced the memory traffic alone, and other data transmissions, such as data transmission between accelerators, sensors, and other peripheral units, are not approximated. Furthermore, the

memory compression approaches need considerable modification in standing IPs, including processing elements and caches, and thereby, they are not suitable for hard IP cores. The compression methods are used at the interconnects to minimize the data traffic in NoCs through the compression of analogous values in the data input or leaving all the congested data [16].

The packet compression method exploits data redundancy in the NoC packets to shrink the size of the packets and minimize the network power [17]. Hence, the approximated communication model uses the error tolerance of the application for compressing the packet. These approximate packet compression methods can reduce the power and delay performance of NoC. The NoC data can be compressed using either the Cache compression method or the network interface (NI). The Cache compression method inserts a compressor circuit between the core and NI. NI compression method adds a compressor circuit to the network interface [18-19]. The cache compression methods expand the space utility, whereas the packet compressors in NI decrease bandwidth usage, network blocking, and power consumption. The packet compression placed at the NI mostly uses a delta compression because of its easiness, less hardware complexity, low power consumption, and network latency [20-21]. The delta compression approach did not compress a packet with a smaller base. Also, they required bulkier de/compressor components to consume more power and area. This work modified the base-delta compression scheme and combined it with error tolerance-based compression to reduce the consumed power and delay of NoCs. The main contributions of this research work are listed as given below:

- To introduce an effective data compression approach for on-chip communication networks using approximate communication to enhance the delta compression method.
- To modify the base-delta compression scheme by identifying the data patterns within a flit and combining it with a truncation method to improve the compression ratio.
- To introduce a truncation method for identifying the optimal error threshold based on error tolerance. This truncation method is used to reduce power consumption and delay.
- To reduce the number of bits required for representing the difference between the base and approximated data using an intra-flit data pattern and a fixed base of size 4-bit. It improves the effect of data error tolerance on minimizing the delta size.

The rest of the paper is structured as follows: Section 2 summarizes the recent research on approximate and absolute data compression techniques for on-chip communication networks. Section 3 gives a detailed explanation of the proposed data compression method. Section 4 provides the simulation results and comparative analysis. Finally, the paper is concluded in Section 5.

2. Related Work

Some of the recent related works on the approximate communication model on NoC are summarized as follows:

Stevens et al. [22] explored an approximate compression for the communication traffic to improve the transmission bandwidth and reduce the interconnects' energy consumption. Especially an approximate bus architecture (AxBA) has been proposed for compressing/decompressing the respective communications on the bus without demanding modification for pre-planned masters and enslaved people. Here, the buses integrate devices such as processors, accelerators, peripherals, and on-chip memory. The communications on the bus are initiated by the bus masters, whereas enslaved people respond to transmission on the bus. AxBA utilized a lightweight compression approach (i.e., base-delta compression scheme) based on approximate deduplication. Base-delta compression scheme is a lossless method, and it represents the lower dynamic range of values in the data segment with a general base and sequences of deltas. Delta denotes the deviation of the data inside the block from the base. Deduplication is utilized for the elimination of duplicate copies in replicating data. Also, a software interface has been introduced in AxBA for identifying the position of the system address space, which is suitable for approximations. Furthermore, an online quality monitoring model has been proposed for detecting the error constraints automatically.

Chen et al. [23] proposed a quality control framework (QCF) for minimizing the time required to compute the approximation level in the approximate communication. This model also employed a configuration algorithm for adjusting the quality of each segment of data according to the deviation between the quality of output and application necessities. This model transferred each request packet with the modernized approximation level while implementing it in a network. In this paper, the size of the data is reduced through the truncation of the least significant bits (LSBs) of floating-point input based on approximated levels. This model reduced the number of flits in every packet and minimized congestion in NoCs without degrading the quality. When the packets need exact communication, the packets will be transmitted to the packet encoder directly. This compression unit differentiated the accurate packets from the approximable packets by referring to the data saved in the quality control table. This table contains the address field, data type, approximation level, and validity. The presence of approximate communication has been indicated by the validity bit.

Chen et al. [24] proposed an approximate transmission approach named DEC-NoC for reducing the power requirement of NoC. This approach controlled the error tolerance of the requests and reduced the quantity of error testing and adjustment in packet transmission. As a result, it reduced the number of retransmitted packets considerably. In this scheme, the amount of protected bits has been reduced by converting the integer into a floating-point format. This DEC-NoC architecture contained an approximate coding logic (ACL), demultiplexers, data type converter, encoder/ decoder, and buffers. Initially, ACL computed the approximation code using protection code and conversion code. Then, the encoder generates the check bits based on the packet's approximation code and data bits. On the decoder side, errors have been checked to utilize approximation code. Then, CIF (Converted Integer to Float) has been selected using demultiplexer for sending the data into the data type converter.

Xiao et al. [25] proposed a lossy compression method using the frequent pattern compression (FPC) that replaced the frequent data patterns with codewords. This method transferred the input data pattern into an approximate pattern using certain don't care bits before executing FPC. Also, an optimization problem has been constructed based on the quality requirement of the application. Then, a congestion-aware quality control approach was introduced, which dropped the network data insistently based on flow prediction and a lightweight heuristic for improving the system performance.

Chen et al. [26] proposed an approximated communication framework (ACF) to reduce power and delay. It used a quality control approach that identified the error-resistant variables and calculated the error thresholds according to the application's quality need by analyzing the source code. This framework also included a lightweight lossy compression method that considerably reduced packet size while transmitting the error-resilient variables. This compression logic differentiates the exact packets from the approximable packets based on the information saved in the quality control table. Here, the data approximation module transmitted a read packet that included the address of the quality control table for obtaining the information regarding approximation. When the quality control table replied with approximation details, then the type of data would be checked. Then, the compression logic used different truncation methods for the integer and floating points, respectively. Thus, the data packet contained fewer flits and guaranteed the quality necessities of applications.

The above analysis shows that a quality control method used in AxBA needs the programmer to label the values to be approximated and their error tolerances. Also, the base-delta compression scheme used by AxBA needs several bits for representing the deviation among the base and the data to be approximated. If the delta value is high, then the effect of improving data error tolerance to reduce the size of the delta is reduced. Thus, AxBA is not suitable for the more error tolerable data. Also, the QCF model did not hold an efficient compression scheme when approximating integer numbers, yielding a low data compression rate. Also, ACF considered FPC to compress the integer numbers after approximation. However, the FPC uses an encoding table that contains only eight frequently occurring patterns. Hence its performance is limited to infrequent patterns and produces more uncompressible flits.

In addition, the existing base-delta compression methods consider the base size in bytes. Hence, they didn't compress a packet whose base value is small. Furthermore, the complexity of the de/compressor units is high. A new approximate communication model is introduced using ABΔZip to tackle these issues. This method uses an efficient Quality control model to choose the level of approximation based on the software interface. Also, the BΔ compression scheme is modified using an intra-flit data pattern and a fixed base of 4-bit size to reduce the deviation between the base and approximated data. Also, the FPC scheme is combined with the BΔ compression scheme, and the metadata of the compressor is stored in the unused regions of the header flit to improve the compression ratio.

