An Online Quality Management Framework for Approximate Communication in Network-on-Chips

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

ABSTRACT

Approximate communication is being seriously considered as an effective technique for reducing power consumption and improving the communication efficiency of network-on-chips (NoCs). A major problem faced by these techniques is quality control: how do we ensure that the network will transmit data with sufficient accuracy for applications to produce acceptable results? Previous methods that addressed this issue require each application to calculate the approximation level for every piece of approximable data, which takes hundreds of cycles. So the approximation information is often not available when a request packet is transmitted. Therefore, the reply packet with the approximable data is transmitted with unnecessarily absolute accuracy, reducing the effectiveness of approximate communication.

In this paper, we propose a hardware-based quality management framework for approximate communication to minimize the time needed for the approximation level calculation. The proposed framework employs a configuration algorithm to continuously adjust the quality of every piece of data based on the difference between the output quality and the application's quality requirement. When the proposed framework is implemented in a network, every request packet can be transmitted with the updated approximation level. This framework results in fewer flits in each data packet and reduces traffic in NoCs while meeting the quality requirements of applications. Our cycle-accurate simulation using the AxBench benchmark suite shows that the proposed online quality management framework can reduce network latency by up to 52% and dynamic power consumption by 59% compared to previous approximate communication techniques while ensuring 95% output quality. This hardware-software codesign incurs 1% area overhead over previous techniques.

CCS CONCEPTS

• **Computer systems organization → Interconnection architectures**; **Multicore architectures**; • **Hardware → Network on chip**.

ICS '19, June 26–28, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6079-1/19/06…$15.00
https://doi.org/10.1145/3330345.3330365

KEYWORDS

Approximate Communication, Quality Control, Network-on-Chips (NoCs)

ACM Reference Format:
Yuechen Chen Ahmed Louri. 2019. An Online Quality Management Framework for Approximate Communication in Network-on-Chips. In *2019 International Conference on Supercomputing (ICS '19), June 26–28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3330345.3330365

1 INTRODUCTION

Interconnection networks [1–6] play a critical role in the performance of parallel computing systems, ranging from chip multiprocessors (CMPs) to supercomputers and exascale systems. Previous research [7–10] showed that approximate computing applications, such as pattern recognition, data mining and synthesis, can tolerate modest errors while yielding acceptable results. Consequently, approximate communication techniques for approximate computing applications are starting to receive increasing research attention [11–17]. With such techniques, network-on-chips (NoCs) can achieve better performance (e.g., end-to-end latency and power consumption) in exchange for reduced accuracy of the transmitted data. To reach this goal, previous studies [12–17] proposed lossy-compression-, value-prediction- and protection-based techniques: lossy-compression-based techniques compress data to a lower quality before transmission and decompress each packet at its destination to reduce traffic intensity [12, 13]; value-prediction-based techniques predict data based on value locality, instead of transmitting the data, to reduce packet transmission [16]; and protection-based techniques approximate data by applying protection only to the critical part of a packet to reduce the cost of error correction [17].

These techniques enable significant performance and energy gains, but managing the quality of communication is still a major issue. To tackle this problem, approximate communication techniques [12–17] simply utilize the quality control frameworks [18, 19] designed for approximate computing without any modifications. In these frameworks, quality configurations are calculated based on error monitoring and prediction to ensure the accuracy of the results. Thus, such a quality control framework requires the program designer to assign the desired result at the beginning of a program, leading the application to change its configuration for each approximable packet during execution to meet the quality target [12–15, 17]. The calculated quality configuration is either attached to each request packet [12, 14, 16, 17] or registered in a quality control table [13] for the network to approximate each reply packet.

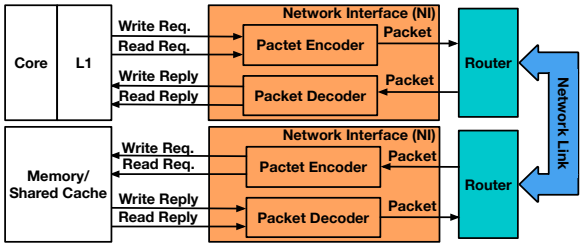


Figure 1: Conventional network interface (NI) design: Read and write request packets are triggered by L1 cache read/write misses.

However, the calculation of the quality configuration requires hundreds of cycles for each piece of data, so the approximation level may not be available before a request packet is sent. The absence of an approximation level in a request packet causes the reply packet to be transmitted with full accuracy instead of being approximated; consequently, the packet requires more time and power to traverse the network. As a result, we observe that previous approximation techniques do not take full advantage of the error tolerances of applications. For example, an approximate computing application may yield a result with an average error of less than 1% even though the application's error tolerance is 5%. This observation indicates that the approximation framework is not utilizing the remaining range of 4% due to inadequate quality management.

To address this issue, we propose an online quality management framework for approximate communication. The goal of this framework is to design a quality control system for a network that can automatically configure the approximation levels of data to be transmitted based on the corresponding applications' output quality requirements. To achieve this goal, we introduce a new NoC design that monitors the application error and adjusts the data approximation level accordingly. Additionally, we design the software interface of the quality management framework to capture the output quality requirements and provide tuning knobs for program designers to manage the system. As a result, the framework can fully utilize the advantages of an application's error tolerance and achieve a balance between output quality and network performance.

The major contributions of this work are as follows:

- The proposed hardware-based quality management framework can continuously modulate the approximation level for each packet based on the output error and the corresponding application's error threshold.
- A software interface is provided to capture the application requirements and provide tuning knobs for program designers to control the quality of approximate communication.
- The performance evaluation of our proposed framework shows that it reduces end-to-end latency and dynamic power consumption by up to 52% and up to 59%, respectively, compared to previous approximate communication techniques while ensuring 95% output quality.

