2018 IEEE 36th International Conference on Computer Design

DEC-NoC: An Approximate Framework based on Dynamic Error Control with Applications to Energy-efficient NoCs

Yuechen Chen, Md Farhadur Reza, Ahmed Louri
Department of Electrical and Computer Engineering
The George Washington University
Washington D.C.
Email: {yuechen, farhadur_reza, louri}@gwu.edu

***Abstract*—Network-on-Chips (NoCs) have emerged as the standard on-chip communication fabrics for multi/many core systems and system on chips. However, as the number of cores on chip increases, so does power consumption. Recent studies have shown that NoC power consumption can reach up to 40% of the overall chip power [1]–[3]. Considerable research efforts have been deployed to significantly reduce NoC power consumption. In this paper, we build on approximate computing techniques and propose an approximate communication methodology called DEC-NoC for reducing NoC power consumption. The proposed DEC-NoC leverages applications' error tolerance and dynamically reduces the amount of error checking and correction in packet transmission, which results in a significant reduction in the number of retransmitted packets. The reduction in packet retransmission results in reduced power consumption. Our cycle accurate simulation using PARSEC benchmark suites shows that DEC-NoC achieves up to 56% latency reduction and up to 58% dynamic power reduction compared to NoC architectures with conventional error control techniques.**

***Index Terms*—Approximate Communication, Error Control, Energy Consumption, Networks-on-Chip (NoCs)**

I. Introduction

Networks-on-chips (NoCs) are becoming standard communication solutions for connecting multiple processing cores on chip [4]–[6]. To mitigate communication errors in NoCs, error correction techniques such as error control codes (ECCs) and automatic repeat query (ARQ) are commonly used. Such techniques rely on redundant bits and retransmission to check and correct errors. Previous research [7], [8] shows that both ECC and retransmission techniques incur high power consumption and high latency with an increase in on-chip communication traffic.

Recent research has shown that several approximate computing applications, such as pattern-recognition, image processing, and scientific-computing, can tolerate errors while yielding acceptable approximate results [9]–[13]. We observe that error resilient applications can tolerate errors and therefore do not require full data accuracy. This implies that current error control techniques, which provide full error coverage, impose expensive and excessive data protection for these applications. This inspired us to re-examine NoC error control techniques for these applications, with the aim of reducing communication power consumption. The proposed framework, DEC-NoC, consists of a dynamic error control scheme and its hardware support. DEC-NoC can dynamically adjust the amount of data protection (i.e the number of bits to be protected by error control hardware) based on the application's error tolerance. We show that DEC-NoC decreases the amount of retransmitted packets, which results in significant improvement in power consumption and end-to-end latency. Specifically the contributions of this work include:

- An approximate communication technique, which dynamically configures the error correction length of a data packet by utilizing the relaxed accuracy constraints of a given application.
- A hardware implementation for implementing the proposed technique in NoCs.
- Performance evaluation of the proposed technique showing the reduction of end-to-end latency by up to 56% and power reduction by up to 58% compared to conventional error control techniques, such as ARQ with cyclic redundancy check(CRC) and ARQ with single error correction and double error detection (SECDED).

II. Motivation and Challenges

A. Motivation

Time-consuming and costly communication. Increasing data movement of emerging big-data applications results in heavy NoC communication loads. Communication takes relatively more time for running tasks (compared to computation) of parallel applications with the increase of the number of compute elements (processing core) in the systems [14], [15]. Communication is also more costly compared to computation in terms of energy consumption [14]–[16]. Error control techniques are often deployed in NoCs to mitigate all kinds of communication errors. However, the error control techniques introduce more communication overhead and higher power consumption as were shown in [7], [8]. It is therefore imperative to revisit these techniques for error tolerant applications using approximation.

Error tolerance of applications. Several applications have been identified to be tolerant to inaccuracies and therefore

2576-6996/18/$31.00 ©2018 IEEE
DOI 10.1109/ICCD.2018.00078

do not need exact data for computation. These applications include recognition, mining and synthesis [12], [13], [17], [18]. The data error tolerance or threshold is defined as the maximum acceptable accuracy loss of a value. For example, if the value of 100 has the data error threshold of 10%, the computation can accept the value within the range from 90 to 110. We exploit this error resiliency to reduce the overhead of error control techniques. Therefore, communication data can be approximated for performance improvement and energy efficiency.

B. Challenges

Maintaining output quality. Controlling errors while approximating is required to ensure output quality. To ensure the quality of output, previous research proposed an EnerJ framework [19] which can be used by programmers to annotate approximable sections of the data in an application. Other research [20] have proposed to differentiate overall output quality and individual data errors, as overall output quality varies with the change in the approximation error of the individual elements. This implies that an approximate communication technique should be able to control error rate individually in each packet and across the whole application execution.

Low overhead of approximation. An approximation communication technique requires approximation logic (including hardware support) to control errors for guaranteed output quality. Approximation logic incurs overhead in terms of area, latency, and energy. The implementation needs to be carefully designed so that overheads of approximation do not exceed the energy and performance gain from approximate communication.