3. Proposed Approximate Communication Model For Noc

A new Approximate Base delta (ABΔZip) packet compression framework is proposed in this work. The proposed ABΔZip compression technique is employed at the Network Interface (NI) unit of a 4X4 mesh topology, as illustrated in Figure 1. Here, the tiles represent the cores with the L1 shared cache. A Network Interface (NI) is used to connect every tile and arrange them as mesh topology. In this NoC structure, the memory unit or cache accepts a read request packet if a cache miss occurs while loading a memory. The memory/cache uses this read reply packet for sending the necessary data to the core. If a cache miss is generated during the memory store process, the data is merged with the write request packet and transmitted towards the memory/cache via NoC. After receiving the data, the memory/cache sends a written reply to the core to confirm the successful memory write.

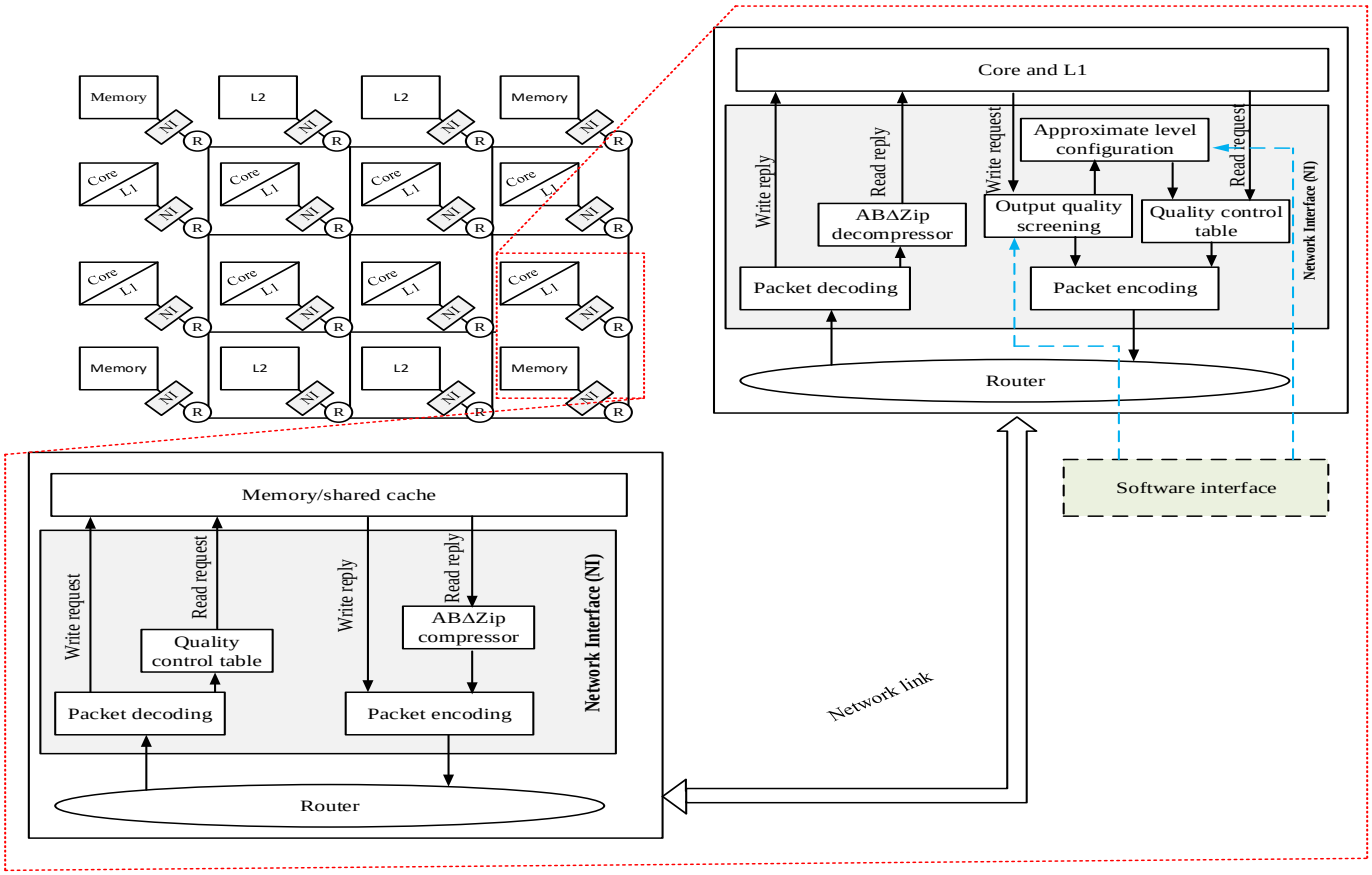


Fig. 1 Proposed NoC Architecture for approximate communication

In Figure1, the NI is reconfigured by adding the ABΔZip compression logic, the ABΔZip decompression logic, approximate level configuration framework, output quality screening unit, and a quality control table. Here, the ABΔZip de/compression logic is used by the memory/shared cache and core/L1 cache to reduce the data size according to the approximation data saved in the quality control table. This ABΔZip framework truncated and compressed data according to the approximate level, error tolerances, and data type. Here, a quality control model modulates each packet's approximate level based on the output data's error rates and application needs. The quality control table is formulated using the approximate level configuration framework and the software interface.

Here, the error-resilient variables are automatically identified by the software interface that uses the code analyzer to configure the quality control model. These error-resilient variables determine the tolerance and quality of data in the system's address space. The quality control table includes approximate data in four columns: addresses, data types, approximate level, and validity. The size of the data type column is 1 bit, and this bit is used to indicate the integer (1) or floating point (0) data. The validity bit is used to signify the enabling condition of the approximate communication. If accurate communication is needed, the validity bit is fixed to 0, which is changed to 1 for approximate communication.

After forming the approximate pattern, a base delta compression method is used to compress the approximate pattern. Here, the BΔ compression approach is modified by dividing the content of each flit into 4-bit chunks to reduce the variations among the chunks. This BΔ compression approach determines the minimum and maximum values of chunks to determine the base. Then, the compressible packets are transmitted by finding their difference from the base. ABΔZip uses the unused regions of the head flit to store the compressed packet's metadata. The data decompression logic recovers the original packet after loading the truncated bits with zeros. The packets are generated and injected into the network using the packet encoder.

Similarly, the packets are ejected from the network using a packet decoder to extract the packets' data. In addition, an output quality screening framework is used to extract the quality data that every application has produced. Subsequently, it embeds the extracted quality data into its write request traffic.

3.1. Quality control model

The quality management framework uses an approximate level configuration framework for the computation and adjusting of the approximate level of the input data. The concept of the proposed quality control model is illustrated in Figure 2. This model uses an approximate level configuration framework to adjust the input data's approximate level in the feedback path. Here, the function to be approximated is to read and process the input data while running the approximate computing application and generating the output of that function. The output quality screening framework captures the computed error rate for sending it to the approximate level configuration framework. This approximate level configuration framework utilizes an error prediction approach for adjusting the quality of the input data. The error prediction approach uses a simple moving average process for predicting the output data's error rate with the help of the seized data quality.

