On Performance Optimization and Quality Control for Approximate-communication-enabled Networks-on-Chip

Siyuan Xiao, Xiaohang Wang, *Member, IEEE,* Maurizio Palesi, *Senior Member, IEEE,* Amit Kumar Singh, *Member, IEEE,* Liang Wang and Terrence Mak, *Senior Member, IEEE*

Abstract—For many applications showing error forgiveness, approximate computing is a new design paradigm that trades application output accuracy for mitigating computation/communication effort, which results in performance/energy benefit. Since networks-on-chip (NoCs) are one of the major contributors to system performance and power consumption, the underlying communication is approximated to achieve time/energy improvement. However, performing approximation blindly causes unacceptable quality loss. In this paper, first, an optimization problem to maximize NoC performance is formulated with the constraint of application quality requirement, and the application quality loss is studied. Second, a congestion-aware quality control method is proposed to improve system performance by aggressively dropping network data, which is based on flow prediction and a lightweight heuristic. In the experiments, two recent approximation methods for NoCs are augmented with our proposed control method to compare with their original ones. Experimental results show that our proposed method can speed up execution by as much as 29.42% over the two state-of-the-art works.

Index Terms—approximate computing, many-core system, networks-on-chip

1 Introduction

Many-core systems have been widely used for running machine learning and multimedia applications. Many of them, classified as recognition, mining, and synthesis (RMS) applications, are intrinsically error forgiving [1]. There are no golden results for these applications. Therefore, trade-offs between accuracy and other metrics (*e.g.* execution time, power consumption) can be made. Approximate computing [2], which exploits the error-resilience of these applications, introduces new opportunities for system design. In approximate computing, the computation effort is reduced (*e.g.* using low precision adder/multiplier [3], loop perforation [4], load value approximation [5], *etc.*) to either accelerate execution or reduce power consumption. Each of the approximators that actually performs the approximate computation/communication works with a quality control method. A quality control method is designed to make good use of the approximator such that the final quality loss is within a user-defined bound. If a quality control method is not well-designed, it results in either approximation *underutilization* (performing approximation conservatively such that the performance improvement is not fully exploited) or *overutilization* (performing approximation too aggressively which leads to unacceptable quality degradation in the application output).

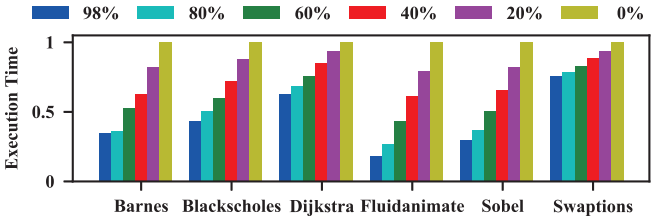


Fig. 1. Execution time at varying data drop rate for different applications.

Among various on-chip hardware components, networks-on-chip (NoCs) have a large impact on system performance and power consumption. Therefore, approximate NoCs become an attractive design, which drop data packets before they are injected into the network and recover them at the destination to reduce network workloads. In an approximate NoC, each network interface (NI) is equipped with a data dropper and a recoverer, used for data dropping and recovering, respectively.

A motivational experiment is presented in Figure 1 to show the feasibility of improving system performance by dropping network traffic. We define drop rate as the proportion of data that are discarded at each NI, which never

- *Siyuan Xiao and Xiaohang Wang are with the School of Software Engineering, South China University of Technology, Guangzhou, China, 510006. Xiaohang Wang is the corresponding author.*
- *Maurizio Palesi is with Department of Electrical, Electronics and Computer Engineering, University of Catania, Catania 95124, Italy. E-mail: maurizio.palesi@dieei.unict.it.*
- *Amit Kumar Singh is with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, United Kingdom. E-mail: a.k.singh@essex.ac.uk.*
- *Liang Wang is with the Institute of Microelectronics, Tsinghua University, Beijing, China. Email: wl23189@163.com.*
- *Terrence Mak is with the School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, United Kingdom, and with the Guangzhou Institute of Advanced Technology, CAS, Guangzhou, 511458, China. E-mail: tmak@ecs.soton.ac.uk.*

This research program is supported by the Natural Science Foundation of Guangdong Province No. 2018A030313166, Pearl River S&T Nova Program of Guangzhou No. 201806010038, the Fundamental Research Funds for the Central Universities No. 2019MS087, Open Research Grant of State Key Laboratory of Computer Architecture Institute of Computing Technology Chinese Academy of Sciences No. CARCH201916, and the Natural Science Foundation of China No. 61971200.

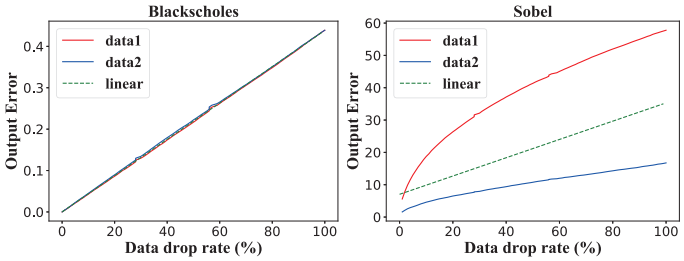


Fig. 2. The output errors under different data drop rates for two applications (a) Blackscholes and (b) Sobel.

enter the network. In this example, the on-chip network of a full-system many core simulator [6] is modified to change the size of data packets uniformly according to a given drop rate, right before they are injected into the network. The drop rate varies from 0% (undropped) to 98% (almost dropped). As shown in Figure 1, dropping data can significantly reduce execution time. For example, dropping 60% data speeds up the execution by as much as 25.77% compared to the case of only dropping 20% data, on average. The reason is that more communication workloads lead to longer execution stalls, and the overall performance is thus degraded.

However, previous works on approximate NoCs [7], [8], [9] are unable to maximize the performance gain introduced by approximation. [7], [9] do not consider network congestion. Since network latency is sensitive to network congestion, dropping data without congestion awareness causes approximation underutilization (performing approximation on non-critical packets such that the benefit is less significant). Moreover, dropping data without estimating the application-specific quality loss leads to approximation overutilization (unacceptable error).

Besides performance optimization, [7], [8], [9] do not have an accurate quality model for applications. The quality models of [8], [9] assume that output error grows linearly with respect to input data error. However, this linearity assumption does not hold for many applications, which is shown in Figure 2.

In Figure 2, quality loss results are collected using the data dropping method ABDTR [8], with "data1" and "data2" being output error results with two different input data, respectively. The dashed line "linear" is obtained by linear regression. For Blackscholes, as stated in the previous work [9], it is almost linear. On the contrary, for Sobel, it shows non-linearity. Besides, Figure 2(b) indicates that the output error depends not only on data drop rate, but also input data. Previous works [8] [9] [10] neglect this important factor, which leads to inaccuracy in their quality models.

Therefore, in this paper, we first propose an accurate quality model. We formulate the performance optimization problem for approximate NoCs, followed by proposing ACDC (Accuracy- and Congestion-aware Dynamic traffic Control), a lightweight control mechanism to solve it.

The main contributions of this paper are as follows:

1) Quality loss, network congestion, and zero load latency are analyzed and modeled. An optimization problem is formulated to minimize network latency, subject to a user-defined error bound.
2) A lightweight control mechanism is proposed to solve the problem, which consists of a periodically-triggered global controller and local controllers. Based on the flow prediction, the global controller first solves the congestion minimization problem, followed by exploiting the opportunity to reduce zero load latency.
3) Compared against two recent works [7], [8], experimental results show that ACDC mitigates approximation underutilization and overutilization. For some applications (*e.g.*, Sobel, Blackscholes), ACDC achieves execution speedup and energy saving by more than 20% over the two recent works.