2 MOTIVATIONS AND CHALLENGES

2.1 Motivations

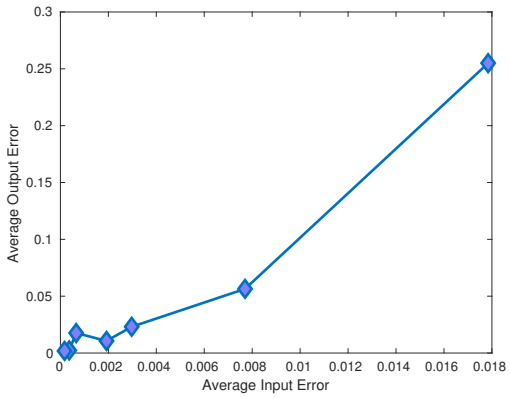


Figure 2: Relative errors on input data versus output data for the Black-Scholes benchmark: There is a clear linear relationship between the relative error on the input data and the relative error on the output data.

2.1.1 The NoC can monitor the input and output data of an application. Figure 1 shows the conventional network interface (NI) design [1, 20, 21]. In this system, when the core loads data from memory and misses the L1 cache, a read request packet will be sent to the memory or the shared cache through the NoC. The memory or shared cache uses a read reply packet to send the required data back to the core. When the core writes the result back to the memory, the result is incorporated into a write request packet and sent to the memory or shared cache through the NoC. After the memory or shared cache has received the data, a write reply is sent back to the core to verify the transmission. In this architecture, if an NI has the address information for the input and output data of an application, the NI can monitor those data during the execution of that application. This observation motivates us to build a quality management framework in the NI to observe and control the approximation levels of both the input and output data.

2.1.2 For quality control purposes, the approximable data are often independent of the rest of the data in an application. Previous research (Rumba [18]) showed that an approximable code block in an application reads and writes data only in a specific address region. Since data will be processed only by approximate computation code, it is easier for a quality control method to monitor and adjust the quality of the output. Rumba suggests that to achieve accurate quality control, approximate computing applications need to analyze the error after a result is calculated. After this analysis, the quality information is stored in the shared memory space to allow the program to decide whether this result meets the quality requirements. Thus, a quality management framework for approximate communication can monitor the output error of an application by examining the traffic to the address where this quality information is stored.

2.1.3 The output error depends on the input error. Previous research [18, 22–25] on quality control indicated that the output quality strongly depends on the input quality. These studies suggest that the output error can be easily and accurately predicted using simple algorithms, such as the moving average. Figure 2 shows the

relative error on the input data versus the relative error on the output data for the Black-Scholes benchmark. The linear relationship between the input and output errors can be clearly seen from this figure. Therefore, a simple error-predicting algorithm can be implemented that incurs low hardware overhead while being capable of managing the approximation level of the input data based on the output error.

2.2 Challenges

2.2.1 Approximation Level Adjustment. The quality management framework needs to adjust the approximation level of the requested data based on the error estimate for the calculated output. To achieve this goal, the framework predicts the output error and then compares it against the application's quality requirements. The error prediction and approximation level calculation procedures must be carefully designed so that the error on the output will not exceed the application's corresponding quality requirement, which could lead to a system crash or re-execution. Meanwhile, the latency of the approximation level generation process must be strictly limited to ensure effective quality control.

2.2.2 Different users and applications have different output quality requirements. In an approximate computing application, the user should be able to tune the output quality based on the program's characteristics. Therefore, the design of the quality management framework should provide sufficient flexibility for the program designer to control the output quality with minimum programming effort.

3 QUALITY MANAGEMENT FRAMEWORK DESIGN PRINCIPLES

The goal of our proposed quality management framework is to continuously modulate the approximation level for each packet based on the error rate of the output data and the application requirements. By comparing an application's error threshold against the error rate of the output data, the framework can automatically adjust the approximation level of the input data to meet the application's needs. This framework can dynamically adjust the approximation level for every packet, whether the application has low or high requirements in terms of data accuracy. The modeling of the approximation level results in aggressive approximation while ensuring the quality of the output, ultimately reducing the NoC power consumption and latency.

3.1 Mathematical Model

Figure 3 shows the mathematical model of the proposed quality management framework for approximate communication, which is based on the three motivations mentioned in Section 2.1. In this framework, we build a feedback loop with an approximation level configurator to adjust the approximation level of the input data. When an approximate computing application runs, the input data are read and processed by an approximable function, and the output data are the result of that function. The output error monitoring logic captures the error rate (quality information) estimated by the application and sends it to the approximation level configurator. The approximation level configurator then uses an error prediction algorithm and an approximation level configuration algorithm to adjust the quality of the input data. The error prediction algorithm

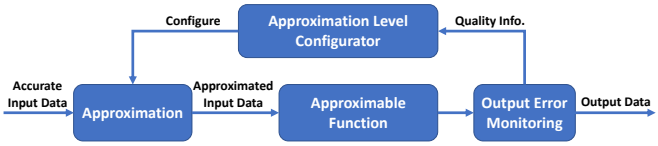


Figure 3: Mathematical model of the proposed online quality management framework for approximate communication.

predicts the error rate of the output data based on the captured quality information using a simple moving average algorithm. The approximation level configuration algorithm adjusts the approximation level by comparing the predicted error rate against the application's quality requirements.

3.2 Approximation Level Configurator

The approximation level configurator includes an error prediction algorithm and an approximation level configuration algorithm for accurate quality management.

We use a simple moving average algorithm to predict the error rate for the output of an application. Equation 1 describes the algorithm for calculating the moving average (MA) of the error rates.

$$MA = Previous\ MA + \frac{p_M}{n} - \frac{p_{(M-n)}}{n} \quad (1)$$

where P_M = error rate of the current output, n = window size, and MA = predicted output error rate. The window size is defined as the number of output errors needed for error predication. When $M < n$, the initial approximation level is used for each request packet. This algorithm calculates the average error rates sampled by a moving window, while the application itself estimates the error rate for every output.

After error prediction, the approximation level configuration algorithm compares the value of MA against the application's quality requirements and adjusts the approximation level accordingly. The quality requirements include both input and output error thresholds. The input error threshold defines the maximum error rate on the input data that the application can tolerate. The output error threshold describes the application's requirement with respect to output quality. The application's output quality is defined as 1 minus the average error rate of the application's output data. For example, suppose that an approximate computing application generates a result with 2 output data points, which have error rates of 2% and 3% according to the evaluation metric. The output error can be calculated to be $(2\% + 3\%)/2 = 2.5\%$, which can be translated into an output quality of 97.5%. The approximation level represents the amount of error injected into the packet due to the applied approximation and will be discussed in detail in Section 4.2 (Table 1).