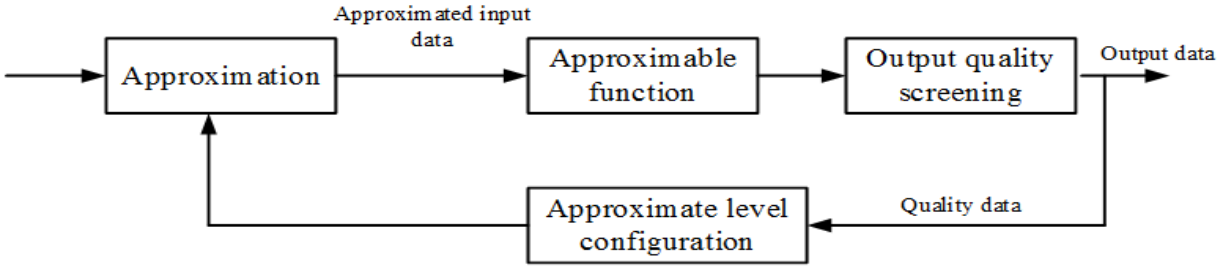


Fig. 2 Quality control model

The error rate's moving average can be calculated using the following expression:

$$M_A = preceding\ M_A + \frac{R_e(x)}{\omega} - \frac{R_e(x-\omega)}{\omega} \quad (1)$$

where $R_e(x)$ denotes the present output's error rate, ω represents the size of the window and $M_A$denotes the predicted output error rate. The number of output errors required to predict the error is described as the window size. If $x < \omega$, all the request packets will use the initial approximate level. This method determines the mean error rate using the moving window-based sampling approach. At the same time, the error rate of each output is estimated using the application itself. The approximate level configuration framework relates the value of M_AWith the quality need of an application, after predicting the error and correcting the approximate level appropriately. The quality needs to hold both input and output error thresholds. The maximum tolerable error rate in the input data is defined as the input error threshold. The necessity of the application based on the output quality is described as the output error threshold. Initially, the approximate level configuration framework matches the predicted error with the threshold of the output error. If the value of $M_A$is high compared to the output error threshold, then the input error is very high. Hence, the approximate level configuration framework decreases the input value's approximate level. The input error is very low when the output error threshold is high compared to the predicted error rate. Hence, the quality control model increases the approximate level of the packet without increasing the input error threshold.

3.2. Compression method: AB∆Zip compressor

The data approximation logic presented in the memory differentiates the accurate packets from the packets to be approximated based on the approximation data saved in the quality control table (QCT). The data approximating block transmits a read packet with the address to the QCT for getting the approximate data. When the reply message comprises approximate data, the data approximation unit identifies the input's data type (integer or floating-point). According to the error tolerance, the proposed AB∆Zip initially truncates the integer or floating-point input. The error threshold is defined as the maximum tolerable error for a packet. The error tolerance can be determined as:

$$R_e = \frac{|p-\tilde{p}|}{p} \leq T_e \quad (2)$$

Where $\tilde{p}$ denotes the approximate value of p. $R_e$and T_e Represent the relative error and error tolerance, respectively. The proposed AB∆Zip compares the error tolerance and thresholds to truncate the Least significant Bits (LSBs) while receiving the floating-point inputs. The error thresholds for the floating-point inputs are detected utilizing the IEEE 754 standard [27].

If the input is defined in the format of IEEE 754 standard, the AB∆Zip omits the first bit of the mantissa. The extreme relative error for the floating-point input can be computed while protecting q bits (of the 23-bit mantissa) as $\sum_{i=q+1}^{23} 2^{-q}$. The summation of the geometric sequence $\sum_{i=1}^{k} f\gamma^{i-1} = f(1-\gamma^k)/1-\gamma$ shows that the maximum relative error is lesser than 2^{-q}. Here, $f$and $k$denote the initial term and the number of terms, respectively. Also, the

general ratio in the sequence is denoted as γ. Hence, the data error tolerance for the floating-point data can be deduced using the following expression:

$$T_e = 2^{-k} (1 \leq k \leq 23) \quad (3)$$

Where the value of T_e is in the range of 0 and 1. Also, k is the approximate level determined by the quality control model. This k is utilized for truncating the number of MSBs from the mantissa part of the floating-point data. Hence, the relative error is smaller than 2^{-k} When $23 - k$ bits are truncated, table 1 provides the correlation of the error tolerance and approximate level with the number of LSBs to be truncated.

Table 1. Connection of Error tolerance and Approximate level of the number of LSBs to be truncated

Approximate level (k)	Error tolerance for float input (T_e)	Quantity of LSBs to be truncated
0	0	0
19	1.90735×10^{-6}	4
16	1.52588×10^{-5}	7
13	0.00012207	10
12	0.000244141	11
11	0.000488281	12
10	0.000976563	13
9	0.001953125	14
8	0.00390625	15
7	0.0078125	16
6	0.015625	17
5	0.03125	18
4	0.0625	19
3	0.125	20
2	0.25	21

The proposed AB∆Zip uses the depiction of a signed integer to compute the error tolerance for integer data. The MSB of the signed integer denotes the sign, and the rest of the 31 bits denote the value. The maximum error produced after truncating k bits (of the 31 LSBs) is detected as $\sum_{i=0}^{k} S_i 2^i$ ($S_i = 0\ or\ 1$). Hence, the error tolerance for an integer input can be computed using the following formula.

$$T_e = \frac{\sum_{i=0}^{k} S_i 2^i}{\sum_{i=0}^{31} S_i 2^i} \quad (4)$$

This truncation approach sends several MSBs for an integer with a smaller absolute value than the integer with a larger absolute value for maintaining the same error threshold as input. This issue can be tackled by the use of the B∆ compression approach. The detailed description of the proposed AB∆Zip compressor is illustrated in Figure 3.

After determining the approximate pattern, every truncated data is segmented into 4-bit chunks ($\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_i$). Here, the chunks are bounded by the largest and smallest value. Hence, the smallest $\mathcal{C}_s$and largest $\mathcal{C}_l$ Chunks are averaged to determine the base β. Here, a priority encoder is used to determine the collection of data within a flit by considering the $\mathcal{C}_s$and $\mathcal{C}_l$ as input, this 2-bit encoding is utilized to indicate whether truncated data is compressible or incompressible. When $\mathcal{C}_s$ is equal to $\mathcal{C}_l$The data can be compressed completely (100%) by producing the encoding bit as $\epsilon = 00$. The base value should be subtracted from the chunks to compute the differences (Δ_i) of all other conditions. If Δ_i is signified using a maximum of 2 bits, then the respective truncated flit is identified as compressible flit; if not, the truncated data is incompressible. Hence, when $\epsilon = 00$, the truncated data is compressible, and when $= 11$, the truncated data is incompressible. At last, the B∆ compression approach produces Base β, Encoding bits ϵ, and a set of Δ_i 's for the compressible flit and the truncated data for the uncompressible flit.