In this paper, our previous work [10] has been significantly extended as follows:

1) Quality loss of different applications are examined by a motivational experiment. Experimental results show that quality loss also depends on input data. Besides, in some applications, quality loss does not increase linearly with respect to data drop rate. We update the quality model to include these factors.
2) We have analyzed the flow changes, and enhanced the runtime control method to handle bursty communications, which start suddenly with a high traffic volume.
3) Another state-of-the-art work APPROX-NoC [7] is compared with our proposed method. Experimental results show that half of the applications violate the quality loss constraint using APPROX-NoC, but none of them violate the constraint using our proposed method.
4) The rest of the paper has also been substantially extended. For example, we have included runtime drop rate computation. We have also evaluated the congestion changes caused by approximate communication, and the quality impact on Sobel with a real image.

The rest of this paper is organized as follows. We first survey related work in Section 2. The optimization problem is formulated in Section 3. After that, we elaborate our proposed control method in Section 4. Experimental results are evaluated in Section 5. Finally, we conclude our work in Section 6.

2 Related Work

Approximate computing [11], as an emerging powerful compute paradigm, trades application output quality for energy-efficiency based on error forgiveness in applications. Approximate computing approaches span from application layer to circuit layer.

In the application layer, loop perforation [4] reduces computation effort by selectively skipping some iterations, while SAGE [12] generates a set of CUDA kernels with various levels of approximation. At the architecture level, EnerJ [13] enhances the Java programming language with approximate annotations, along with several approximation-aware instruction set architecture (ISA) extensions.

Esmaeilzadeh et al. [14] proposed a general purpose approximate computing approach that transforms and maps the computational intensive kernels of an application onto an energy-efficient accelerator. Approximate memory was proposed in [5] [15] [16] [17]. For instance, for a cache miss, load value approximation [15] avoids remote cache accesses by predicting the miss value locally. In the circuit layer, researchers proposed imprecise logic [3], voltage overscaling [18], and frequency tuning [19].

Quality control is a crucial component in approximate computing. A lot of works are devoted to designing quality control mechanisms. OPPROX [20] aims at maximizing performance improvement. MEANTIME [21] leverages approximate computing to meet realtime requirement. Rumba [22], CoAdapt [23], and PowerDial [24] propose power-aware quality controls. DES [25] tries to maximally exploit error resilience under quality limitation, while AdAM [26] considers memory lifetime. Moreover, several works [27] [28] [29] discuss about quality modeling, and AxGames [30] designs several Web games to model users' sensitivity to quality loss.

Since the underlying interconnect is an important component of a many-core system [31] [32] [33], recent studies try to drop data in NoCs. The runahead NoC [34] relaxes the constraint of lossless communication by using an additional low-latency lossy network, and retransmits the dropped packets by the regular lossless network. Though data packets are occasionally dropped, the communication remains lossless thanks to the retransmission.

On the contrary, other works [7] [8] [9] [35] [36] [37] [38] [39] apply approximate computing to NoCs. APPROX-NoC [7] allows more data to be compressed using existing pattern-based NoC compression techniques, by compressing similar patterns into a shorter word. ABDTR [8] drops data with a fixed interval, while recovering the lost data based on linear interpolation. Data are dropped according to the runtime buffer utilization of each router. [9] directly truncates the Least Significant Bits (LSBs) to reduce network traffic, and the number of truncated bits for each single value is decided with respect to the relative error. To guarantee quality safety, [9] uses a linear model to estimate output error at runtime. A dual-voltage router architecture was proposed in AxNoC [35], which tolerates bit flips running in a low-power mode. In DAPPER [36], similar data flits can be transferred simultaneously using an auxiliary network. Ascia et al. [37] proposed a method to save energy consumption by reducing the voltage swing of links, which in effect introduces some bit flips. They also proposed approximate wireless-NoCs which consider long-distance wireless communication [38] [39]. These works about approximate NoCs either perform approximation conservatively or are not well-guided (lacking application-specific quality loss models), thus result in approximation underutilization/overutilization.

3 Problem Formulation

In this section, we first introduce the preliminaries, followed by modeling important metrics of approximate NoCs. Finally, an optimization problem is formulated to optimize network performance, subject to a quality loss bound.

TABLE 1
Notations

Name	Description
n	The number of nodes
m	The number of links, depending on topology
ω	The size of each data packet, in bits
f_{ij}	The flow from nodes i to j
v_{ij}	The lossless communication volume of f_{ij}, in bits
p_{ij}	The proportion of approximable data in f_{ij}
c	Link capacity, in bits, which is the maximum amount of data communication that do not cause a link to be congested
r_{ijk}	$r_{ijk} = \begin{cases} 1 & \text{if } f_{ij} \text{ passes through link } k \\ 0 & \text{otherwise} \end{cases}$
$X = \{\pi_1, \ldots, \pi_d\}$	A set of data drop rate options in an ascending order
x_{ij}	Data drop rate for f_{ij}, in fraction
$q_a(\pi)$	The quality model for application a, which returns quality loss given a data drop rate
θ_a	The quality requirement of application a, defined as the maximum acceptable error
g	The total error budget, in bits, which is the maximum amount of data that can be dropped
μ	The error budget for congestion minimization, in bits
μ_i	The portion of μ allocated to node i, in bits
ν	The error budget for reducing serialization latency, in bits
ν_i	The portion of ν allocated to node i, in bits
c_k^{link}	The link congestion metric of link k, in bits
c_{ij}^{flow}	The flow congestion metric of f_{ij}, in bits

3.1 Preliminaries

Notations used throughout this paper are listed in Table 1.

Let n denote the number of network nodes. The number of links is denoted by m, which is topology-dependent.

A flow f_{ij} is defined as the communication between a pair of a source and a destination. v_{ij}, the volume of f_{ij}, is defined as the amount of data communication along that flow. Link capacity c is defined as the maximum amount of data communication that a link can afford. r_{ijk} is a binary value indicating whether f_{ij} passes through link k or not, which depends on the underlying routing algorithm.

A set of the available drop rate options is defined as $X = \{\pi_1, \ldots, \pi_d\}$, where $\pi_i \in [0, 1]$, $\pi_i < \pi_{i+1}$, $i = 1, \ldots, d$. Dropping data under a drop rate $\pi_i \in X$ indicates that the amount of data after dropping is $(1 - \pi_i)$ of the original one. A quality model $q_a(\cdot)$ is built for each application a, which estimates the quality loss caused by dropping a certain proportion of approximable data. θ_a is a user-defined quality requirement (quality loss upper-bound), indicating the maximum acceptable error for application a.

The proportion of data that can be dropped is $q_a^{-1}(\theta_a)$, where $q_a^{-1}(\cdot)$ is the inverse function of $q_a(\cdot)$. The total flow volume of the whole network is written as $\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}(p_{ij}v_{ij})$, *i.e.*, summing the approximable volumes of all $n(n-1)$ flows. The total error budget g is

$$g = q_a^{-1}(\theta_a)\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}(p_{ij}v_{ij}) \tag{1}$$

which is the maximum amount of flow volume allowed to be dropped, according to the user-defined quality requirement.