Algorithm 1 describes the process of approximation level adjustment in detail. First, the algorithm compares the predicted error rate against the output error threshold. When the value of MA is larger than the output error threshold, this indicates that the input error is too large. Therefore, the approximation level configuration algorithm will reduce the approximation level of the input value. If the predicted error rate is substantially less than the output error threshold, this means that the input error is too small. Therefore,

Algorithm 1: Approximation Level Configuration

```
if Predicted Error Rate >= Output Error Threshold then
    Reduce Approximation Level
if Predicted Error Rate < Output Error Threshold then
    if Current Approximation Level < Input Error Threshold
      then
        Increase Approximation Level
```

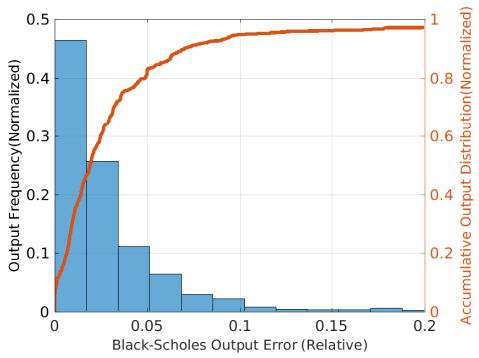


Figure 4: Histogram of output errors and accumulative output error distribution.

without exceeding the input error threshold, the quality management system will increase the approximation level of the packet to achieve better performance. Meanwhile, since this algorithm compares the moving average against the output error threshold, it can ensure that the output error will meet the application's requirements.

The results of our preliminary experiment on the Black-Scholes benchmark are shown in Figure 4, in which are plotted the frequency of output errors and the accumulative output error. Figure 4 shows that with an output error threshold of 5%, a maximum input error of 0.8% and a window size of 10, 80% of the output lies below the 5% error threshold, corresponding to an output quality of 94.9%. This result proves the effectiveness of our proposed quality control algorithm.

4 QUALITY MANAGEMENT FRAMEWORK IMPLEMENTATION

Figure 5 provides a high-level overview of the online quality management framework along with the proposed hardware and software enhancements. A compression-based approximate communication technique uses approximate data compression/decompression logic at the memory/shared cache and core/L1 cache nodes to reduce the data size based on the approximation information stored in the quality control table. Based on the mathematical model and the approximation level configuration algorithm described in Section 3, an approximation level configurator is implemented in the quality management framework to compute and adjust the approximation level of the input data. The output quality monitoring logic extracts the quality information that is generated by each application and embedded in its write request traffic. Then the logic sends the extracted quality information to the approximation level configurator. The quality control table registers the approximation information, including addresses, approximation levels, data types and validity bits.

4.1 Quality Control Work Flow

When an approximate computing application starts running, the quality management framework requires the programmer to set the address of the approximable input data, the data type, the initial approximation level, the output error threshold, the input error threshold, the window size and the address for quality information using the software interface. Then, the approximation level configurator sets the quality control table on the core side using the information captured by the software interface. During the execution of the approximate computing function, the approximation level configurator sets the validity bit in the quality control table to 1. Whenever a read request is issued, the quality control table checks its address. If the requested data can tolerate error, then the quality control table attaches the corresponding approximation information to the packet and sends it out. When the NI at a memory/shared cache node receives a read request packet, the quality control table at that node is updated. When a read reply is sent from a memory/shared cache node, the approximate data compressor reduces the packet size based on the information in the quality control table. Whenever a write request is issued, the application output quality monitor checks its address and extracts the quality information. The quality information is sent to the approximation level configurator for the algorithm to adjust the approximation information registered in the quality control table.

The detailed design of the approximation level configurator is discussed in Section 4.3. The detailed design of the quality control table and the application output quality monitor is discussed in Section 4.4. The software interface of the online quality management framework is detailed in Section 4.5.

4.2 Approximate Data Compression and Decompression

In this framework, we design an approximate data compression method based on data truncation. This work mainly focuses on the quality management system. We use the idea of data truncation from [17] to perform approximate data compression.

4.2.1 Approximate Data Compression. The data compressor reduces the data size by truncating the least significant bits (LSBs) of floating point data based on the approximation level. The higher the approximation level is, the more LSBs will be truncated. We use Table 1 to translate between the approximation level, the relative error and the number of truncated LSBs.

At a memory/shared cache NI, the approximate data compression module distinguishes approximable packets from accurate packets based on the information stored in the quality control table. By checking the address contained by a read reply packet against the quality control table, the approximate data compression logic acquires the transmission requirements for that packet. If the packet requires accurate transmission, the approximate data compression logic directly sends it to the packet encoder. Otherwise, the approximation data compression logic distinguishes the data type based on the information in the quality control table. If the data are of the floating point type, the data compression logic truncates the