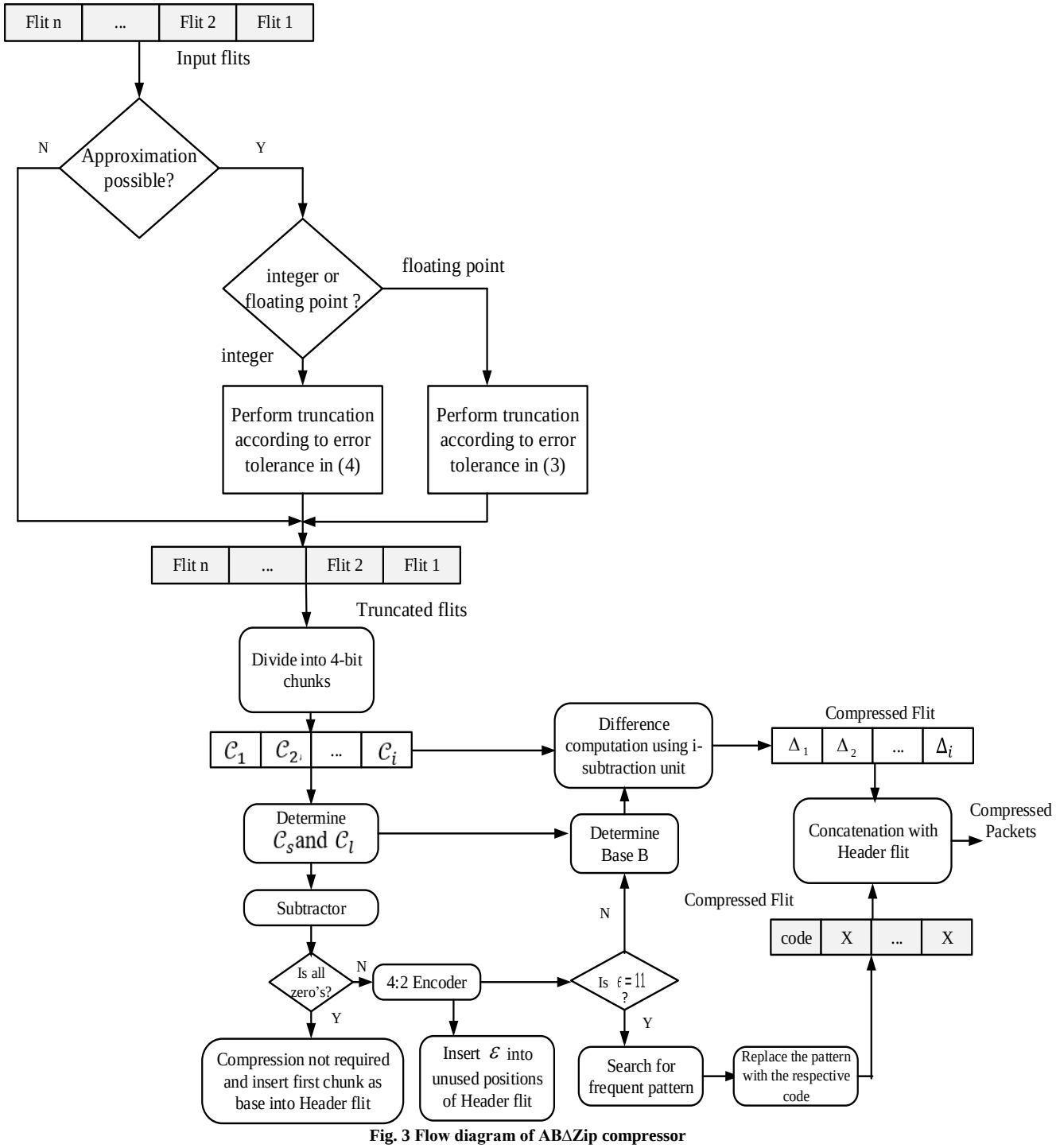


Fig. 3 Flow diagram of AB∆Zip compressor

If the flit is uncompressible, the frequent data pattern compression (FPC) approach is adopted for compressing the truncated data. The FPC approach eliminates 0's and 1's from the MSBs and LSBs from integer and floating values without degrading the accuracy. The code used for replacing the frequent pattern in the uncompressible flit is provided in Table 2. The notation X denotes any "Zero" or" one." Also, 0xFF denotes all 1-bits, and 0x00 denotes all 0-bits. The FPC compression matches the patterns with each data segment. When it identifies the matched pattern, the pattern is replaced with the respective code. This compression process does not change the 0 or 1 signified in terms of X.

Table 2. Code for frequent pattern

Code	Frequent pattern			
000	0x00	0x00	0x00	0x00
001	0x00	0x00	0x00	00000XXX
	0xFF	0xFF	0xFF	11111XXX
010	0x00	0x00	0x00	0XXXXXXX
	0xFF	0xFF	0xFF	1XXXXXXX
011	0x00	0x00	0XXXXXXX	XXXXXXXX
100	XXXXXXXX	XXXXXXXX	0x00	0x00
101	0x00	0XXXXXXX	0x00	0XXXXXXX
	0xFF	1XXXXXXX	0xFF	1XXXXXXX

The header flit for 128-bits link bandwidth contains the packet number (2-bits), source tile (6-bits), which creates the cache miss, destination tile (6-bits), Flit type (2-bits): head/body/tail, VC number where the input flit is saved (2-bits), Message Type (3-bit): REQ/REP/coherence packets and missed memory address (32-bits). In this header flit, seventy-five Least Significant Bits are unused. Hence, the proposed BΔ-NIS compressor uses these unused locations for storing the metadata. Thereby it increased the compression ratio.

3.3. ABΔZip decompressor

The destination node initially receives the head flit and extracts the metadata from the header. If the encoding bits $\epsilon = 00$, then the proposed ABΔZip decompressor unit copies the base by the number of chunks to regenerate the flits. Alternatively, the ABΔZip decompressor unit obtains Δ_iThe compressed flit is given as input to the subtractor unit along with the base value to retrieve the truncated data. Finally, NI adds zeros to the truncated portion for converting the data back into an integer. The error of the resultant integer number is within the respective error threshold. The flow diagram of the ABΔZip decompressor is shown in Figure 4.