3.2 The Quality Model

Approximation level and input data influence output quality loss. In this paper, we propose a lightweight quality model considering both of these two factors. It is built by offline profiling as follows:

1) Approximable data in source code (*e.g.*, the pixels of input images in the application Sobel) of an application are figured out. The source code is modified such that the original data are replaced by approximated (lossy) version, which are generated by a piece of code that mimics the approximator.
2) The approximator provides a set of d data drop rate options $X = \{\pi_1, \ldots, \pi_d\}$. An input dataset is provided for each application. For each instance of input data, the application runs d times by varying the data drop rate π, and the corresponding quality loss ϵ is collected. That is, for the i^{th} input data instance, the j^{th} application execution generates a pair of $\langle \pi_{i,j}, \epsilon_{i,j} \rangle$.
3) Three statistical features of input data are used as input variables to the quality model, which are mean value, Mean Absolute Error (MAE), and Relative Mean Square Error (RMSE), denoted by ξ_{mean}, ξ_{MAE}, and ξ_{RMSE}, respectively. For a data vector $\{s_1, s_2, \ldots, s_{n_s}\}$ with n_s values, these features are calculated as
$$\xi_{\text{mean}} = \frac{\sum_{i=1}^{n_s} s_i}{n_s}$$
$$\xi_{\text{MAE}} = \frac{\sum_{i=1}^{n_s} |s_i - \xi_{\text{mean}}|}{n_s}$$
$$\xi_{\text{RMSE}} = \sqrt{\frac{\sum_{i=1}^{n_s} (s_i - \xi_{\text{mean}})^2}{n_s}}$$
Statistical features of all the instances are calculated, for training the quality model in the next step.
4) The quality model takes the statistical features of input data and drop rate as input, and quality loss as output. The quality model can be fitted using a polynomial regression model $h(\cdot)$ of order η.
$$h(\pi, \xi_{\text{mean}}, \xi_{\text{MAE}}, \xi_{\text{RMSE}}) = \sum_{i=1}^{\eta} (\phi_{1i} \cdot \pi^i + \phi_{2i} \cdot {\xi_{\text{mean}}}^i + \phi_{3i} \cdot {\xi_{\text{MAE}}}^i + \phi_{4i} \cdot {\xi_{\text{RMSE}}}^i) + \phi_0 \quad (2)$$
In Equation 2, π is the data drop rate, while ξ_{mean}, ξ_{MAE}, and ξ_{RMSE} are the three statistical features of input data. $\{\phi_{11}, \phi_{21}, \ldots, \phi_{3\eta}, \phi_{4\eta}\}$ and ϕ_0 are regression coefficients. Based on the training data collected in steps 2 and 3, this polynomial model is fitted using the Gradient Descent method with L1 and L2 regulations [40].

3.3 The Congestion Model

The link capacity c, which is the upper limit of data communication on a single link, can be used as a congestion threshold. If the total volume of flows passing through a link exceeds c, this link is regarded as congested. Moreover, the exceeded flow volume indicates the level of congestion. For link k, originally (without any approximation), its link congestion can be written as
$$\max\left(0, \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} (r_{ijk} v_{ij}) - c\right) \quad (3)$$
where r_{ijk} is 1 if flow f_{ij} passes through link k, and 0 otherwise. That is, $r_{ijk} v_{ij} = 0$ if f_{ij} does not pass through link k, *i.e.*, this flow is not counted when calculating flow volume on this link.

By using data dropping, we define x_{ij} as the data drop rate for f_{ij}, and p_{ij} as the proportion of approximable data in f_{ij}. p_{ij} is a feature of an application, which can be profiled offline. The link congestion model is updated to be
$$c_k^{\text{link}} = \max\left(0, \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} \left(r_{ijk} \left(1 - x_{ij} p_{ij}\right) v_{ij}\right) - c\right) \quad (4)$$
where $\left(1 - x_{ij} p_{ij}\right) v_{ij}$ is the flow volume after data dropping.

3.4 The Zero Load Latency

When the network is not severely congested, the most critical factor of network performance is not the queueing latency, but the packet zero load latency. The zero load latency of a packet transmitted from nodes i to j is
$$L_{ij}^{\text{zero}} = L_{ij}^{\text{h}} + L_{ij}^{\text{s}} \quad (5)$$
where L_{ij}^{h} is the latency for the header flit to reach the destination depending on communication distance (hop count), and L_{ij}^{s} is the serialization latency, *i.e.*, the latency that the body and tail flits have to reach the destination. Thus the serialization latency can be written as
$$L_{ij}^{\text{s}} \propto \left\lceil \frac{\omega(1 - x_{ij})}{\omega^{\text{flit}}} \right\rceil \quad (6)$$
where ω is the packet size, ω^{flit} is the flit size. This serialization latency can be reduced by decreasing the number of body/tail flits.

3.5 Problem Formulation

Based on the above models, we define an optimization problem to minimize network latency by selecting an optimal data drop rate x_{ij} for each f_{ij} with $x_{ij} \in X$, while satisfying the quality requirement.

The optimization problem can be formulated as
$$\begin{aligned} &\min \sigma_1 \sum_{k=1}^{m} c_k^{\text{link}} + \sigma_2 \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} \left(L_{ij}^{\text{zero}} \cdot \frac{p_{ij} v_{ij}}{\omega}\right) \\ \Rightarrow \quad &\min \sigma_1 \sum_{k=1}^{m} \max\left(0, \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} \left(r_{ijk} \left(1 - x_{ij} p_{ij}\right) v_{ij}\right) - c\right) \\ &\quad + \sigma_2 \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} \left((1 - x_{ij}) p_{ij} v_{ij}\right), \end{aligned} \quad (7)$$
subject to

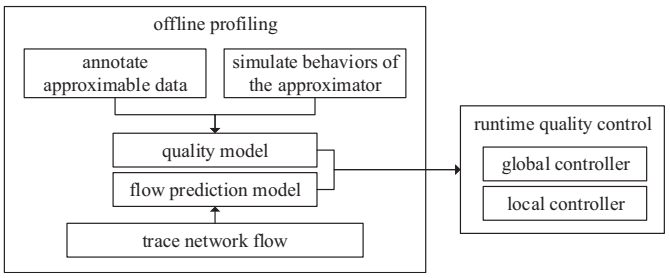


Fig. 3. An overall flow for the proposed framework.

$$\sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} (x_{ij} p_{ij} v_{ij}) \leq g \tag{8}$$

for each $x_{ij} \in X$.

Equation 7 is to minimize the accumulated congestion on all of the m links and packet zero load latency, biased by their respective weights σ_1 and σ_2. The values of σ_1 and σ_2 depend on their contributions to network performance. Equation 8 guarantees that the amount of dropped data is under the total error budget g, which is the maximally allowed amount of data to be dropped. g can be computed by Equation 1, which involves the user-defined quality requirement.

4 ACDC: The Quality Control Method

4.1 Overview

As shown in Figure 3, the proposed framework has the following steps. First, the approximable data must be annotated in the source code. At the same time, the approximator of the target system is also implemented as a software library, which can simulate the data variation caused by approximation. By offline profiling, these two together generate data for quality loss modeling. To train a flow prediction model, the target application is executed to get the network flow traces.