III. DEC-NoC FRAMEWORK

The essence of our proposed approximate communication framework, DEC-NoC, is to carefully use the application error threshold and reduce the amount of retransmissions by limiting the number of bits checking at the source and destination. Reduction in the number of bits checking (error correction length) decreases the chance of retransmission, which will result in decreasing in overhead communication traffic in NoC. The decrease in communication overhead eventually translates into a reduction of NoC power consumption. In this work, we focus on approximating data packets, containing both integer and floating-point values. We assume the error threshold requirement is provided by the application and is readily available at the communication protocol level. We initially analyze error correction length for floating point and integer values based on the data error threshold in Sec. III-A. In Sec. III-B, we discuss the data-type conversion from integer to floating point to reduce the error correction length based on the data precision. Finally, we give an example on the operation of DEC-NoC technique in Sec. III-C.

A. Error Correction Length Calculation for Floating Point and Integer Data

Eqs. 1 and 2 show the representation of single precision floating point value based on IEEE 754 standard [21].

$$float = (-1)^S \times mantissa \times 2^{exp} \tag{1}$$

$$mantissa = 2^0 + \sum_{k=1}^{23} X_k 2^{-k} \ (X_k = 0 \ or \ 1) \tag{2}$$

Based on the Eqs. 1 and 2, the mantissa always starts with one. According to the IEEE 754 standard [21], when the data is represented in the floating point data format, the first bit of the mantissa is omitted. We observe that when c bits (of the 23-bits mantissa) are protected, the maximum error of this floating point data will be $\sum_{k=c+1}^{23} 2^{-k}$, which is less than 2^{-c} according to the summation of geometric progression ($\sum_{k=1}^{n} a * r^{k-1} = a(1-r^n)/1-r$, where a is the first term, n is the number of terms, and r is the common ratio in the sequence). Therefore, using Eq. 2, we can deduce an expression for data error threshold as follows:

$$Error \ threshold = 2^{-n} \ (1 \leq n \leq 23) \tag{3}$$

In the Eq. 3 above, data error threshold is a number between 0 to 1, and n is the number of protected mantissa in this floating point value. In the floating point data, 1-bit sign and 8-bits exponent (total 9-bits) are the critical information, which requires protection. Thus, by protecting $n + 9$ bits, we can ensure 2^{-n} data error threshold. For example, to achieve 10% data error threshold for any floating point value, we only need to protect 13 most significant bits (MSBs), which yields maximum data error of 6.25%.

Eq. 4 shows the representation of a signed integer. In a signed integer, MSB bit represents the sign, and the rest of the 31 bits represents the value.

$$int = \sum_{k=0}^{30} X_k 2^k \ (X_k = 0 \ or \ 1) \tag{4}$$

We observe that an integer with small absolute value requires more protected bits to get the same data error threshold compared to the integer with large absolute value. For example, 31 MSBs need to be protected for integer value 10 to achieve 10% data error threshold. On the other hand, 27 MSBs need to be protected for integer value 1000 to achieve the same data error threshold (10%).

B. Integer to Floating Point Conversion

Based on our observation, integer (especially for small values) requires more bit protection than floating point. We therefore convert integers to floating point representation to reduce the number of protected bits. However, as integer has higher precision than floating point, directly converting an integer with large absolute value to floating point value will cause accuracy loss. Therefore, based on Eq. 4, we define the conversion value range (CVR) between -2^{24} and 2^{24}, where, in this range, an integer can be converted to a floating point

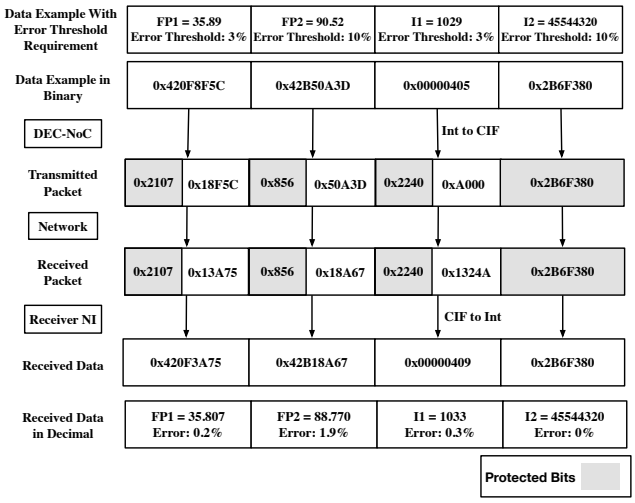

Fig. 1. **DEC-NoC technique example**

value without accuracy loss, as an integer requires less than 24 bits to represent an accurate value. If an integer is within the CVR, DEC-NoC converts the integer to floating point, and that floating point data is defined as CIF (Converted Integer to Float). DEC-NoC follows Eq. 3 to calculate the number of protected bits for CIF. If an integer exceeds the CVR, the original data is fully protected during communication. At the destination, DEC-NoC converts the CIF back to integer to maintain original data type. As integer has higher precision than floating point, there will be no accuracy loss when a CIF is converted back to an integer at the destination. As a result, DEC-NoC ensures data error thresholds for both floating point and integer values, and achieves the minimum amount of protected data bits (error correction length).