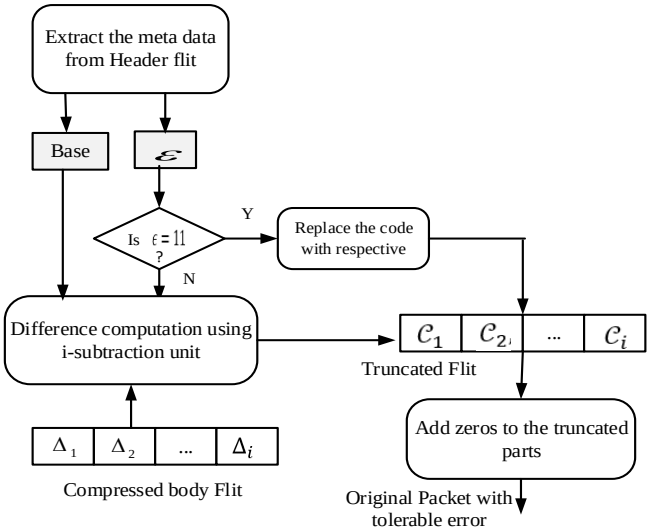


Fig. 4 Flow diagram of ABΔZip decompressor

3.4. Sample Illustration for ABΔZip De/Compressor

Figure 5 illustrates the process of ABΔZip by considering four random flits (f_1, f_2, f_3, f_4). This example contains 2 integers ($\mathcal{I}_1$ and $\mathcal{I}_2$) and 2 floating-point data ($\mathcal{F}_1$)and ($\mathcal{F}_2$). If the Approximate level of $\mathcal{F}_1$and $\mathcal{F}_2$are identified as 9 and 7, respectively, then the source node truncates 14 LSBs of $\mathcal{F}_1$ and 16 LSBs of $\mathcal{F}_2$. Similarly, the source node truncates 10 LSBs of $\mathcal{I}_1$and 1 LSB of $\mathcal{I}_2$. After detecting the approximate pattern, the truncated data is split into 4-bit chunks. Here, the truncated $\mathcal{F}_1$ contains five chunks $\mathcal{C}_1 = 1$, $\mathcal{C}_2 = 0$, $\mathcal{C}_3 = 8$, $\mathcal{C}_4 = 13$ and $\mathcal{C}_5 = 8$. From this example, it is observed that $\mathcal{C}_s = 0$ and $\mathcal{C}_l = 13$. Hence, $\beta_1 = 7$and $\Delta_1$needs 3-bits. Hence, the truncated $\mathcal{F}_1$ is it not compressible based on the BΔ compressor. Hence, the proposed ABΔZip compressor searched for frequent patterns in Table 2. The truncated $\mathcal{F}_1$ does not match with any pattern. Thereby, it can't be compressed. However, the truncated $\mathcal{F}_2$ contains $\mathcal{C}_1 = 4$, $\mathcal{C}_2 = 2$, $\mathcal{C}_3 = 3$, and $\mathcal{C}_4 = 1$. For this flit, it is observed that $\mathcal{C}_s = 1$ and $\mathcal{C}_l = 4$. Hence, $\beta_2 = 3$and $\Delta_1$needs 2-bits. Hence, the truncated $\mathcal{F}_2$ is compressible based on the BΔ compressor. Hence, the differences (Δ_i) are computed through the subtraction of every chunk from the base. Then coding bit and the base value are stored in header flits. Similarly, the truncated $\mathcal{I}_1$is compressible by a BΔ compressor. But the truncated $\mathcal{I}_2$is not compressible, so it searched for the frequent pattern in Table 2. $\mathcal{I}_2$matches with pattern 010, the proposed ABΔZip uses the respective code for replacing the zeros in the MSBs.

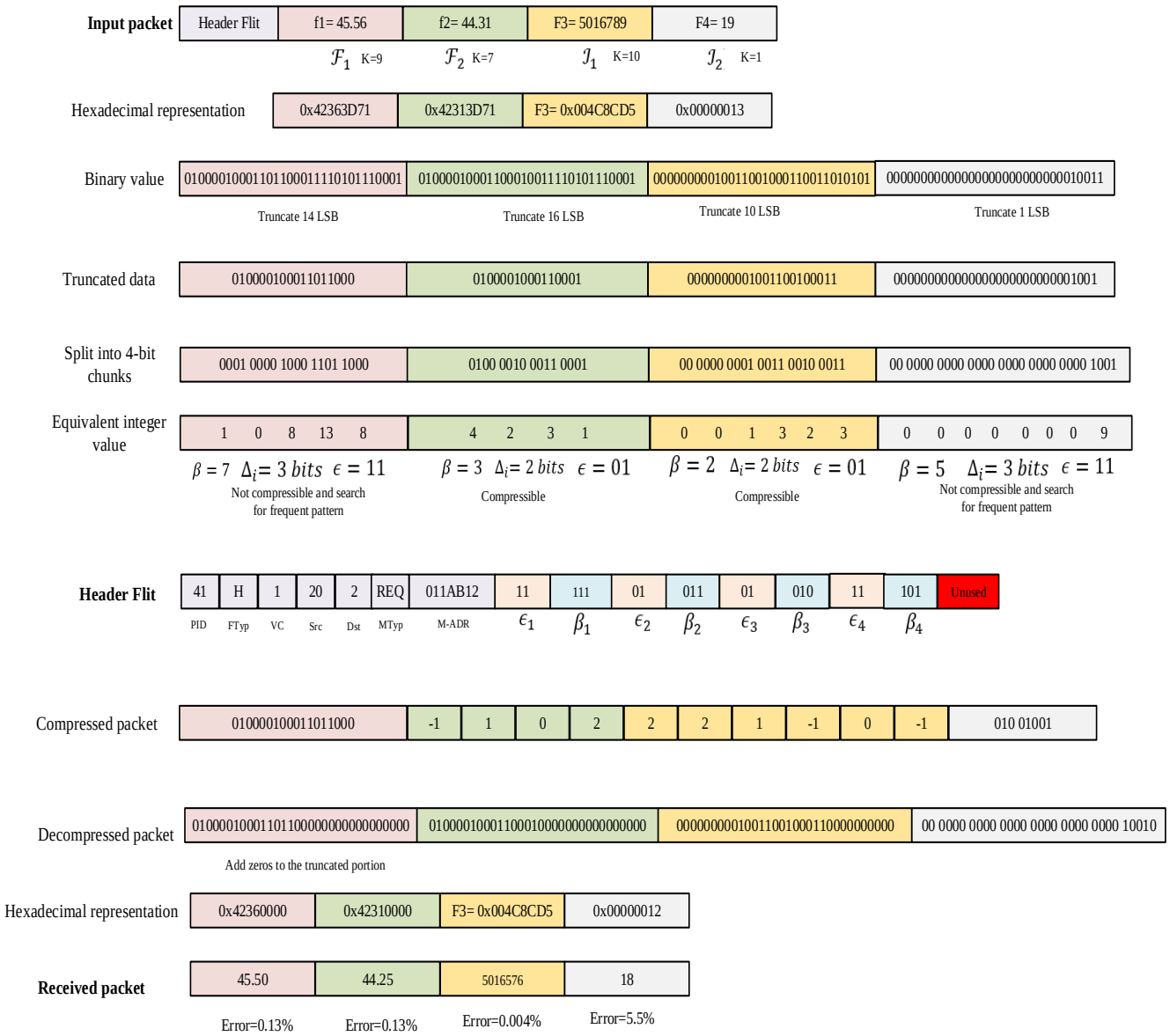


Fig. 5 Illustration for ABΔZip De/Compression

The NI placed at the destination node excerpts the metadata from the header flit. The metadata of $\mathcal{F}_1$contains $\epsilon_1 = 11$, and it indicated that $\mathcal{F}_1$is uncompressible flit. The metadata of $\mathcal{F}_2$contains $\epsilon_2 = 01$ and $\beta_2 = 011$. Hence, $\Delta_i$are subtracted with $\beta_1 = 111$for extracting the original flit. Then, the truncated bits are filled with zeros to convert the data into decimals. The same process is repeated for $\mathcal{I}_1$. It is observed that the resultant integer number has error rates of 0.13%, 0.04%, and 5.5%, which are within the respective error thresholds.