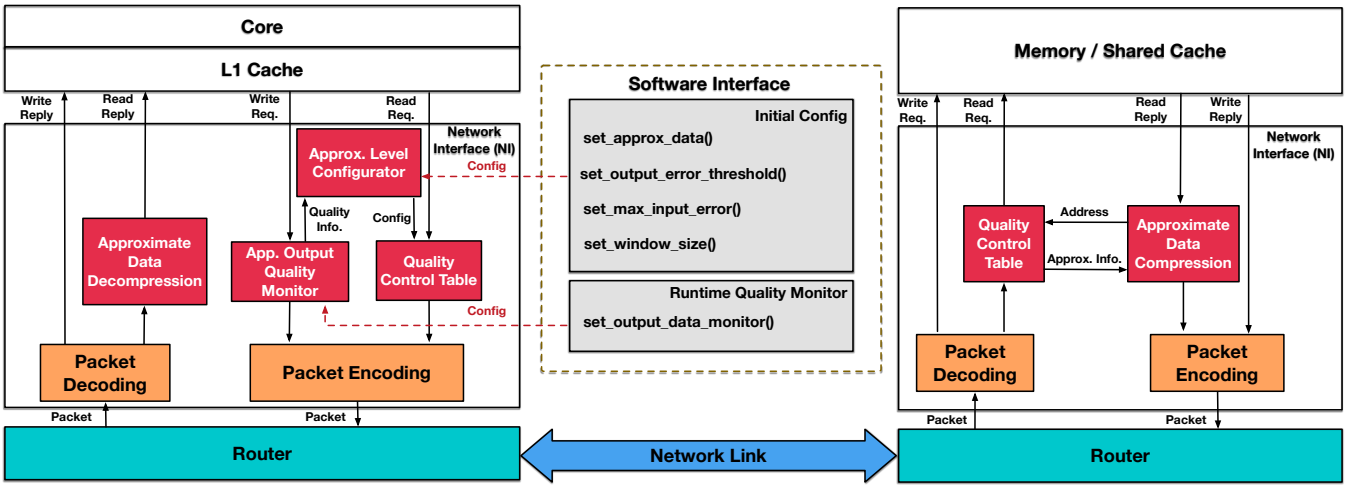


Figure 5: Online quality management framework: The quality information (Quality Info.) includes the error rates measured by the application in accordance with a given evaluation metric. The output quality monitoring logic extracts the quality information from write requests (Write Req.). The quality control table registers the address, the data type, the approximation level and a validity bit for each piece of approximable data. The approximation level configurator uses the error prediction algorithm and the approximation level configuration algorithm to control the approximation level of each read reply packet.

Table 1: Relationship among the relative error, the truncated bits of a floating point value, and the approximation level.

Relative Error	Truncated LSBs	Approximation Level
0.25	21	14
0.125	20	13
0.0625	19	12
0.03125	18	11
0.015625	17	10
0.0078125	16	9
0.00390625	15	8
0.001953125	14	7
0.000976563	13	6
0.000488281	12	5
0.000244141	11	4
0.00012207	10	3
1.52588E-05	7	2
1.90735E-06	4	1
0	0	0

LSBs by looking up the approximation level in Table 1. If the data are of the integer type, the data compression logic checks the data value. If the value is within the range between -2^{24} and 2^{24}, where an integer can be converted into a floating point value without accuracy loss, the data compression logic converts the data into floating point data and then truncates the LSBs in accordance with the approximation level. Otherwise, the integer is sent for packet encoding.

4.2.2 Approximate Data Decompression. By searching for the data address associated with a read reply packet in the quality control table at a core/L1 cache node, the approximate data decompression logic acquires the approximation information for that packet. If the data in a packet are lossy-compressed floating point data, the decompression logic fills in the truncated part with 0s to maintain the standard floating point data structure. Otherwise, if the data in a packet are lossy-compressed integer data, the decompression logic fills in the truncated part and then converts the data into an integer. In other cases, the read reply packet is sent directly to the L1 cache.

4.3 Approximation Level Configurator

The approximation level configurator has two components: the error prediction logic and the approximation level configuration logic. The error prediction logic (Figure 6) is composed of shift registers and a signal converter (Window Size -> Control Signal). The shift registers function as a queue that stores the output errors captured by the application output error monitor. When the number of output quality samples in the queue matches the specified window size, the full flag is raised by the queue. Then, a control signal is generated based on the window size and is input to the AND gates. The signal converter (Window Size -> Control Signal) sets a number of highest bits to 1 based on the window size to filter out data that exceed the length of the window. For example, a window size of 5 is converted into a control signal of 1111100000 (the highest 5 bits are set to 1). On this basis, only the output quality information sampled by the window is calculated, and other quality information is set to zero. The predicted output error result is calculated by the average calculator, which sums all the quality information and divides by the window size. Afterward, the approximation level configuration logic compares the predicted error against the output error threshold, as described in Algorithm 1. If the predicted error is greater than the output error threshold (set_output_error_threshold()), the configurator reduces the approximation level of the approximable data by updating the approximation level in the quality control table. If the predicted error is less than the output error threshold, the configurator checks whether the current approximation level exceeds the maximum input approximation level (set_max_input_error()). If the current input error level is lower than the input error threshold, the configurator increases the approximation level and updates the

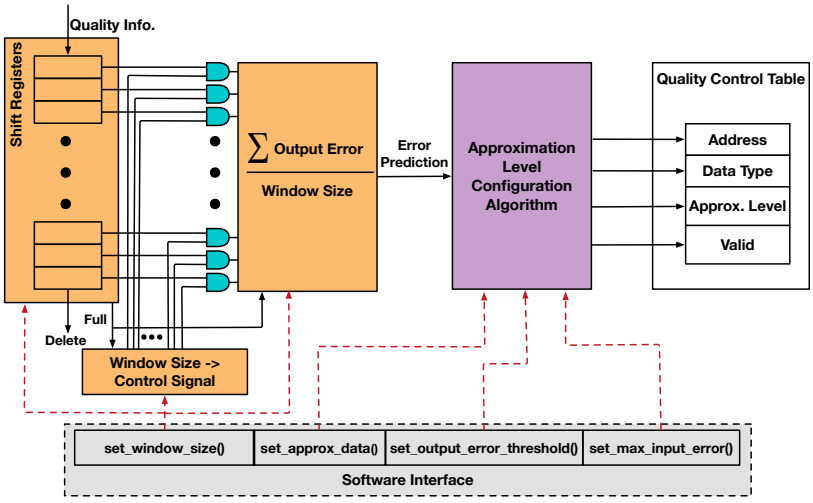


Figure 6: Hardware design of the approximation level configurator: Error prediction is accomplished by the orange part. Approximation level configuration is accomplished by the purple part.