C. Working Example of DEC-NoC Technique

Fig. 1 shows DEC-NoC working example of two floating point numbers (FP1 and FP2) and two integers (I1 and I2). In this example, DEC-NoC protects 15 MSBs (of FP1) and 13 MSBs (of FP2) at the source node to ensure the floating point values meet the corresponding error thresholds of 3% and 10%, respectively. When the floating point numbers reach the receiver network interface (NI), we observe that the error rates of 0.2% and 1.9% of the approximated floating point data are within the error thresholds of 3% and 10%, respectively. In the case of integer numbers, DEC-NoC only converts integer I1, which is within the CVR. For integer I2 that exceeds the CVR, DEC-NoC protects the full data to satisfy accuracy requirement. When the approximated integer reaches the destination, the receiver NI converts the CIF data back to integer, and we observe that the errors 0.3% and 0% of the integer numbers are also within the corresponding error thresholds.

IV. IMPLEMENTATION OF DEC-NOC FRAMEWORK

A baseline network interface with ECC (shown in Fig. 2) in a multi-core architecture contains packet encoder, decoder,

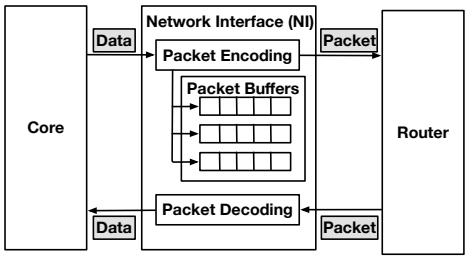

Fig. 2. **Baseline Architecture**: Packet encoder applies ECC to the whole packet. Packet decoder checks bits errors using check bits. Packet buffer stores transmitted packet to satisfy retransmission requirement.

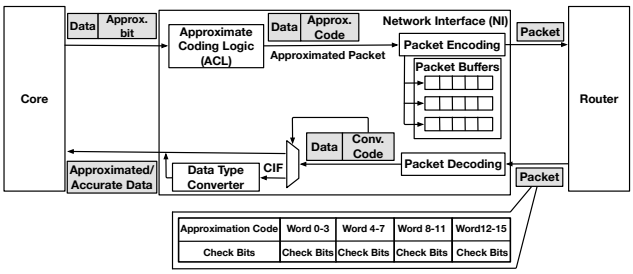

Fig. 3. **DEC-NoC Architecture**: Approximate coding logic (ACL) differentiates approximate and exact transmission based on the data type and approximation (Approx.) bit, and then calculates approximation (Approx.) code based on error threshold. Packet encoding module applies ECC based on the approximation code to protect bits. Packet decoding module checks for errors in the protected bits. Data type converter converts the CIF (Converted Integer to Float) back to integer based on conversion (Conv.) code.

and packet buffers [22]. The encoder generates check bits for the whole data packet. The decoder checks the data for errors using check bits. The packet buffers are used as a storage for the transmitted packets to satisfy retransmission requirements. When a packet is sent, it is usually stored in a buffer until an error-free acknowledgement is received. Should there be errors in transmission, the packet is retransmitted in the following cycles.

The proposed DEC-NoC architecture is depicted in Fig. 3. We modified the baseline NI and included the following additional components: an approximate coding logic (ACL), demultiplexers, data type converter, along with the encoder/ decoder and buffers. The approximate indicator (Approx. bit) indicates accurate and approximable data based on the data annotations in the application [19]. The ACL consists of approximation (Approx.) code calculation and data-type conversion. The ACL calculates approximation code, which contains protection code (indicates the number of MSBs to be protected) and conversion code (indicates whether the data is CIF or not). The encoder generates check bits based on both the approximation code and data bits in the packet. The decoder checks the data for errors using approximation code. The demultiplexer selects CIF based on conversion code, and sends data to the data type converter for conversion.

The proposed approximate communication work flow is discussed in Sec. IV-A. Packet encoding and decoding module to dynamically adjust error correction length is discussed in Sec. IV-B.

TABLE I
RELATIONSHIP AMONG ERROR THRESHOLD, ERROR CORRECTION LENGTH, AND PROTECTION CODE

Error Threshold	Error Correction Length	Protection Code
0.125	12	000
0.0625	13	001
0.015625	15	010
0.001953125	18	011
0.00012207	22	100
1.52588E-05	25	101
1.90735E-06	28	110
0	32	111

A. DEC-NoC Work Flow

Fig. 4 shows the operation flowchart describing the functionality of the ACL. ACL distinguishes approximable packet from accurate packet depending on the annotations in the application. If the packet requires accurate transmission, ACL directly sends the packet to be encoded for full protection. If the packet can be approximated, ACL calculates protection code by looking up the error threshold in Table I, which is derived from Eq. 3. If the error threshold of the packet is between two error threshold numbers in Table I, the ACL selects the lower error threshold level to ensure data quality. For example, if the data error threshold is 7%, 6.25% error threshold is selected to guarantee the data quality, and protection code 001 is generated.