4. Simulation Results

The performance of the proposed ABΔZip De/compression unit is validated using Xilinx ISE tools, and the FPGA family chosen for synthesizing the design is Virtex 7, with the device being XC7VX550T and package FFG1927. The proposed model is evaluated in terms of compression rate, approximation, network latency, area, and power consumption. The effectiveness of the proposed model is also compared with state-of-the-art models such as AxBA [22], QCF [23], and ACF [26]. The simulated NoC Architecture contains ABΔZip De/Compressor, NI's, routers, and interconnects channels. Here, 16 routers are combined to form a 4×4 Mesh topology. Table 3 provides the device utilization summary of the proposed 4×4 Mesh NoC Architecture.

Table 3. Device utilization summary

Logic utilization	Used	Available
Number of Slice Registers	2614	692800
Number of Slice LUTs	4351	346400
Number of fully used LUT-FF pairs	2661	4426
Number of bonded IOBs	658	600
Number of BUFG/BUFGCTRL/BUFHCEs	2	272

4.1. Compression rate analysis

Initially, the compression rate is evaluated by varying the data error threshold. The higher performance data compression scheme's compression rate is usually higher than the lower performance scheme for a similar data error threshold. The variation of data compression rate against the error threshold while considering the blackscholes benchmark is illustrated in Figure 6. Here, the data error threshold varies from 1% to 10%. It is observed from Figure 6 that the compression rate of the proposed model is high as compared to all the existing models. The data compression rate of AxBA[22] is very low because the BΔ compression approach used by AxBA [22] needs several bits to represent the deviation between the base and truncated data. Also, this approach contains incompressible packets for certain data patterns. Hence, this approach's compression rate is less than the proposed method. The QCF [23] did not use an efficient data compression approach while approximating the integer numbers. Hence, it reduced the compression rate.

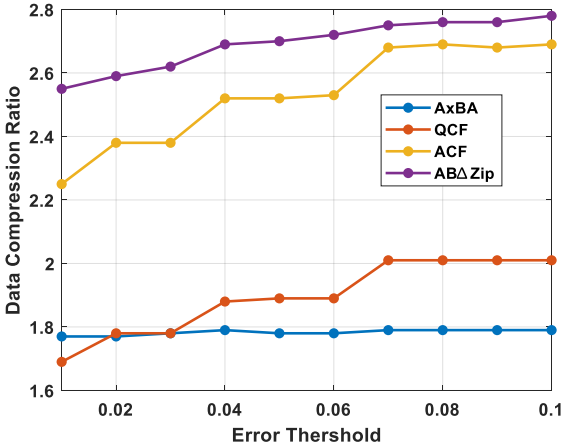


Fig. 6 Compression rate for varying error thresholds

The ACF increased the compression rate radically but used the FPC approach to reduce the integers' data size alone. Thus, the compression rate is reduced for the floating-point values. Instead, the proposed method uses the best quality control model for approximating the input packets and combines the BΔ compression scheme with the FPC approach for reducing the data size of both floating points and integers. Hence, it produces a smaller number of incompressible packets. Also, it reduced the number of bits required to represent the base's difference from the truncated data using an intra-flit data pattern and a fixed base of size 4-bit. The compression rate of the proposed approach has been further increased by storing the metadata in the unused portion of the header flit. Figure 7 provides the compression rate analysis for different benchmarks while considering the loss threshold of 2%. Here, the proposed method increased the compression rate compared to the existing methods on all benchmarks.

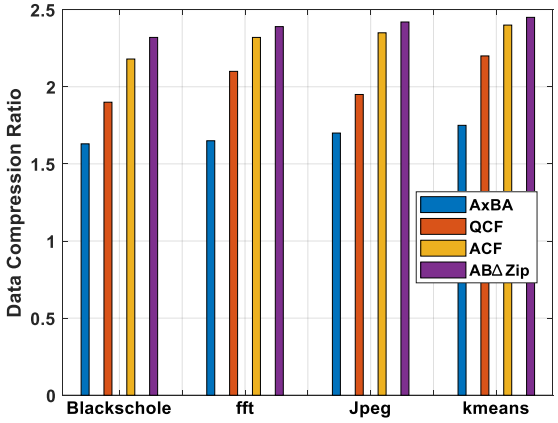


Fig. 7 Compression rate analysis for different benchmarks

4.2. Approximation analysis

The percentage of approximated data packets (ρ_{apx}) can be computed utilizing the following expression:

$$\rho_{apx} = \frac{K_{apx}}{K_T} \tag{5}$$

where K_{apx} denotes the quantity of approximated input and K_T Represents the total quantity of input. Figure 8 compares the percentage of approximated data packets of various approximation models while considering the 5% error tolerance.

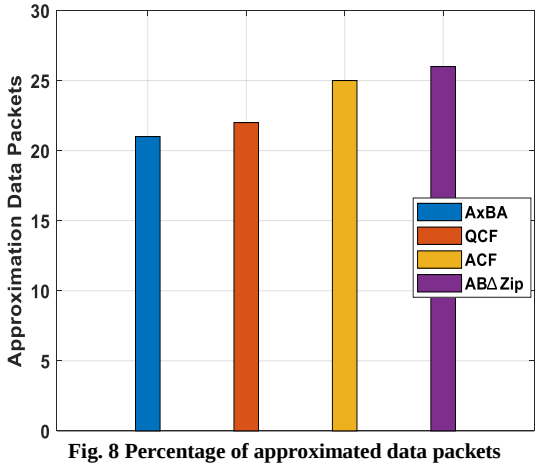


Fig. 8 Percentage of approximated data packets

The proposed model approximated several data packets compared to AxBA [22] and QCF [23]. However, the percent of approximating data packets is well-matched with the ACF due to a software interface for identifying the error-resistant variable. Hence, it does not need any humans to identify the error-resistant variable. But most of the earlier works relied on a program designer to identify the error-resistant variable. Hence, the number of approximate data packets is limited by those methods.

4.3. Output Error and quality Analysis

The application-specific measures are used to measure the output errors. Table 4 provides application-specific measures for benchmarks such as blackschole, fft, jpeg, and k means.

Table 4. Application-specific measures for benchmarks

Benchmark	Performance measure
Blackschole	Average relative error
fft	Average relative error
Jpeg	Mean pixel difference
kmeans	Mean pixel difference

The average output error of the proposed model is compared with other existing models in Figure 9 for different benchmarks. Here, the average output error shown on the y-axis denotes the quality of the output. If the average output error for the Blackschole benchmark is 5.2 %, the output quality will be 94.8%. Here, the necessary output error for an approximate computing application is indicated using the green line on the top.