After the offline profiling, an error budgeting heuristic manages quality loss at runtime. The quality requirement is transformed into error budget, that is, the amount of data allowed to be dropped. Since approximating a data packet results in some data being dropped, each data approximation consumes some error budget. To guarantee quality safety, data approximation is not allowed when the error budget is insufficient. To optimize network performance by dropping data while satisfying the quality requirement, we propose a two level control system as in Figure 4.

Figure 4(a) shows the components of a node. The control system includes a global controller, together with a local controller at each node. Each network interface (NI) is equipped with a data dropper and a recoverer. A data dropper drops part of data in a packet (see Section 4.3) according to the given drop rate, before the packet is injected into the network. After receiving such an incomplete packet, the data recoverer recovers the missing data, according to the received data. Same as in [7], the packet header is extended with an approximable flag and a data type field, for network routers to identify whether a packet is approximable.

The drop rate optimization is shown in Figure 4(b). In Equation 7, the optimization objective is defined as the sum

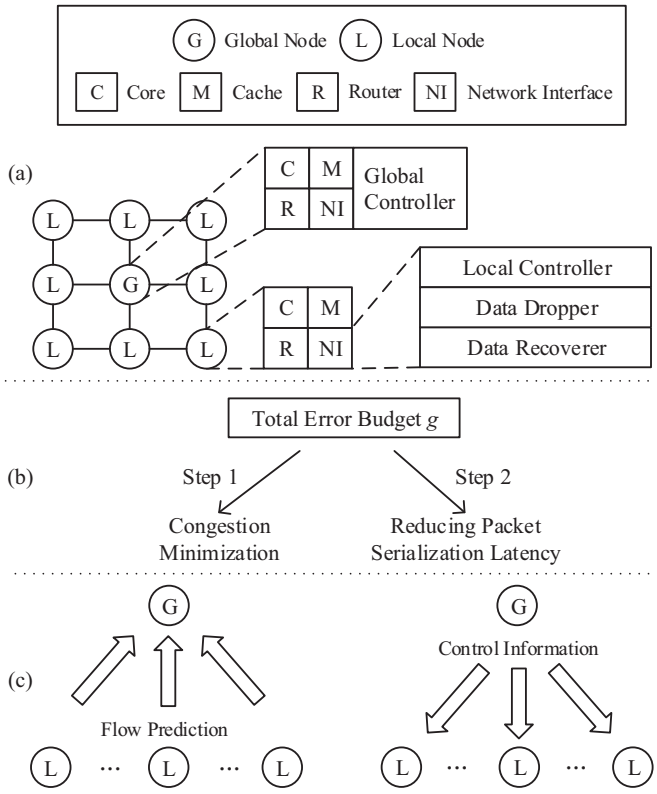


Fig. 4. An overview of ACDC. (a) Components of a node, (b) two steps of optimization, and (c) control data transmitted in the network.

of weighted combination of network congestion and zero load (serialization) latency. In most cases, as network performance is sensitive to congestion, we assume that $\sigma_1 \gg \sigma_2$, *i.e.*, congestion is the first consideration in optimization.

In the global optimization, first of all, the total error budget g is estimated to support these two steps of optimization. In step 1, an algorithm is proposed to determine a baseline setting of drop rates that minimizes network congestion. During this process, part of the total error budget is allocated since performing approximation consumes error budget. In step 2, if the error budget is not used up in step 1, the remaining error budget is assigned to the nodes. A drop rate higher than the baseline is set to further reduce packet serialization latency using this remaining error budget.

Because of the variation of application workloads in different execution phases, this runtime control is performed periodically. At time t,

1) Flow prediction (see Section 4.4) is performed in a distributed way that each local node i predicts the flow volumes at $t+1$ whose source node is node i, and sends them to the global controller node, as shown in Figure 4(c). After receiving these flow predictions, the global controller triggers the global optimization.
2) In the global optimization, first, a lightweight heuristic is performed to calculate a baseline congestion-minimizing setting of flow drop rates, and to estimate the error budget that this setting is going to consume. Part of the total error budget is allocated according to this estimation. Second, the remaining error budget is assigned to each node for

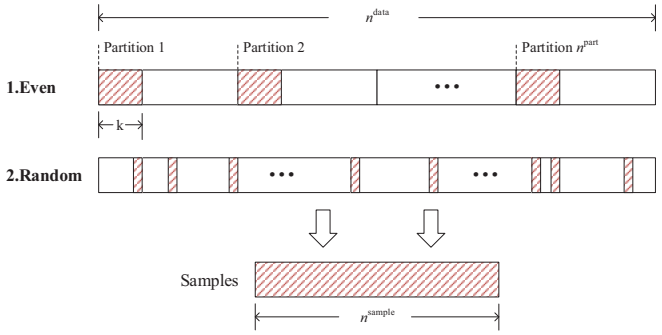


Fig. 5. Two runtime sampling methods.

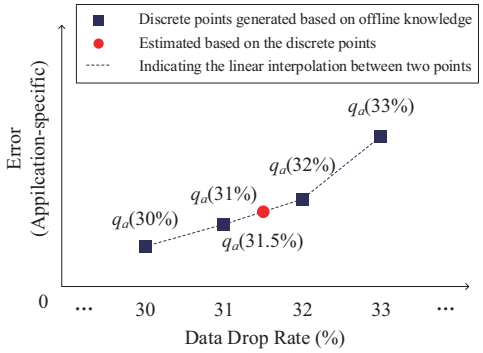


Fig. 6. An example of the resulting quality model for non-integer valued data drop rate.

further reducing serialization latency. The control information delivered to each node, as shown in Figure 4(c), includes the baseline flow drop rate setting, the error budgets for congestion minimization and serialization latency reduction.

3) The local controller of each node determines the data drop rate for each approximable data packet. The data dropper drops data according to this data drop rate, before the packet is being injected into the network.

4.2 Drop Rate Computation

Given the quality requirement θ_a and the statistical features $(\xi_{\text{mean}}, \xi_{\text{MAE}}, \xi_{\text{RMSE}})$ of input data, the maximum allowable drop rate π^* should be calculated at runtime according to the quality model. That is,

$$h(\pi^*, \xi_{\text{mean}}, \xi_{\text{MAE}}, \xi_{\text{RMSE}}) = \theta_a \tag{9}$$

To make the computation low-overhead, the following optimizations are performed. First, when the size of application input data is large, calculating the statistical features can be costly. Therefore, instead of computing over the entire input data vector, only a subset of sampled data is used. To sample n^{sample} data out from the entire n^{data} input data, two methods are used as shown in Figure 5.

1) The first method (termed as Even) divides the input data vector evenly into n^{part} partitions of the same length. For each partition, the first k values are used. In total, $n^{\text{sample}} = n^{\text{part}}k$.
2) The second method (termed as Random) collects the n^{sample} samples randomly from the input data vector.

The method Even is able to get good samples when the input data have strong locality (*e.g.*, in image data, adjacent pixels tends to be similar).

Given a quality requirement, the following steps are used to compute the maximum allowable drop rate in a low-overhead manner.