The ACL also generates conversion code to identify the data type conversion at the source. If the approximable packet contains floating point value, the conversion code is set to 0, as there will be no data type conversion. If the approximable packet is an integer value, the ACL checks whether the integer value is within the CVR. If the value is within the CVR, the ACL converts the integer to CIF and sets the conversion code to 1, and then looks up the protection code using Table I. If the integer value exceeds CVR, the ACL sets conversion code to 0 and protection code to 111, which indicates that all the data bits require protection.

Using the example described in Fig. 1, the approximation code for FP1 is 0010. In this approximation code, the highest bit is a conversion code (0), and the lowest 3 bits are a protection code (010). The approximation code for FP2 is 0001 (conversion code = 0, protection code = 001). For approximable integer I1, which is within the CVR, the approximation code is 1010. For approximable integer I2 that exceeds the CVR, the approximation code is 0111.

After the approximation code is generated, the packet with the approximation code is sent to the packet encoder for ECC encoding (which is described in the next section).

B. Packet Encoding and Decoding

1) Packet Encoding: The packet encoder generates approximate packet, which contains the approximation code (for all the words in the packet) and check bits for each flit (head/body/tail flit). The check bits are generated based on the flit type and protection code for each word. The work flow of

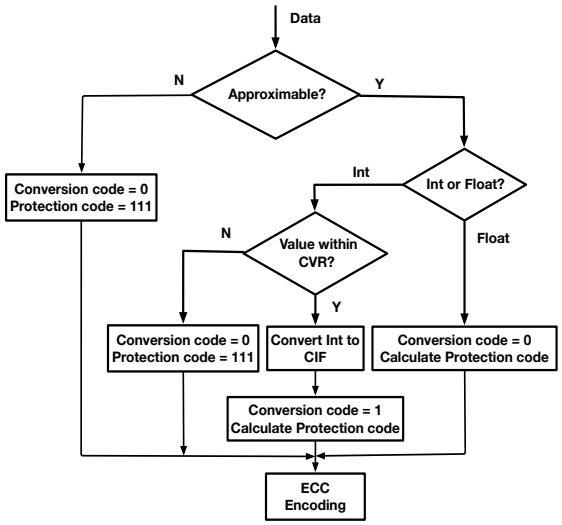

Fig. 4. **Approximate Coding Logic (ACL) Operation Flow Chart**

Algorithm 1: Packet Encoding

```
for all the flit in the packet do
    if flit_type = HEAD then
        Calculate ECC for all bits
    if flit_type = BODY or TAIL then
        for all words in the flit do
            Select the MSBs based on
              ProtectionCode[word]
        Calculate ECC for selected bits
```

the packet encoding is described in algorithm 1. If the flit type is a head flit, the packet encoder applies full protection to the flit.

For the rest of the flits in the packet, the packet encoder applies ECC based on the protection code of each word. To achieve this, the packet encoder selects the number of bits of a flit based on the protection code. After that, the packet encoder calculates check bits based on the selected bits only.

As an example, we can assume four values (one row) in Fig. 1, as a body/tail flit. Based on protection codes, the protected bits of the values in the flit are 15, 13, 15, and 32 MSBs. The packet encoder generates check bits for only the protected bits using ECC coding algorithm.

2) Packet Decoding: The packet decoder checks for errors in the protected bits based on the protection code. The workflow of the packet decoding is described in algorithm 2. In the algorithm, the packet decoder checks the head flit for errors. If the flit contains an error, the router will retransmit the filt. Otherwise, the decoder reads approximation code from the head flit and checks for data integrity of the protected bits using CRC or SECDED based on the protection code. In the case of CRC, if the protected bits do not pass the error checking, the receiver sends a NACK signal back to the sender. In the case of SECDED, error correction is applied at the receiver. If the error correction fails, the receiver sends a

Algorithm 2: Packet Decoding

```
if flit_type = HEAD and flit has error then
    Ask router to retransmit the flit
if flit_type = HEAD and flit is error-free then
    C[word] ← Approximation Code[word]
    for all the word in this packet do
        Select and check data integrity based on
          C[word].Protection Code
        if word has error then
            Ask sender for retransmission
        else if C[word].Conversion Code = 1 then
            Notify sender a successful transmission
            Send data to data type converter
        else
            Notify sender a successful transmission
            Send data to core
```

NACK signal back to the sender, and the sender retransmits the corresponding packet from the packet buffer. If the protected bits pass the error checking (CRC) or correction (SECDED), the decoder sends an ACK signal to the sender to confirm successful transmission. Following that, the destination NI checks for CIF value, and converts CIF back to integer based on conversion code. Finally, the destination NI sends the approximated data to the core.

V. EXPERIMENTAL SETUP

In this section, we describe the simulation setup and benchmarks used.