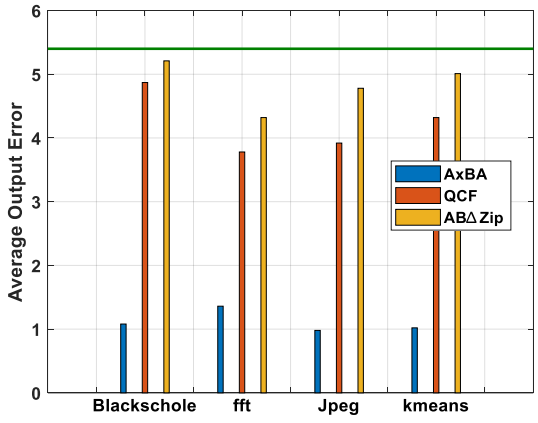


Fig. 9 Average output error

As illustrated in Figure 10, the proposed compression model ensures the output quality within the tolerance. The proposed model achieved output errors of 5.2% for the Black-Scholes benchmark. However, the output error of the AxBA model is 1.1%, which shows poor approximation performance. The approximate level of the proposed model is reduced to avoid re-execution in the presence of a substantial output error. Some applications provide fewer possibilities to the models for reducing the approximate level. The proposed model lowers the input error threshold to ensure such applications' output quality. The accumulative output error distributions of the different models are shown in Figure 9.

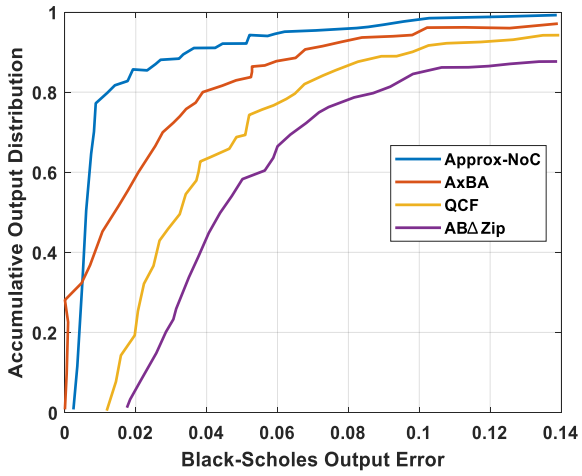
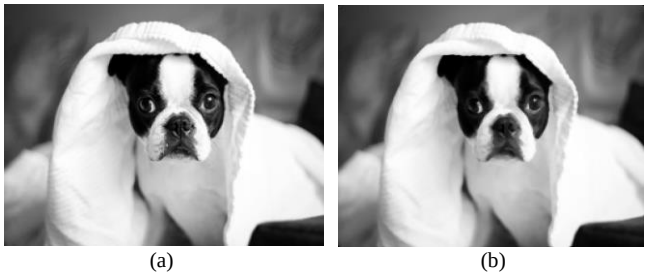


Fig. 10 Accumulative output error distributions

The proposed model monitors the output error after the completion of every iteration and regulates the approximate level of the requested data. The AxBA model depends on the quality control table to reduce the quantity of full-accuracy results. The proposed model's low accumulative output error curve specifies the larger mean output error.

To prove the quality of the proposed method, the accurate and approximate outputs of the proposed method on jpeg benchmark is shown in Figure 11. However, the deviation among these two outputs is insignificant, and the humans cant recognize the deviations.



(a) (b)

Fig. 11 Quality analysis on Jpeg benchmark (a) Accurate output (b) approximated output

4.4. Network Latency analysis

The number of clock cycles required to send the packets from the source node to the destination node is described as network latency. Hence, the network latency depends on three modules: packet initiation, packet transfer, and data

extraction. The source NI generates the packet using data compression units. Similarly, the destination NI extracts the data using packet decompression units. Figure 12 compares the network latency of the proposed method with other models. In this Figure, normalized network latency is provided for the Black-Scholes benchmark.

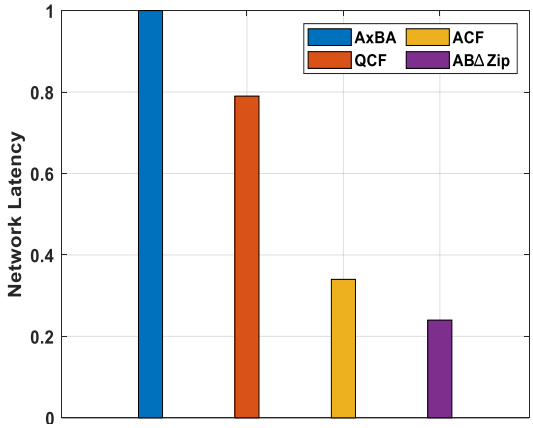


Fig. 12 Network latency

Figure 12 shows that the average network latency of the suggested model is reduced to 34.21% compared to ACF. The proposed ABΔZip model reduced the latency on the Black-Scholes benchmark because of the providence of a great percent of approximal data packets and the higher compression ratio. The proposed method combines BΔ and FPC compression to reduce the number of uncompressible flits in the approximated data. Hence, it achieved the best compression ratio, thereby attaining latency reductions.

4.5. Area and Power analysis

In this section, the overhead of the proposed method is validated in terms of area and power. The proposed model is also synthesized with 32 nm technology using Synopsys Design Vision software for evaluating the area and power. The total dynamic power can be computed by considering the dynamic power of both NI and NoC. The dynamic power consumption of the proposed model is compared with other models in Figure 13 by considering an output quality of 95%. The proposed model consumes 22.58% less power than ACF due to reducing the number of flits per packet. Table 5 compares the design overheads of every NI in terms of area and static power.

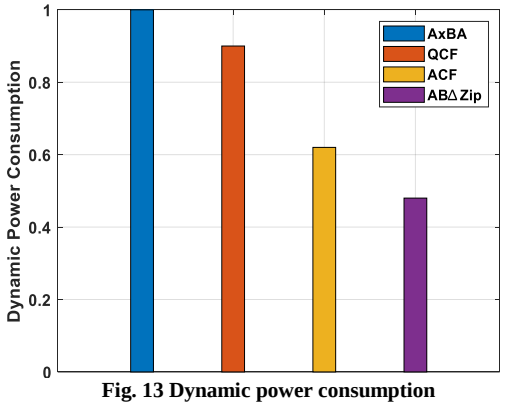


Fig. 13 Dynamic power consumption

Table 5. Comparative analysis of area and power

Models	Area (μm^2)	Power (mW)
AxBA	9.28	2.9
QCF	8.23	2.6
ACF	4.79	1.7
Proposed	4.56	1.5

It is observed that the area usage of the proposed model is 4.56 μm^2 for each NI, almost 1% of the total NoC area. The area usage of the proposed model is reduced by 44.59 %, 4.80 %, and 50.86 % compared with the models QCF [23], ACF [26], and AxBA [22], respectively. The total power consumption of the proposed model is 1.5 mW. The proposed method decreases each packet's flits without degrading the packet's quality heavily due to the consideration of an efficient quality control framework and hybrid BΔ and frequency pattern-based compression methods. Delta compression has less hardware overhead and consumes less power due to its fastest and simplest design consideration. Hence, the proposed model reduced the dynamic power consumption.

5. Conclusion

This paper proposes a new ABΔZip De/compressor unit for De/compressing packets at the NI of the NoC structure before packet injection/reception. This unit contains an approximate level configuration framework for calculating the error threshold of every packet based on the output quality and the quality necessities of the respective approximated computing application. After truncating the data packets using the approximate level, the approximated pattern has been compressed using a modified BΔ and FPC scheme. This compression framework determines the data patterns within a flit to reduce the difference between the base and the approximated data. Also, it replaced the frequent pattern with shortened code words for incompressible packets of the delta compression method.