Notice that the polynomial model $h(\cdot)$ in Equation 2 can also be written as

$$h(\pi, \xi_{\text{mean}}, \xi_{\text{MAE}}, \xi_{\text{RMSE}}) = \sum_{i=1}^{\eta}(\phi_{1j} \cdot \pi^i) + \sum_{i=1}^{\eta}(\phi_{2j} \cdot {\xi_{\text{mean}}}^i + \phi_{3j} \cdot {\xi_{\text{MAE}}}^i + \phi_{4j} \cdot {\xi_{\text{RMSE}}}^i) + \phi_0 \tag{10}$$

Except $\sum_{i=1}^{\eta}(\phi_{1j} \cdot \pi^i)$, other terms do not depend on data drop rate. Therefore, those terms are grouped into a constant c_{feature}, as shown in Equation 11.

$$h(\pi, \xi_{\text{mean}}, \xi_{\text{MAE}}, \xi_{\text{RMSE}}) = \sum_{i=1}^{\eta}(\phi_{1j} \cdot \pi^i) + c_{\text{feature}} \tag{11}$$

c_{feature} is calculated once per execution since the input data do not change during execution. Now the problem is simplified to solve the following one,

$$\sum_{i=1}^{\eta}(\phi_{1j} \cdot {\pi^*}^i) = \theta_a - c_{\text{feature}} \tag{12}$$

To reduce the computation overhead at runtime, we build a lookup table $T_a(\pi)$, which takes the quantized drop rates $1\%, 2\%, \ldots, 100\%$ as input and the quality loss as output. This table is trained offline. At runtime, to solve Equation 12, the table is retrieved. Now, given the input data, the quality model $q_a(\cdot)$ is

$$q_a(\pi) = T_a(\pi) + c_{\text{feature}} \tag{13}$$

Note that for a drop rate not recorded in this table (*e.g.*, a drop rate of 3.5%), the quality loss is estimated by linear interpolation, as shown in Figure 6. Furthermore, Algorithm 1 describes that given the quality requirement, how the maximum allowable drop rate is calculated. This function can be denoted as ${q_a}^{-1}(\cdot)$, *i.e.*, the inverse function of the quality model $q_a(\cdot)$. When calculating Equation 1, the target quality loss ϵ is the given quality requirement θ_a.

4.3 Data Dropper and Data Recoverer

The proposed control mechanism is designed to exploit the full power of data dropping methods. In this paper, we

Algorithm 1: Find the target data drop rate

Input : ϵ: the target quality loss.
Output : π: the corresponding data drop rate

```
for i from 0 to 99 do
    if ε ≥ q_a(i%) and ε ≤ q_a((i+1)%) then
        π = i% + (ε − q_a(i%)) / (q_a((i+1)%) − q_a(i%));
        break;
    end
end
```

adopt the data droppers and recoverers proposed in ABDTR [8] and APPROX-NoC [7]. A brief introduction of these methods is given in this subsection.

4.3.1 Data Dropper and Recoverer in ABDTR

Figure 7 shows how data are changed during the process of approximation, *i.e.*, being dropped and then recovered. Data payload of a packet is formated as a vector of data units (*e.g.*, a 32-bit integer), denoted as $\beta_1, \beta_2, \cdots$.

The process of data dropping is similar to sampling, which skips one data unit after scanning every l data units. These skipped data units are dropped. Besides the shortened data vector, the skipping interval l is encapsulated together with other meta-data in the header flit.

After receiving a lossy packet at the destination, the data recoverer recovers the dropped data units. Inversely, it recovers the dropped data units $(\beta_{l+1}, \beta_{2l+2}, \ldots)$ based on the skipping interval l, *i.e.*, inserting one recovered data unit after every l received data units. For a dropped data unit β_i in the source node, the corresponding recovered data unit β_i' in the destination node is calculated as the average of its neighboring units

$$\beta_i' = (\beta_{i-1} + \beta_{i+1})/2 \tag{14}$$

For a special case that a data unit has only one neighbor (*e.g.*, the last data unit is dropped), the value of its neighbor is copied.

4.3.2 Data Dropper and Recoverer in APPROX-NoC

APPROX-NoC is based on frequent pattern compression, which replaces the hardcoded frequently-appearing data patterns by shortened words. An overview of APPROX-NoC is shown in Figure 8, which summarizes how data are processed by the data dropper.

For floating-point data, only the mantissa part is extracted and word-completed by filling 0's in the most significant bits. The sign and exponent bits are kept unchanged. For data units larger than a word (double, long, *etc.*), they are processed word-by-word.

After that, the word is processed by the approximate logic. Before doing the frequent pattern compression, the original data pattern should be transformed into an approximate pattern with some don't care bits (wildcards in the least significant bits). Given the error threshold, the number of don't care bits is $\lfloor \log_2(\text{value} \times \text{error\_threshold}) \rfloor$, based on the value of the word. The approximate pattern is formed once the number of don't care bits is determined. Then the frequent pattern compression is done by sequentially matching the approximate pattern to one of the patterns predefined in Table 2, and replacing it into the corresponding pattern-indexing and compressed data bits.

TABLE 2
Frequent Pattern Compression in APPROX-NoC

Index	Pattern encoded	Data size
000	Zero run	3 bits
001	4-bit sign-extended	4 bits
010	One byte sign-extended	8 bits
011	Halfword sign-extended	16 bits
100	Halfword padded with a zero halfword	16 bits
101	Two halfwords, each a byte sign-extended	16 bits
111	Uncompressed word	32 bits

Data recovery is performed in reverse. It scans the 3-bit indices serially. For each of them, the original data are recovered according to its corresponding pattern in Table 2.

4.4 Flow Prediction

To predict flow volumes at the next time unit, they are treated as time series variables, which can be predicted from history.

In this paper, the autoregressive (AR) model is used. An AR model is

$$y^{\text{AR}}(t) = \lambda_1 y_{t-1} + \lambda_2 y_{t-2} + \cdots + \lambda_b y_{t-b} + \mu + \lambda_0 \tag{15}$$

where $y^{\text{AR}}(t)$ is the flow volume prediction based on b previously observed values. y_t is the actual flow volume at time t. The model order b is selected using Bayesian information criterion (BIC) [41].

The AR model is trained offline. In the offline profiling, training samples are collected by monitoring the traffic of each network flow at each time unit. Then the parameters are estimated using the least square method [40].

Furthermore, the flow prediction is corrected using a runtime feedback,

$$\begin{aligned} \hat{y}_t &= y^{\text{AR}}(t) + \Delta_t, \\ \Delta_t &= \Delta_{t-1} + \alpha(y_{t-1} - \hat{y}_{t-1}) \end{aligned} \tag{16}$$

where α is the learning rate, and Δ_t is the feedback term.

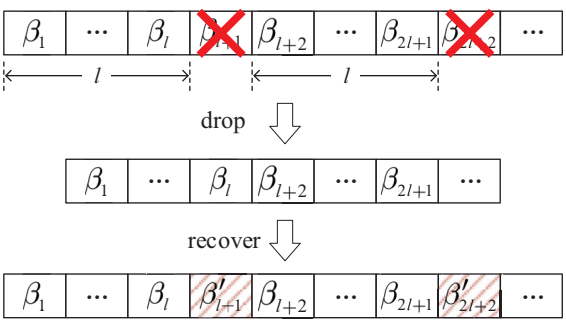


Fig. 7. How a data vector changes during approximation in ABDTR.