A. Simulation Setup

We evaluate DEC-NoC using BookSim2 [23] and Netrace [24] simulators, where Netrace is integrated with BookSim2. The Netrace simulator is used to capture and inject cycle accurate benchmark traces for BookSim2. We have also modified the BookSim2 simulator to incorporate an error injection model. We integrated an error injection model from [25] to mimic realistic production of per-link bit error rate, ϵ. We set the bit error rate to 10^{-4} similar to what was used in [7], [8]. The error rate for each flit is calculated for both CRC and SECDED error control coding techniques. Since CRC only checks for errors in the flit, and the occurrence of an error in each transmitted bit is an independent event, the probability P_{flit} of a n-bit flit containing soft error can be calculated for each link using Eq. 5.

$$P_{flit} = 1 - (1 - \epsilon)^n \tag{5}$$

Because of the capability of single-bit error correction using SECDED, the transmitted flit is considered to be a fault flit only if it contains more than 2 errors. Therefore, the flit error rate when SECDED is deployed can be calculated using Eq. 6.

$$P'_{flit} = 1 - (1 - \epsilon)^n - n \times \epsilon \times (1 - \epsilon)^{n-1} \tag{6}$$

To evaluate the performance of the proposed design, we compare the performance of DEC-NoC to a baseline NoC with

TABLE II
SIMULATION SETUP

NoC Parameters	8 × 8 2D Mesh 8 Virtual Channel Wormhole Switching X-Y Routing
System Parameter	64 on-chip Cores at 2GHz 32KB L1 Instruction Cache 32KB L1 Data Cache 4-way associative 64 bank fully shared, 16 MB L2 Cache
Error threshold	15%, 10%, 5%
Integer to Floating point Conversion Ratio	25%, 50%, 75% (Default)
Error Control technique	ARQ+CRC, ARQ+SECDED

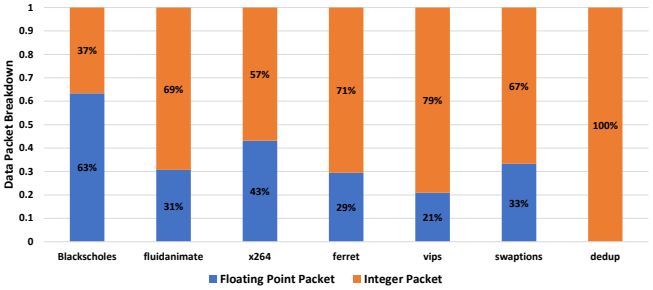

Fig. 5. **Integer and Floating Point Data Packet Percentages in PARSEC Benchmarks**

static full-protection error control techniques using ARQ+CRC and ARQ+SECDED. We evaluate three different error threshold levels 15%, 10%, and 5% in our experiments, where we set the rate of integer to floating point conversion to be 75% (default). Moreover, since the different rate of integer to floating point conversion will affect the number of protected bits in a packet, we conduct sensitivity study with three different rates of conversion: 75%, 50%, and 25%. Detailed simulation setup is shown in Table II.

B. Workload Analysis

We use PARSEC benchmark [26], with simmedium, as the workload for NoC performance simulation [27]. We evaluate the dynamic power breakdown for both ARQ+CRC and ARQ+SECDED error control techniques. It can be seen from Fig. 6, for ARQ+CRC, the dynamic power consumption of retransmission traffic can be as high as 90% (on average). Even with the capability of single error correction in SECDED, the retransmission dynamic power consumption for

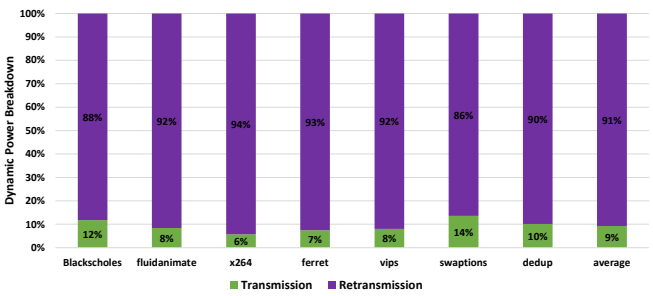

Fig. 6. **Dynamic power consumption percentages of transmitted and retransmitted packets for ARQ+CRC under PARSEC Benchmarks**

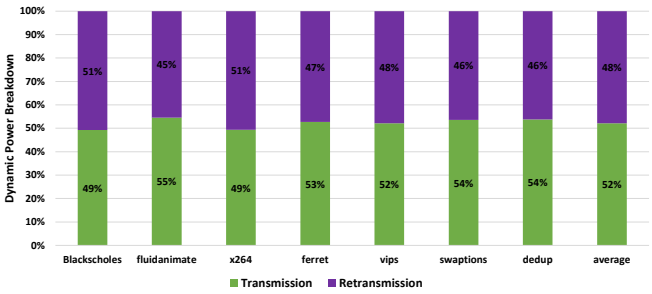

Fig. 7. **Dynamic power consumption percentages of transmitted and retransmitted packets for ARQ+SECDED under PARSEC Benchmarks**

ARQ+SECDED is still around 47%, as shown in Fig. 7. As retransmission has high power consumption, it is crucial to reduce retransmission traffic using approximate communication.

We obtained the percentage of integer and floating point data packets for each benchmark of PARSEC from [26], as shown in Fig. 5. We observe that the blackscholes benchmark has the highest percentage (63%) of floating point data packet compared to other benchmarks, while the dedup benchmark has no (0%) floating point data packet. We conduct sensitivity study in Sec. VI-C to see the impact of integer and floating point packets on the performance.