Furthermore, it uses the unused positions of the header flits to save the metadata. Thus, the compression rate of the ABΔZip De/compressor unit is increased. It is also effective in power consumption because it consumes only 1.5 mW. The proposed method's complete validation shows the end-to-end delay reduction and area reduction by up to 34.21% and 4.80 %, correspondingly compared with the earlier best approximate communication method.

Conflict of Interest Statement

Authors T. Pullaiah, K. Manjunathachari, and B.L. Malleswari declare that they have no conflict of interest. Patients' rights and animal protection statements: This research article does not contain any studies with human or animal subjects.

Acknowledgment

I sincerely thank K. Manjunathachari and B.L. Malleswari for their guidance and encouragement in carrying out this research work.

References

[1] Jedidi, Detection and Monitoring Intra/Inter Crosstalk in Optical Network on Chip, International Journal of Electrical & Computer Engineering. 8(6) (2018) 2088-8708.
[2] S. Kashi and A. Patooghy, Row/Column-First: A Path-Based Multicast Algorithm for 2D Mesh-based Network on Chips, Iranian Journal of Electrical and Electronic Engineering. 14(2) (2018) 124-136.
[3] S.S. Kendaganna, A. Jatti and B.V. Uma, Design and Implementation of Secured Agent Based Noc Using Shortest Path Routing Algorithm, International Journal of Electrical and Computer Engineering. 9(2) (2019) 950.
[4] M.F. Reza & P. Ampadu, Approximate Communication Strategies for Energy-Efficient and High Performance Noc: Opportunities and Challenges, In Proceedings of the on Great Lakes Symposium on VLSI. (2019) 399-404.
[5] K. Zhou, Y. He, R. Xiao, J. Liu & K. Huang, A Customized NoC Architecture to Enable Highly Localized Computing-On-the-Move DNN Dataflow, IEEE Transactions on Circuits and Systems II: Express Briefs. (2021).
[6] Y. Wang, H. Li, Y. Han & X. Li, A Low Overhead In-Network Data Compressor for the Memory Hierarchy of Chip Multiprocessors, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37(6) (2018) 1265–1277.
[7] S. Kim & Y. Kim, Novel XNOR-based Approximate Computing for Energy-Efficient Image Processors, Journal of Semiconductor Technology and Science. 18(5) (2018) 602-608.
[8] Y. He, X. Yi, Z. Zhang, B. Ma & Q. Li, A Probabilistic Prediction-Based Fixed-Width Booth Multiplier for Approximate Computing, IEEE Transactions on Circuits and Systems I: Regular Papers. 67(12) (2020) 4794-4803.
[9] F. Betzel, K. Khatamifard, H. Suresh, D.J. Lilja, J. Sartori & U. Karpuzcu, Approximate Communication: Techniques for Reducing Communication Bottlenecks in Large-Scale Parallel Systems, ACM Comput, Surveys. 51(1) (2018) 1–32.
[10] S. Xiao, X. Wang, M. Palesi, A.K. Singh & T. Mak, ACDC: An Accuracy- And Congestion-Aware Dynamic Traffic Control Method for Networks-on-Chip, In Proc. IEEE Design Autom. Test Eur. Conf. Exhibit, (DATE), Florence, Italy. (2019) 630–633.
[11] R. Fenster & S. Le Beux, RELAX: A Reconfigurable Approximate Network-on-Chip, In IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), IEEE. (2021) 381-387.
[12] Y. Chen & A. Louri, Learning-Based Quality Management for Approximate Communication in Network-on-Chips, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 39(11) (2020) 3724-3735.
[13] A. Jain, P. Hill, S.C. Lin, M. Khan, M.E. Haque, M.A. Laurenzano & J. Mars, Concise Loads and Stores: The Case for an Asymmetric Compute-Memory Architecture for Approximation, In Proc. of MICRO. (2016).
[14] A. Ranjan, A. Raha, V. Raghunathan & A. Raghunathan, Approximate Memory Compression for Energy-Efficiency, In Proc. of ISLPED. (2017).
[15] J.S. Miguel, J. Albericio, A. Moshovos & N.E. Jerger, Doppelgänger: A Cache for Approximate Computing, In Proc. of MICRO. (2015).
[16] R. Boyapati, J. Huang, P. Majumder, K.H. Yum & E.J. Kim, APPROXNoC: A Data Approximation Framework for Network-On-Chip Architectures, In Proc. of ISCA. (2017).
[17] Q. Wang, Y. Li & P. Li, Liquid State Machine Based Pattern Recognition on FPGA with Firing-Activity Dependent Power Gating and Approximate Computing, In 2016 IEEE International Symposium on Circuits and Systems ISCAS, IEEE. (2016) 361-364.
[18] J.S. Kim, J.B. Hong, J.Y. Kang & T.H. Han, Lifetime Improvement Method Using Threshold-Based Partial Data Compression in Noc, In 2018 International SoC Design Conference ISOCC, IEEE. (2018) 269-270.
[19] N. Niwa, Y. Shikama, H. Amano & M. Koibuchi, A Case for Low-Latency Network-on-Chip Using Compression Routers, In 2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), IEEE. (2021) 134-142.
[20] J. Zhan, M. Poremba, Y. Xu & Y. Xie, NoD: Leveraging Delta Compression for End-to-End Memory Access in Noc Based Multicores, in Proc. 19th Asia South Pacific Des. Automat. Conf. (2014) 586–591
[21] Y. Zhang, Y. Yuan, D. Feng, C. Wang, X. Wu, L. Yan & S. Wang, Improving Restore Performance for In-Line Backup System Combining Deduplication and Delta Compression, IEEE Trans. Parallel Distrib. Syst. 31(10) (2020) 2302–2314.
[22] J.R. Stevens, A. Ranjan, & A. Raghunathan, AxBA: An Approximate Bus Architecture Framework, In Proceedings of the International Conference on Computer-Aided Design. (2018) 1-8.
[23] Y. Chen & A. Louri, An Online Quality Management Framework for Approximate Communication in Network-On-Chips, In Proceedings of the ACM International Conference on Supercomputing. (2019) 217-226.

[24] Y. Chen, M.F. Reza & A. Louri, DEC-NoC: An Approximate Framework Based on Dynamic Error Control with Applications to Energy-Efficient NoCs, In 2018 IEEE 36th International Conference on Computer Design ICCD, IEEE. (2018) 480-487.
[25] S. Xiao, X. Wang, M. Palesi, A.K. Singh, L. Wang, & T. Mak, On Performance Optimization and Quality Control for Approximate-Communication-Enabled Networks-on-Chip, IEEE Transactions on Computers. 70(11) (2020) 1817-1830.
[26] Y. Chen & A. Louri, An Approximate Communication Framework for Network-on-Chips, IEEE Transactions on Parallel and Distributed Systems. 31(6) (2020) 1434-1446.
[27] IEEE Standards Committee et al., IEEE Standard for Floating-Point Arithmetic, IEEE Computer Society Standard. 517 (2008).
[28] Krutthika H.K, A R Aswatha, Design of Efficient FSM Based 3D Network on Chip Architecture, International Journal of Engineering Trends and Technology. 68(10) (2020):67-73.