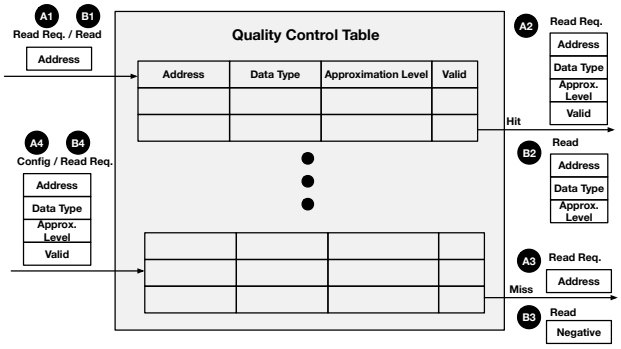


Figure 7: Hardware design of the quality control table: A1, A2, A3, and A4 represent packets encountered by the table at a core/L1 cache node. B1, B2, B3, and B4 represent packets encountered by the table at a memory/shared cache node.

quality control table. Otherwise, the configurator maintains the current approximation level.

4.4 Quality Control Table and Application Output Quality Monitor

4.4.1 Quality Control Table. Figure 7 shows the hardware design of the quality control table at a core/L1 cache node (A) and at a memory/shared cache node (B). The quality control table consists of 4 columns of approximation information: address, data type, approximation level and validity bit. The data type column describes whether a piece of data is an integer or a floating point value. The validity bit indicates whether approximate communication is enabled. The validity bits in the quality control table are set to 0 for accurate communication and 1 for approximate communication.

At a core/L1 cache node, the address column in the quality control table is compared against the address in a read request packet (A1). If the requested address matches an entry in the quality control table, the corresponding approximation information (data type, approximation level, and validity bit) will be attached to the packet (A2). Otherwise, the read request packet (A3) will contain only the address from which data need to be fetched from the memory/shared cache. When a configuration packet (A4) arrives at the quality control table, the address column in the quality control table is compared against the address in the packet to check whether the current entry requires an update. If the address matches an entry in the table, the corresponding information in the rest of the columns of the table is compared against the approximation information in the packet. If the approximation information matches, the entry will not be updated. Otherwise, the entry is updated. If the address does not match any existing entry in the quality control table, a new row is created.

In the NI of a memory/shared cache node, the approximate data compressor sends a read packet (B1) to the quality control table to acquire the approximation information for the data. If the address in the read packet (B1) matches an entry in the quality control table, the corresponding validity bit in the table is checked. If the validity bit is 1, the data type and approximation level (B2) are sent back to the approximate data compressor. Otherwise, a negative signal (B3) is sent to the approximate data compressor to indicate that these data require accurate transmission. When a read request packet (B4) arrives at the quality control table, it is first checked for approximation information. If the packet contains approximation information, the registered addresses in the table are compared against the address in the packet to identify whether an entry requires updating. If the address matches an entry in the table, the corresponding information in the rest of the columns is checked against the approximation information in the packet. If the information matches, the entry will not be updated. Otherwise, the entry is updated. If the address does not match any existing entry in the quality control table, a new row is created.

In this way, this scheme synchronizes the quality control information calculated by the approximation level configurator at a core/L1 cache node with the approximate data compressor at a memory/shared cache node.

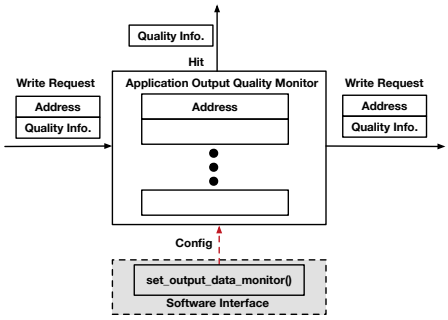


Figure 8: Hardware design of the application output quality monitor: The quality information (Quality Info.) includes the output error estimated by the approximate computing application.

4.4.2 Application Output Quality Monitor. Figure 8 shows the hardware design of the application output quality monitor. The monitor contains a software-controlled table to register the addresses that applications' quality information is stored. When a write request arrives at the quality monitor, the quality monitor compares the address in the packet against the addresses registered in the table. If the write request matches an entry in the table, the monitor copies the quality information carried by the packet and sends it to the approximation level configurator. Then, the packet proceeds to the packet encoder and is sent to the memory/shared cache node.

4.5 Software Interface of the Online Quality Management Framework

We propose a software interface that allows program designer to configure the quality control framework. The software interface of the framework includes the functions (set_approx_data(), set_output_data_monitor()) which allow the quality management framework to identify the approximation tolerance information and quality information in the system address space. Specifically, to create or update an entry in the quality control table, the function set_approx_data() has four arguments: address, data type, initial approximation level and validity. To update the application output quality monitor, the function set_output_data_monitor() needs only one argument: address. The functions set_window_size(), set_output_error_threshold() and set_max_input_error() are designed as tuning knobs for the programmer to adjust the communication quality and set quality requirements of applications. These functions set the window size, the output error threshold and the input error threshold in the approximation level configurator.

5 EXPERIMENTAL SETUP

5.1 Simulation Setup

We evaluated the performance of the proposed online quality management framework using the GEM5 simulator [26] and the AxBench benchmark suite [27]. By running the AxBench benchmarks on the modified GEM5 simulator, we could evaluate the end-to-end network latency. We used DSENT [28] to capture the dynamic power consumption of the network. The detailed settings of the GEM5 simulator are shown in Table 2.

AxBench is an approximate computing benchmark suite with application-specific quality metrics (Table 3). We implemented the proposed software interface in AxBench to evaluate the effectiveness and efficiency of the quality management framework. We integrated the approximate data compression algorithm and the quality control algorithm into the GEM5 simulator to simulate the error injected by the approximate communication technique. In the simulation, our target was 95% output quality.

Table 2: Simulation Environment Setup

NoC Parameters	8 × 8 2D mesh 8 virtual channels Wormhole routing X-Y routing
System Parameters	64 on-chip cores @2 GHz 32 kB L1 instruction cache 32 kB L1 data cache 4-way associative 64-bank fully shared 16 MB L2 cache
Output Error Threshold	5%

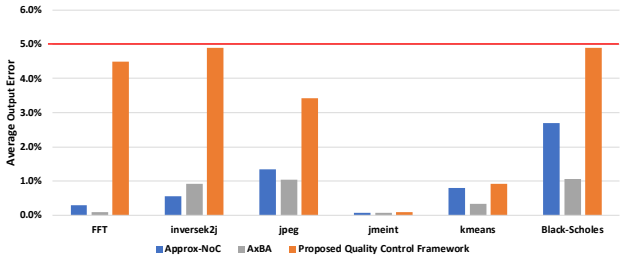


Figure 9: Average output error: The red line represents the application's quality requirement. The closer to the red line, the higher the effectiveness of the quality control method.