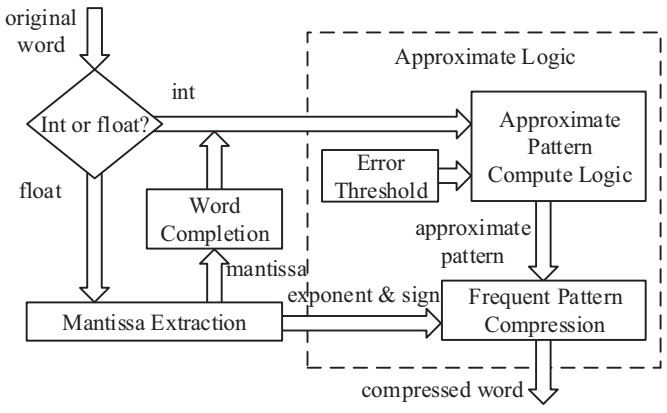


Fig. 8. The lossy compression mechanism used in APPROX-NoC.

4.5 The Global Controller

First, given the total error budget, the global controller finds a congestion-minimizing setting of flow drop rates. At the same time, it estimates and allocates the error budget for serialization latency reduction. Second, the remaining error budget is assigned to each node, to further reduce serialization latency.

4.5.1 Congestion Minimization

Algorithm 2: Network congestion minimization

Input : $\{v_{11}, v_{12}, \ldots, v_{nn-1}\}$: a set of flow volumes.
Output : $\{x_{11}, x_{12}, \ldots, x_{nn-1}\}$: a set of flow data drop rates.
Function: Find a flow drop rate setting that minimizes congestion.

1 Initialize each x_{ij} to be 0;
2 Initialize total error budget g (see Equation 1);
3 Initialize each c_k^{link} to be $\max\left(0, \sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\left(r_{ijk}v_{ij}\right) - c\right)$;
4 Initialize each c_{ij}^{flow} to be $\sum_{k=1}^{m}\left(r_{ijk}c_k^{\text{link}}\right)$;
5 Initialize each e_{ij} to be 1;
6 $(i^*, j^*) = \arg\max_{(i,j)}\left(e_{ij}c_{ij}^{\text{flow}}\right)$;
7 **while** $e_{i^*j^*} = 1$ **and** $c_{i^*j^*}^{\text{flow}} > 0$ **and** $g > 0$ **do**
8 $k^* = \arg\max_k\left(r_{i^*j^*k}c_k^{\text{link}}\right)$;
9 $x_{i^*j^*} = \min\left(\pi_d, \min\left(c_{k^*}^{\text{link}}, g\right) / \left(p_{i^*j^*}v_{i^*j^*}\right)\right)$;
10 $e_{i^*j^*} = 0$;
11 $g = g - x_{i^*j^*}p_{i^*j^*}v_{i^*j^*}$;
12 **for** k' **that** $r_{i^*j^*k'} = 1$ **do**
13 $c_{k'}^{\text{link}} = \max\left(0, c_{k'}^{\text{link}} - x_{i^*j^*}p_{i^*j^*}v_{i^*j^*}\right)$;
14 **for** (i,j) **that** $r_{ijk'} = 1$ **and** $e_{ij} = 1$ **do**
15 $c_{ij}^{\text{flow}} = \sum_{k=1}^{m}\left(r_{ijk}c_k^{\text{link}}\right)$;
16 **end**
17 **end**
18 $(i^*, j^*) = \arg\max_{(i,j)}\left(e_{ij}c_{ij}^{\text{flow}}\right)$;
19 **end**

The decision space of the optimization problem formulated in Section 3.5 is $O\left(|X|^{n(n-1)}\right)$, since there are $n\,(n-1)$ variables (one for each flow) and $x_{ij}{\in}X$.

When the network size scales up, the optimal decision cannot be efficiently found at runtime.

We propose a lightweight heuristic to solve the optimization problem in Section 3.5, as detailed in Algorithm 2. The input of this algorithm is the predicted flow volumes, and the output is the congestion-minimizing drop rates for each f_{ij}. We define two vectors, c^{link} and c^{flow}, to indicate congestion statuses of the flows and links, respectively. Initially, the drop rates are set to be 0. After that, the total error budget is initialized, and congestion of each link is calculated by Equation 3. The congestion of each flow is initialized as the sum of the link congestions along its path. Each flow f_{ij} is associated with a label e_{ij} indicating whether it is unprocessed or not. As in line 5, those labels are initialized to be 1 (unprocessed).

After initialization, drop rates for the flows are set iteratively. Before each iteration, the most congested flow $f_{i^*j^*}$ is selected from the unprocessed ones (see line 6). The most congested link k^* along the path of this selected flow is figured out (see line 8).

For $f_{i^*j^*}$, if the exceeded flow volume (*i.e.*, the level of congestion, as in Section 3.3) equals to the amount of dropped data, *i.e.*, $c_{k^*}^{\text{link}} = x_{i^*j^*}p_{i^*j^*}v_{i^*j^*}$, the drop rate $x_{i^*j^*} = c_{k^*}^{\text{link}} / \left(p_{i^*j^*}v_{i^*j^*}\right)$ can be set to offset this congestion. However, the amount of dropped data must be smaller than the error budget, thus $x_{i^*j^*} = \min\left(c_{k^*}^{\text{link}}, g\right) / \left(p_{i^*j^*}v_{i^*j^*}\right)$. This drop rate cannot be larger than the maximum drop rate π_d. Therefore, $x_{i^*j^*} = \min\left(\pi_d, \min\left(c_{k^*}^{\text{link}}, g\right) / \left(p_{i^*j^*}v_{i^*j^*}\right)\right)$ as in line 9.

Next, $f_{i^*j^*}$ is marked as processed, and the available error budget g is updated to be $g - x_{i^*j^*}p_{i^*j^*}v_{i^*j^*}$, *i.e.*, subtracting the amount of dropped data estimated in this iteration. Since the flow volume is reduced, all the links along this flow are affected, and so are the flows passing through those links. Congestion statuses of the affected links and flows are updated, except the flows already processed. The loop terminates when there is no unprocessed flow, or the congestion is eliminated, or the error budget is used up. By analyzing the innermost loop, the worst case time complexity of Algorithm 2 is $O(mn^3)$.

Denote the estimated error budget allocated for congestion minimization as μ, and the portion of μ assigned to node i as μ_i. μ_i can be calculated as

$$\mu_i = \sum_{j=1,i\neq j}^{n}\left(x_{ij}p_{ij}v_{ij}\right) \tag{17}$$

which is the estimated amount of dropped data for all flows starting from node i, when using the congestion-minimizing flow drop rates for approximation. μ is the sum of all μ_i's

$$\mu = \sum_{i=1}^{n}\mu_i \tag{18}$$

4.5.2 Reducing Serialization Latency

Denote the remaining error budget as ν, and the portion of ν assigned to node i as ν_i. After running Algorithm 2, ν is calculated as

$$\nu = g - \mu \tag{19}$$

If ν is larger than zero, higher drop rates can be used to further reduce serialization latency. We define ρ_i as an indicator of the ability to drop more data for node i. It is measured by

$$\rho_i = \sum_{j=1,i\neq j}^{n}\left(\left(\pi_d - x_{ij}\right)p_{ij}v_{ij}\right) \tag{20}$$

For each flow f_{ij}, x_{ij} is a congestion-minimizing drop rate. The gap between the maximum drop rate π_d and the congestion-minimizing drop rate x_{ij} is $\pi_d - x_{ij}$. By multiplying the approximable flow volume $p_{ij}v_{ij}$, the value of $\left(\pi_d - x_{ij}\right)p_{ij}v_{ij}$ indicates the potential to perform approximation more aggressively on flow f_{ij}. ρ_i is the sum of them for all flows starting from node i. In this way, ν_i is calculated as

$$\nu_i = \nu \cdot \frac{\rho_i}{\sum_{j=1}^{n}\rho_j} \tag{21}$$

The global controller then sends control information, *i.e.*, μ_i, ν_i and $\{x_{i1}, x_{i2}, \ldots, x_{in}\}$, to node i. At each node, the

local controller dynamically decides the final data drop rate for each incoming data packet, based on the received control information.