VI. EVALUATION AND ANALYSIS

We evaluate DEC-NoC in terms of end-to-end latency, power consumption, and quality of output using PARSEC benchmarks. We also discuss the overhead of our design.

A. End-to-End Latency

We evaluate average end-to-end latency, under the simulation setup shown in Table II. End-to-end latency is defined as the number of clock cycles taken from the packet injection at the source node to the successful delivery of packet at the destination. We compare our proposed design with baseline error control techniques. The normalized test results of end-to-end latency are shown in Figs. 8 and 9. As can be seen in Fig. 8, DEC-NoC achieves an average end-to-end latency reduction of 56%, 55%, and 52% over static ARQ+CRC baseline across all benchmarks, for error thresholds of 5%, 10%, and 15%, respectively. Fig. 9 shows that compared to ARQ+SECDED, DEC-NoC achieves end-to-end latency reduction of 28%, 28%, and 27% for the above mentioned error thresholds.

The largest end-to-end latency reduction in both experiments is achieved for blackscholes benchmark, while the least end-to-end latency improvement is obtained for dedup benchmark. Latency improvement is higher for benchmarks with a higher percentage of floating point packets, as DEC-NoC significantly reduces protection for floating point packets.

B. Power Consumption

Since DEC-NoC reduces retransmission traffic, we analyze overall dynamic power consumption. Static power remains unchanged in the proposed framework. Figs. 10 and 11 show the evaluation results, which are normalized to the baseline

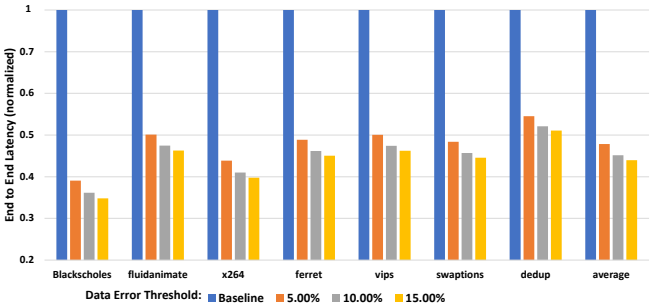

Fig. 8. **End-to-End latency of ARQ+CRC error control technique under different error threshold configurations**: The results are normalized to baseline.

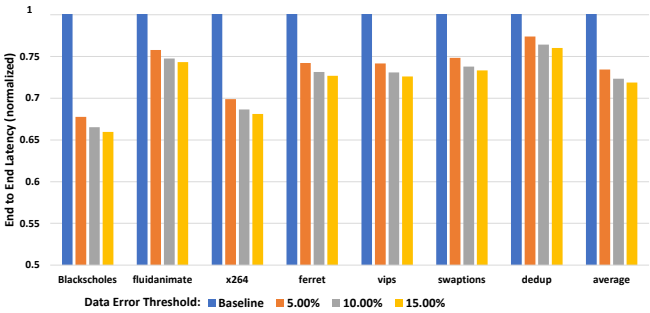

Fig. 9. **End-to-End latency of ARQ+SECDED error control technique under different error threshold configurations**: The results are normalized to baseline.

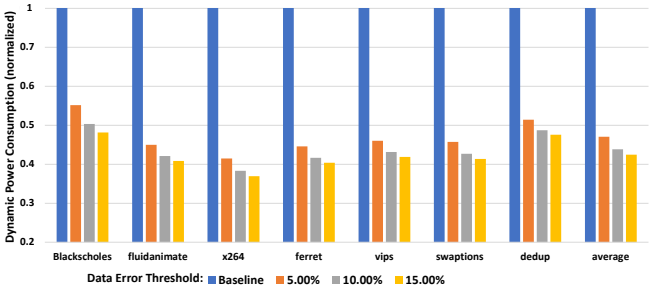

Fig. 10. **Dynamic power consumption of ARQ+CRC error control technique under different error threshold configurations**: The results are normalized to baseline.

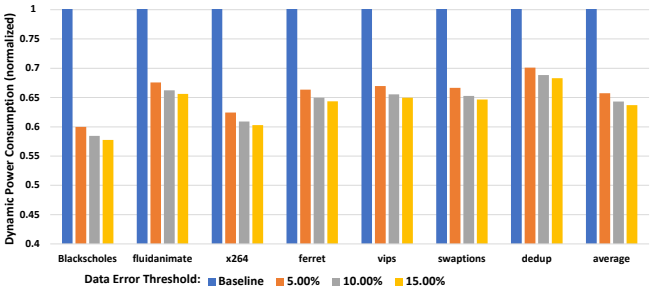

Fig. 11. **Dynamic power consumption of ARQ+SECDED error control technique under different error threshold configurations**: The results are normalized to baseline.