6 EVALUATION AND ANALYSIS

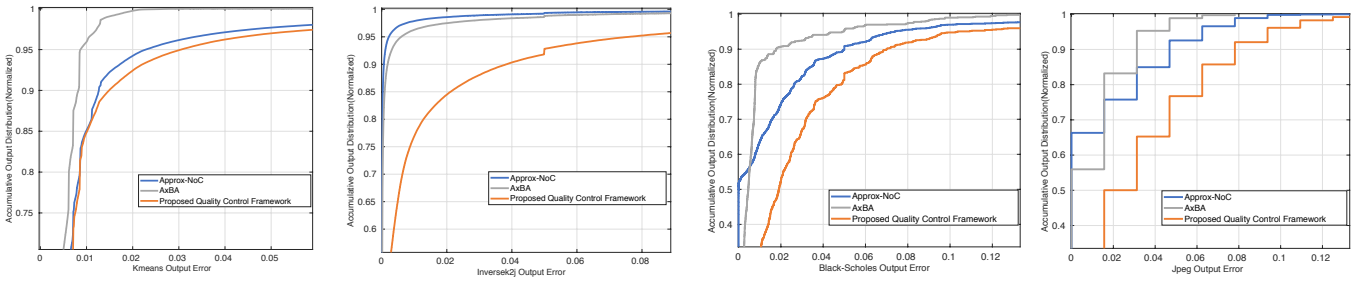
We evaluated the quality management system from four perspectives: output quality, dynamic power consumption, end-to-end latency and overheads. We compared the proposed quality management system with two quality management systems for previous approximate communication techniques: Approx-NoC [12] and AxBA [13]. The quality management systems for both techniques require each application to assign and calculate the approximation levels for the data. The Approx-NoC system requires the approximation level to be attached to the read request packet if the requested data can be approximated. The AxBA system includes a quality control table with a software interface to register the approximation levels for the data.

6.1 Output Quality

6.1.1 Output Error Analysis. The output errors were measured using the application-specific metrics given in Table 3. Figure 9 shows the errors on the outputs for different benchmarks and quality control methods. The y-axis in this figure shows the average output error, which represents the quality of the result. An average output error of 5% indicates 95% output quality. The red line in this figure indicates the approximate computing application's output error requirement. The techniques that achieve errors closest to the output error threshold in this figure exert the most effective quality control.

Table 3: AxBench Benchmarks Suite [27]

Benchmark	Input Data Size	Output Data Size	Evaluation Metric	Output Quality Monitoring Frequency
Black-Scholes	64k fp	64k fp	Average relative error	1 per output
fft	5k random fp numbers	5k fp values	Average relative error	1 per output
inversek2j	100k random (x,y) points	100k (x,y) points	Average relative error	1 per output
jmeint	10k pairs of 3D triangles	10k Boolean values	# of mismatches	1 per 100 outputs
jpeg	512 * 512 pixel image	512*512 pixel image	Average pixel diff.	1 per 64 pixels
kmeans	512 * 512 pixel image	512*512 pixel image	Average output diff.	1 per 60 pixels



(a) Kmeans (b) Inversek2j (c) Black-Scholes (d) Jpeg

Figure 10: Output error distribution.

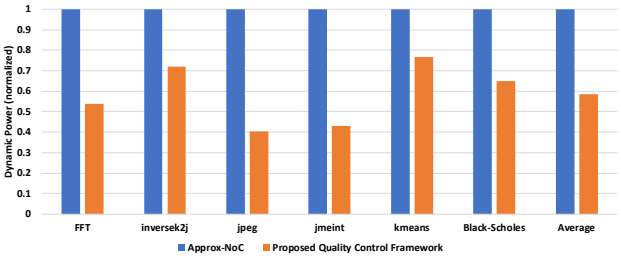


Figure 11: Dynamic power consumption of the proposed quality management framework: The results are normalized with respect to those of Approx-NoC (lower is better).

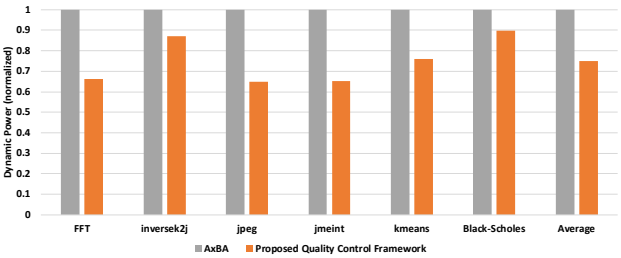


Figure 12: Dynamic power consumption of the proposed quality management framework: The results are normalized with respect to those of AxBA (lower is better).

This figure shows that our online quality management framework achieves higher effectiveness than the software-based quality management systems while ensuring that the output quality is within the tolerance. We can see that on the fft, inversek2j, jpeg, jmeint, kmeans and Black-Scholes benchmarks, our method achieves output errors of 4.5%, 4.9%, 3.4%, 0.1%, 0.9% and 4.9%, respectively. Due to the latency caused by calculating the error threshold for every packet, the quality control method used by Approx-NoC achieves output errors of only 0.2%, 0.5%, 1.3%, 0.08%, 0.8% and 2.7%, respectively. Moreover, AxBA achieves output errors of only 0.08%, 0.92%, 1.05%, 0.08%, 0.34% and 1.06%, respectively, showing significantly worse approximation performance than the proposed approach.