4.6 The Local Controllers

Algorithm 3: Local controller in node i

Input : j: the destination of the incoming packet.
Output : γ: the data drop rate for the incoming packet.
Function: Decide the drop rate γ for the incoming packet.

```
Initialize γ to be 0;
if μ_i > 0 then
    μ_i = μ_i − x_ij ω;
    γ = x_ij;
end
if ν_i > 0 then
    ν_i = ν_i − (π_d − γ) ω;
    γ = π_d;
end
```

In node i, after receiving control information from the global controller, the two local error budgets (μ_i and ν_i) and congestion-minimizing drop rates for its $n-1$ flows are updated to be the values received from the global controller. The local controller decides the data drop rate for injecting an incoming approximable packet, corresponding to a reduction in error budget. If the error budget is reduced to 0, approximation is not allowed until the error budget is updated to be non-zero, to guarantee quality safety.

The local controller in node i works as in Algorithm 3. For an approximable packet being sent to node j, the process of local control consists of the following two steps:

1) A temporary drop rate γ is initialized to be 0. If the error budget μ_i is larger than 0, γ is temporarily set to be the congestion-minimizing setting x_{ij}. The amount of dropped data $x_{ij}\omega$ must be subtracted from μ_i, where ω is the size of each data packet.
2) If the error budget ν_i is larger than 0, the maximum drop rate π_d is used for injecting this packet. ν_i is decreased by $(\pi_d - \gamma)\,\omega$, where $(\pi_d - \gamma)$ is the increment of data drop rate.

4.7 Handling Bursty Communications

Let y_{ij}^t and $\hat{y}_{ij}^t$ denote the actual and predicted flow volumes of flow f_{ij} at time t, respectively. Figure 9 shows an example of the two values for the benchmark Sobel. One can see from Figure 9 that, the flow prediction model is erroneous when 1) y_{ij}^{t-1} is low, and 2) y_{ij}^t is high (bursty communication).

1) Before time $t-1$, the flow volume increases steadily. Therefore, the model keeps predicting the increasing trend at time $t-1$, as shown in Figure 9. It fails to respond to the sudden decrease of y_{ij}^{t-1}, as indicated by the red shadowed area. The predicted flow volume $\hat{y}_{ij}^{t-1}$ is erroneous if $\hat{y}_{ij}^{t-2} - \hat{y}_{ij}^{t-1} < 0$ and $\frac{y_{ij}^{t-2} - y_{ij}^{t-1}}{y_{ij}^{t-2}} > 0.5$, *i.e.*, when $y_{ij}^{t-4}, y_{ij}^{t-3}, y_{ij}^{t-2}$ keep increasing but y_{ij}^{t-1} suddenly decreases.

Algorithm 4: Handling bursty communications in the global controller

Input : $\{\hat{y}_{11}^{t-1}, \hat{y}_{12}^{t-1}, \ldots, \hat{y}_{nn-1}^{t-1}\}$: the predicted volume of each flow at time $t-1$.
$\{\hat{y}_{11}^{t-2}, \hat{y}_{12}^{t-2}, \ldots, \hat{y}_{nn-1}^{t-2}\}$: the predicted volume of each flow at time $t-2$.
$\{y_{11}^{t-1}, y_{12}^{t-1}, \ldots, y_{nn-1}^{t-1}\}$: the actual volume of each flow at time $t-1$.
$\{y_{11}^{t-2}, y_{12}^{t-2}, \ldots, y_{nn-1}^{t-2}\}$: the actual volume of each flow at time $t-2$.
Output : $\{\mu_1^{\text{burst}}, \mu_2^{\text{burst}}, \ldots, \mu_n^{\text{burst}}\}$: bursty budget assigned to each node.
Function: Calculate the bursty budget assigned to each node.

```
/* After initializing the total error budget g */
for i from 1 to n do
    for j from 1 to n (i ≠ j) do
        μ_i^burst = 0;
        if ŷ_ij^{t-2} − ŷ_ij^{t-1} < 0 and (y_ij^{t-2} − y_ij^{t-1}) / y_ij^{t-2} > 0.5 then
            μ_i^burst = z^burst g / n;
            g = g − μ_i^burst;
            break;
        end
    end
end
```

2) Due to the runtime feedback correction, at time t, $\hat{y}_{ij}^t$ decreases due to the decrease of y_{ij}^{t-1} at time $t-1$. However, y_{ij}^t becomes high (bursty communication) and thus the flow prediction model is erroneous again, as indicated by the blue shadowed area.

In condition 1, a drop rate higher than the optimal one may be used. However, y_{ij}^{t-1} is low and the influence on quality loss is trivial. On the contrary, the influence of condition 2 on performance is non-negligible, since y_{ij}^t is high.

Therefore, an enhancement is made to handle bursty communications. If condition 1 occurs at a node, part of the total error budget, termed as "bursty budget", is assigned to it. It leads to decreases in others' error budgets. Since the total bursty error budget is bounded, the impact is negligible

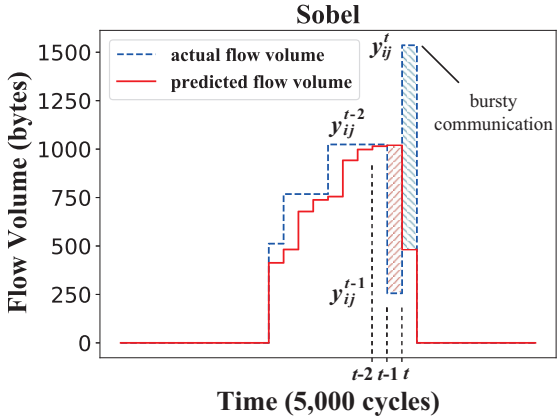


Fig. 9. Flow changes in the benchmark Sobel.

TABLE 3
System Configuration

System Parameters	
Number of processors	64 (Alpha 21264 ISA)
Fetch/Decode/Commit size	4 / 4 / 4
L1 D cache (private)	16 KB, two-way, 32B line, two cycles, two ports, dual tags
L1 I cache (private)	32 KB, two-way, 64B line, two cycles
L2 cache (shared) MESI protocol	64 KB slice/node, 128B line, six cycles, two ports
Main memory size	2 GB, latency 200 cycles
NoC Parameters	
NoC flit size	48 bits
Data packet size	15 flits, 28 flits
Control packet size	2 flits
NoC latency	two cycles for router, one cycle for link
Virtual channel	2 VCs/VN, 3 VNs
NoC buffer	5×12 flits
Routing algorithm	XY routing

TABLE 4
Benchmark Configuration

Application	Quality Metric	QR1	QR2
Blackscholes	Average relative error of output prices	0.05	0.1
Swaptions	Average relative error of the output prices	0.2	0.5
Canneal	Relative error of the final routing cost	0.01	0.05
Ferret	Percentage of images not being searched out in original results	0.2	0.5
Fluidanimate	Percentage of particles not in same cell as in original results	0.2	0.5
Sobel	Average relative error of the output images	10	20

on other nodes. Therefore, it leads to little or no impact if condition 2 (bursty communication) does not happen. However, if condition 2 does occur, the bursty budget can help mitigate the high communication workloads.