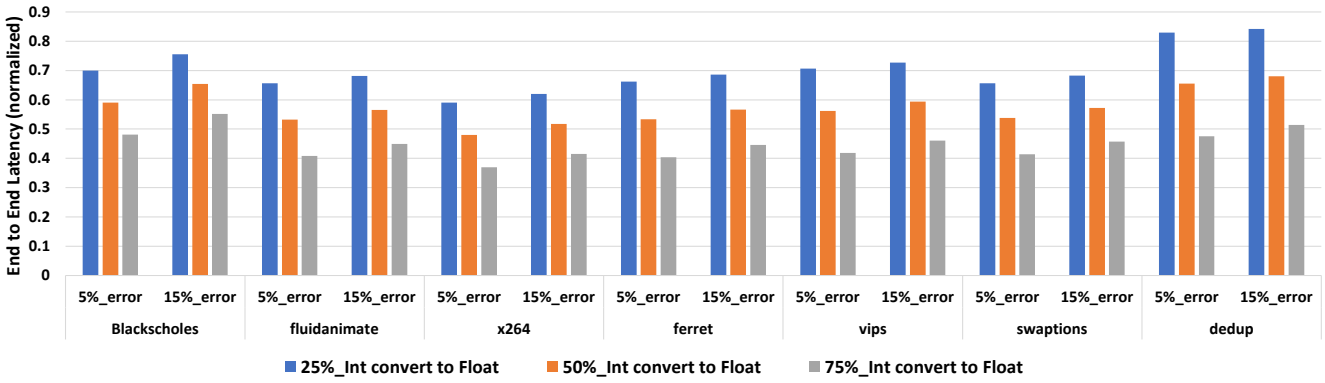

Fig. 12. **Latency sensitivity analysis with the change in integer to floating point packet conversion ratio (25%, 50%, 75%) and acceptable error thresholds (5%, 10%, 15%).**

dynamic power consumption. As shown in Fig. 10, DEC-NoC achieves 58%, 56%, and 53% dynamic power savings on average over static ARQ+CRC baseline across all benchmarks for error thresholds of 5%, 10%, and 15%, respectively. Similarly, Fig. 11 shows that DEC-NoC technique achieves an average dynamic power savings of 36%, 35%, and 34% compared to ARQ+SECDED baseline for the error thresholds mentioned above. Power efficiency is higher for benchmarks with the higher percentage of floating point packets due to the same reasons as mentioned in the previous section.

C. Sensitivity Studies

We analyze the sensitivity of DEC-NoC (with ARQ+CRC) in terms of end-to-end latency with the change in the percentage of integer to floating point packet conversion and acceptable error thresholds. As shown in Fig. 12, the end to end latency improves as the percentage of floating point packet increases, as a floating point packet needs less number of protection bits. For example, latency decreases (improves) with the increase in the data-type conversion from integer to floating point packets, e.g., 25% to 50% conversion. Latency improvement, with the increase in the integer to floating point conversion, is higher for dedup and vips benchmarks, as they have no or low percentage of floating point packets (Fig. 5).

D. Quality of Output Analysis

To analyze the impact of approximate communication to application's output, we evaluate the quality of output in DEC-NoC. The output quality is estimated as follows: output quality requirement is fulfilled as long as output error rate is lower than error threshold level [12]. We illustrate the output error rate of four representative benchmarks in Fig. 13. As shown in that figure, all benchmark applications met output quality requirements under different error threshold constraints.

Furthermore, to demonstrate the impact of approximate communication, we compare the approximate output of vips benchmark (error threshold is set to 15%) and its accurate output. As shown in Fig. 14, the difference between two outputs is negligible and unrecognizable by human vision. To quantify the difference, we measure the output error by using the imabsdiff function in Matlab. The measurement shows very low output error (pixel deviation is only 0.8%).

Fig. 13. **Application Output Accuracy under different error threshold configurations**

Fig. 14. **vips Benchmark Output Comparison**: Error threshold configuration is 15%. Result difference is 0.8%.

E. Area and Latency Overheads

We evaluate the overhead of DEC-NoC in terms of area and latency. We implemented DEC-NoC using verilog and synthesized the design in $32nm$ technology. Synthesis shows that DEC-NoC incurs 0.00031 mm^2 for each network interface, which is only 1% overhead of the overall NoC area. For latency overhead, we found out that the calculation of approximation code and the selection of protected bits at the source node require extra 2 cycles, and the selection of protected bits at the destination node requires extra 1 cycle.

VII. RELATED WORK

Approximate communication has gained attention for energy efficiency and performance improvement, by approximating the output for applications that are fairly tolerant to inaccuracies in the results, like approximate computing. Approximate communication can significantly improve the

system performance as fully accurate communication is costly in terms of power and time compared to computation [14]–[16]. Several works have proposed various approximate communication techniques [28]–[30]. [28] explores the potential benefits of approximate computing for tackling the communication bottleneck issues on massively parallel systems by surveying three promising techniques for approximate communication: compression, relaxed synchronization, and value prediction. The authors in [29] propose a data approximation framework, which reduces the transmission of approximately similar data in the NoC by delivering approximated versions of precise data. This work further uses an underlying NoC compression technique to compress the data block for reducing the volume of data movement across the chip. In [30] the authors propose an approximation-based dynamic traffic regulation, which drops a fraction of packet data to reduce network congestion, and predicts the lost data in packet after being received in the destination node. In this work, we use a very novel approach for approximate communication, which reduces the probability of retransmissions by reducing the number of protected bits based on the error resiliency of an application.