Jmeint, jpeg and kmeans have a lower frequency of quality monitoring compared to the rest of applications in the benchmark suite (Table 3), which leads to less approximation level adjustment for these benchmarks. When a significant output error occurs, the proposed quality control framework reduces the approximation level to avoid re-execution. Since these applications offer fewer chances for the framework to reduce the approximation level, the input error threshold is lowered for the benchmark to ensure the output quality. As a result, jmeint (0.1% error), jpeg (3.4% error) and kmeans (0.9% error) achieve fewer output errors compared to other applications in the benchmark suite.

6.1.2 Output Error Distribution Analysis. Figure 10 shows the accumulative output error distributions for AxBench achieved with the different approximate communication techniques. The output error is monitored at the end of each iteration, and the quality management framework then adjusts the approximation level for the requested data. Compared to the software-based quality management frameworks, the proposed quality management framework has a lower accumulative output error curve, which indicates a higher output error on average. Since AxBA includes a quality control table, the number of full-accuracy outputs is greatly reduced, while in the case of Approx-NoC, Figure 10(b) shows that 80% of its output is calculated at full accuracy. Figure 10(a) shows the accumulative output error for the kmeans benchmark, which has a low frequency of output quality monitoring. In this case, we can see that the low frequency of quality monitoring limits the approximation level adjustment, leading to less approximation on average. Figure 10(d) shows the output error for the jpeg benchmark, which is based on integer calculations. Since quality monitoring is performed every 64 pixels for this benchmark and the evaluation

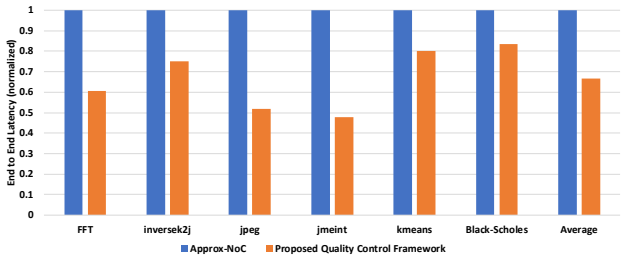


Figure 13: End-to-end latency of the proposed quality management framework: The results are normalized with respect to those of Approx-NoC (lower is better).

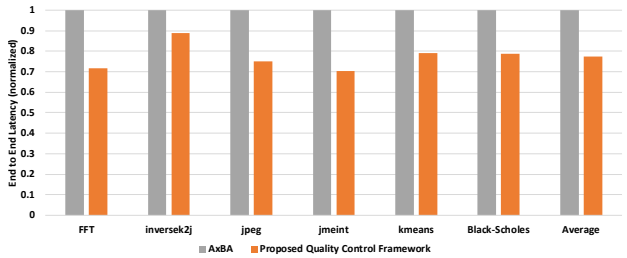


Figure 14: End-to-end latency of the proposed quality management framework: The results are normalized with respect to those of AxBA (lower is better).

metric quantifies the output error as the number of different pixels, the number of output error levels is limited to 64 (1.5% per level) for this benchmark; this is the cause of the discrete output error.

6.2 Power Consumption

Figures 11 and 12 show the dynamic power consumed by the different approximate communication techniques in comparison to our quality management framework for a target output quality of 95%. The y-axis in each of these figures indicates the dynamic power consumption.

As shown in Figure 11, the proposed framework achieves an average dynamic power reduction of 41.5% over Approx-NoC. The largest dynamic power reduction in the experiment is achieved for the jpeg benchmark, while the least dynamic power improvement is obtained for the kmeans benchmark. In this comparison, we find that with a higher rate of output quality monitoring, the proposed framework can adjust the approximation level more frequently and achieve better dynamic power improvement. On the other hand, Approx-NoC transmits some packets at full accuracy, causing high dynamic power consumption.

Figure 12 compares the dynamic power consumption between the proposed framework and AxBA. Because AxBA includes a quality control table to register the approximation levels for the data, all approximable packets can be compressed. However, since the approximation level is not frequently updated, applications use lower approximation levels to avoid re-execution. Consequently, we observe that the proposed quality management framework saves 25% dynamic power on average over AxBA.

A comparison of the average dynamic power savings between Figures 11 and 12 shows that the proposed quality control framework achieves better dynamic power savings relative to Approx-NoC.

6.3 End-to-End Latency

Figures 13 and 14 show the evaluation results for the average end-to-end latency normalized with respect to Approx-NoC and AxBA, respectively. The end-to-end latency is defined as the number of clock cycles between when a packet is injected at the source node and when it is successfully delivered to its destination. As shown in Figure 13, our online quality management framework achieves an average end-to-end latency reduction of 34% over Approx-NoC. Figure 14 shows that the proposed quality management framework achieves an average end-to-end latency savings of 23% compared to AxBA. For the reasons discussed above, the largest latency reduction in both cases is achieved for the jpeg benchmark, whereas the least latency improvement is obtained for the kmeans benchmark. The proposed quality management framework achieves high latency reductions compared to both Approx-NoC and AxBA because the proposed framework aggressively approximates the data in the communication traffic.

6.4 Area and Latency Overheads

We evaluated the overheads of our online quality management framework in terms of area and latency. We implemented the framework with 100 entries in the quality control table, a maximum window size of 100 and an output quality monitor with 20 entries using Verilog. We synthesized the design with 32 nm technology using Synopsys Design Vision. The synthesis results show that the framework incurs an overhead of 8.23 μm^2 for each network interface, which is only 1% of the total NoC area. Regarding the latency overhead, we find that compression and decompression, searching and updating the quality control table, and calculating the quality configuration each require one cycle when the queue is full.

7 RELATED WORKS

As computer architects seek to improve performance and energy efficiency for CMPs, efficient communication between processing elements is critical [5, 6]. Although many techniques were proposed towards an efficient NoC design [29–32], approximate communication is considered as the most effective way to improve network performance when an application can tolerate modest errors [11–16]. One of the most important aspect of designing an approximation method is quality control [9]. A quality management framework ensures that an approximate computing system will generate results with acceptable errors. Rumba [18] uses a lightweight check to eliminate large approximation errors in the execution of an approximate computing technique. ApproxIt [19] relies on a run-time quality calibration scheme and reconfiguration control policy for approximate computing applications based on iterative methods. The authors of [33] proposed the Approxilyzer to quantify the quality impact of a single-bit error and strike a balance between output quality and error resiliency. In [34], the authors suggested controlling the output error by managing the input error. In previous works on approximate communication [12, 13], software-based quality management frameworks were applied to control the quality of communication. Our mechanism reduces the time needed to calculate the quality requirement for each packet and achieves aggressive approximation without violating quality requirements.

8 CONCLUSIONS

Approximate communication techniques can be employed by approximate computing applications in various domains, such as pattern recognition, data mining and synthesis. These techniques enhance communication performance (e.g., power consumption and end-to-end latency) while ensuring acceptable output quality. However, current application-based quality control methods can fail to calculate the approximation level before the transmission of a request packet, causing the reply packet to be sent without approximation and leading to inefficient approximate communication. In this paper, we propose an online quality management system for approximate communication, consisting of a hardware design and a software interface for energy-efficient and low-latency NoCs. We designed an approximation level configurator to calculate the error threshold for each packet based on the output quality and the quality requirements of the corresponding approximate computing application. We compared the proposed framework with two previous approximate communication techniques: Approx-NoC and AxBA. Our detailed evaluation showed that the proposed quality management system reduces end-to-end latency and dynamic power consumption by up to 52% and 59%, respectively, compared with previous approximate communication techniques. Moreover, our evaluation demonstrated that the proposed quality management system improves the distribution of the output error rates while satisfying the stipulated quality requirements.

ACKNOWLEDGMENTS

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

REFERENCES

[1] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[2] William J. Dally and Brian Towles. Route packets, not wires: On-chip inteconnection networks. In *Proceedings of the 38th Design Automation Conference (DAC)*, pages 684–689, June 2001.

[3] G. De Micheli, C. Seiculescu, S. Murali, L. Benini, F. Angiolini, and A. Pullini. Networks on chips: From research to products. In *Proceedings of the 47th Design Automation Conference (DAC)*, pages 300–305, June 2010.

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

[5] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15, 2008.

[6] Daniel A. Reed and Jack Dongarra. Exascale computing and big data. *Commun. ACM*, 58(7):56–68, June 2015.

[7] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 301–312, New York, NY, USA, 2012. ACM.

[8] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *18th IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2013.

[9] Q. Xu, T. Mytkowicz, and N. S. Kim. Approximate computing: A survey. *IEEE Design Test*, 33(1):8–22, Feb 2016.

[10] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, March 2016.

[11] Filipe Betzel, Karen Khatamifard, Harini Suresh, David J. Lilja, John Sartori, and Ulya Karpuzcu. Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems. *ACM Comput. Surv.*, 51(1):1:1–1:32, January 2018.

[12] Rahul Boyapati, Jiayi Huang, Pritam Majumder, Ki Hwan Yum, and Eun Jung Kim. Approx-noc: A data approximation framework for network-on-chip architectures. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 666–677, New York, NY, USA, 2017. ACM.

[13] Jacob R. Stevens, Ashish Ranjan, and Anand Raghunathan. Axba: An approximate bus architecture framework. In *Proceedings of the International Conference on Computer-Aided Design*, pages 43:1–43:8, New York, NY, USA, 2018. ACM.

[14] A. B. Ahmed, D. Fujiki, H. Matsutani, M. Koibuchi, and H. Amano. Axnoc: Low-power approximate network-on-chips using critical-path isolation. In *12th International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, Oct 2018.

[15] V. Y. Raparti and S. Pasricha. Dapper: Data aware approximate noc for gpgpu architectures. In *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, Oct 2018.

[16] L. Wang, X. Wang, and Y. Wang. Abdtr: Approximation-based dynamic traffic regulation for networks-on-chip systems. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 153–160, Nov 2017.

[17] 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 *IEEE 36th International Conference on Computer Design (ICCD)*, pages 480–487, Oct 2018.

[18] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke. Rumba: An online quality management system for approximate computing. In *42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 554–566, June 2015.

[19] Q. Zhang and Q. Xu. Approxit: A quality management framework of approximate computing for iterative methods. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2018.

[20] A. Bakhoda, J. Kim, and T. M. Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 421–432, Dec 2010.

[21] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[22] Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, pages 198–209, New York, NY, USA, 2010. ACM.

[23] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke. Sage: Self-tuning approximation for graphics engines. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 13–24, Dec 2013.

[24] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. Paraprox: Pattern-based approximation for data parallel applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 35–50, New York, NY, USA, 2014. ACM.

[25] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 449–460, Washington, DC, USA, 2012. IEEE Computer Society.

[26] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[27] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran. Axbench: A multiplatform benchmark suite for approximate computing. *IEEE Design Test*, 34(2):60–68, April 2017.

[28] C. Sun, C. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. Peh, and V. Stojanovic. DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. In *IEEE/ACM 6th International Symposium on Networks-on-Chip (NOCS)*, pages 201–210, May 2012.

[29] Hao Zheng and Ahmed Louri. An energy-efficient network-on-chip design using reinforcement learning. In *56th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2019.

[30] K. Wang, A. Louri, A. Karanth, and R. Bunescu. High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In *22th Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019.

[31] K. Wang, A. Louri, A. Karanth, and R. Bunescu. Intellinoc: A holistic framework for energy-efficient and reliable on-chip communication for manycores. In *45th IEEE International Symposium on Computer Architecture (ISCA)*, 2019.

[32] H. Zheng and A. Louri. EZ-Pass: An Energy Performance-Efficient Power-Gating Router Architecture for Scalable NoCs. *IEEE Computer Architecture Letters*, 17(1):88–91, Jan 2018.

[33] Radha Venkatagiri, Abdulrahman Mahmoud, Siva Kumar Sastry Hari, and Sarita V. Adve. Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 42:1–42:14, Piscataway, NJ, USA, 2016.

[34] Michael A. Laurenzano, Parker Hill, Mehrzad Samadi, Scott Mahlke, Jason Mars, and Lingjia Tang. Input responsiveness: Using canary inputs to dynamically steer approximation. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 161–176, New York, NY, USA, 2016.