VIII. CONCLUSIONS

In this work, we propose an approximation communication framework called DEC-NoC, consisting on an approximation technique and hardware support for energy-efficient and high-performance NoCs. We design an approximate coding logic to protect the minimum amount of bits for integer and floating point values while maintaining output quality. DEC-NoC greatly reduces the retransmission overhead, which improves dynamic power consumption and end-to-end latency for NoCs. We compare the proposed framework DEC-NoC with traditional communication techniques, ARQ+CRC and ARQ+SECDED. Our detailed evaluation shows that with different error threshold levels for different applications, DEC-NoC reduces end-to-end latency and dynamic power consumption by up to 56% and 58%, respectively, over the ARQ+CRC and ARQ+SECDED error control techniques with negligible hardware and timing overhead. Moreover, our evaluation shows that DEC-NoC satisfies the accuracy requirements of different applications under different error threshold configurations.

IX. ACKNOWLEDGEMENTS

We sincerely thank the reviewers for their helpful comments and suggestions. This research is supported by NSF grant CCF-1812495, CCF-1547034, CCF-1547035, CCF-1540736, and CCF-1702980.

REFERENCES

[1] Michael Bedford Taylor et al. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, 2002.

[2] Y. Hoskote and Others. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61, 2007.

[3] Hangsheng Wang, Li-Shiuan Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. In *Proceedings of MICRO*, pages 105–116, Dec. 2003.

[4] William J. Dally and Brian Towles. Route packets, not wires: On-chip inteconnection networks. In *Proceedings of DAC*, pages 684–689, 2001.

[5] G. De Micheli and Others. Networks on chips: From research to products. pages 300–305, June 2010.

[6] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[7] Davide Bertozzi, Luca Benini, and Giovanni De Micheli. Error control schemes for on-chip communication links: the energy-reliability tradeoff. *IEEE TCAD*, 24(6):818–831, 2005.

[8] Alireza Ejlali and Others. Performability/energy tradeoff in error-control schemes for on-chip networks. *IEEE TVLSI*, 18(1):1–14, 2010.

[9] Thomas Yeh and Others. The art of deception: Adaptive precision reduction for area efficient physics acceleration. In *Proceedings of MICRO*, pages 394–406, 2007.

[10] Jonathan Ying Fai Tong, David Nagle, and Rob A Rutenbar. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE TVLSI*, 8(3):273–286, 2000.

[11] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of MICRO*, pages 449–460, 2012.

[12] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):62, 2016.

[13] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. Approximate computing: A survey. *IEEE Design & Test*, 33(1):8–22, 2016.

[14] Keren Bergman et al. Exascale computing study: Technology challenges in achieving exascale systems, 2008.

[15] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *Proceedings of VECPAR*, pages 1–25, 2010.

[16] F. Zahn, S. Lammel, and H. Frning. Early experiences with saving energy in direct interconnection networks. In *Proceedings of HiPINEB*, pages 33–40, Feb 2017.

[17] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of DAC*, page 113, 2013.

[18] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. Approxann: an approximate computing framework for artificial neural network. In *Proceedings of DATE*, pages 701–706. EDA Consortium, 2015.

[19] Adrian Sampson et al. Enerj: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, volume 46, pages 164–174, 2011.

[20] Daya S Khudia, Babak Zamirai, Mehrzad Samadi, and Scott Mahlke. Rumba: An online quality management system for approximate computing. In *Proceedings of ISCA*, pages 554–566, 2015.

[21] Dan Zuras et al. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

[22] Srinivasan Murali et al. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test of Computers*, 22(5):434–442, 2005.

[23] Nan Jiang and Others. A detailed and flexible cycle-accurate network-on-chip simulator. In *Proceedings of ISPASS*, pages 86–96, 2013.

[24] Joel Hestness, Boris Grot, and Stephen W Keckler. Netrace: dependency-driven trace-based network-on-chip simulation. In *Proceedings of NoCArc*, pages 31–36, 2010.

[25] Rajamohana Hegde and Naresh R Shanbhag. Toward achieving energy efficiency in presence of deep submicron noise. *IEEE TVLSI*, 8(4):379–391, 2000.

[26] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of PACT*, October 2008.

[27] Nick Barrow-Williams, Christian Fensch, and Simon Moore. A communication characterisation of splash-2 and parsec. In *Proceedings of IISWC*, pages 86–97, 2009.

[28] Filipe Betzel et al. Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems. *ACM Comput. Surv.*, 51(1):1:1–1:32, January 2018.

[29] Rahul Boyapati and Others. Approx-noc: A data approximation framework for network-on-chip architectures. In *Proceedings of ISCA*, pages 666–677, 2017.

[30] L. Wang, X. Wang, and Y. Wang. Abdtr: Approximation-based dynamic traffic regulation for networks-on-chip systems. In *Proceedings of ICCD*, pages 153–160, Nov 2017.