The proposed method is shown in Algorithm 4, which calculates the bursty budget assigned to each node. Let μ_i^{burst} be the bursty budget assigned to node i. Parameter z^{burst} is the maximum proportion of the total error budget g in bursty budget allocation (thus $z^{\text{burst}}g$ is the maximum amount). For node i, if a flow f_{ij} satisfies $\hat{y}_{ij}^{t-2} - \hat{y}_{ij}^{t-1} < 0$ and $\frac{y_{ij}^{t-2} - y_{ij}^{t-1}}{y_{ij}^{t-2}} > 0.5$, the global controller assigns bursty budget to it (see line 5). Since there are n nodes in the system, the bursty budget assigned to a node is set to be $\frac{z^{\text{burst}}g}{n}$, indicating $z^{\text{burst}}g$ is divided equally for the n nodes (see line 6). After that, g is updated to be $g - \mu_i^{\text{burst}}$ (see line 7).

Before being sent to node i, the error budget μ_i is updated by adding the assigned bursty budget

$$\mu_i = \mu_i + \mu_i^{\text{burst}} \tag{22}$$

5 Evaluation

5.1 Experimental Setup

In this section, our proposed control method ACDC augments the data droppers and recoverers in ABDTR [8] and APPROX-NoC (FPVAXX) [7]. Experiments were designed to compare ABDTR+ACDC and FPVAXX+ACDC with the original ABDTR and FPVAXX. Experiments were evaluated using a cycle-accurate full-system many-core simulator [6], with system configurations listed in Table 3. The simulator consists of two models, i.e., a functional model and a timing model. The functional model guarantees data correctness during the simulation and thus the program executes correctly, while the timing model estimates the target architecture performance. To evaluate how approximations influence application output, in the functional model, lossy outputs are generated under different drop rates by dropping data/variables and recovering them, which is similar to the offline profiling. In the timing model, the corresponding data packets are dropped and the timing impact of data packet dropping is estimated. This drop rate connects the two models and thus a corresponding lossy output can be given after the simulation.

Six applications selected from PARSEC [42] and AxBench [43] are used for evaluation. The quality metrics and quality requirement (QR) settings of these applications are listed in Table 4. Two quality requirement settings are used for evaluation, with QR1 being strict and QR2 loose. These QR settings are not used when evaluating the original FPVAXX method since it is not able to estimate the application-specific output error. Instead, as the counterparts of QR1 and QR2, error thresholds of FPVAXX are set to be 0.2 and 0.4, respectively.

To build the quality model, an input dataset must be provided for each application.

- For Sobel, the dataset from AxBench is used.
- For each of the other applications, the "simnative" input data from PARSEC are divided into 100 instances, each of which has a size of "simlarge". As a result, a dataset of 100 instances is created.

70% of the instances are used for building the quality model, as stated in Section 3.2.

Parameters in ACDC for each application are determined by offline profiling. Besides, we tested the performance of ACDC by varying the length of control time unit to be 2000, 5000, and 10000 cycles. The results show that, using different lengths of control time unit leads to no significant

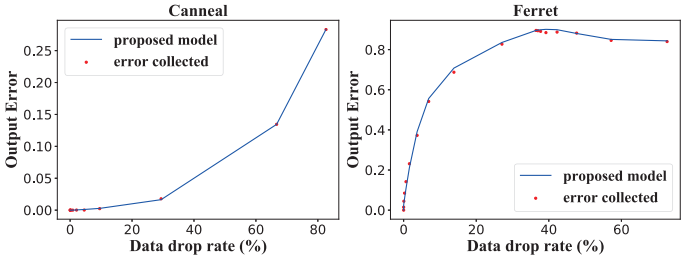


Fig. 10. The quality loss estimations for applications Canneal and Ferret.

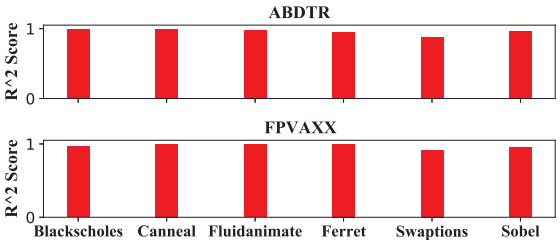


Fig. 11. R^2 score of the proposed model for the methods ABDTR and FPVAXX.

difference in performance. Thus, it is set to be 10000 cycles in the following experiments due to the low overhead.

5.2 Experimental Results

The quality loss estimations on the test data for applications Canneal and Ferret are depicted in Figure 10, using the FPVAXX approximator. Figure 10 shows that the proposed quality model fits these test data well, even if they are highly non-linear. As shown in Figure 11, we have presented the R^2 score [40] of the quality model for all the applications, where an R^2 score close to 1 indicates lower regression error. On average, the R^2 scores of the proposed quality model for ABDTR and FPVAXX are 0.963 and 0.971, respectively, which indicates good data fitting.

For each application, the execution time, energy consumption, and average latency are normalized to those of ABDTR in Figures 12-13, and to FPVAXX in Figures 14-15. Besides, the quality loss is also normalized. In each figure, the dashed line in quality loss evaluation is the quality requirement, which is set to be baseline. Quality requirement is not satisfied if a bar goes beyond the dashed line.

Figures 12 and 13 compare ABDTR with ACDC_ABDTR in terms of execution time, energy consumption, average latency, and quality loss. When the quality requirement is QR1, on average, ACDC accelerates execution by 8.79% and reduces energy consumption by 5.95% over ABDTR. For QR2, on average, ACDC accelerates execution by 13.90% and reduces energy consumption by 13.23% over ABDTR. For some applications (*e.g.*, Sobel), ACDC achieves as much as 29.42% execution speedup over ABDTR.

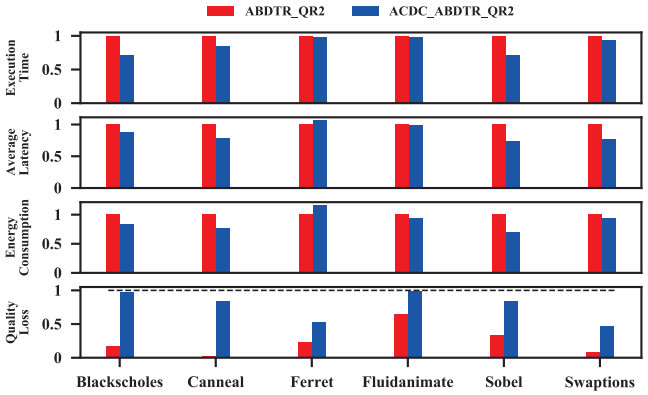


Fig. 13. Comparison of ABDTR_QR2 and ACDC_ABDTR_QR2. ABDTR is the original implementation in [8], and ACDC_ABDTR augments ABDTR with ACDC, evaluated under quality requirement QR2.

The main reason is that ABDTR suffers from approximation underutilization, *i.e.*, the resulting quality loss is much lower than is allowed in all cases. On the other hand, ACDC_ABDTR performs approximation more aggressively. The reason is that the per-packet quality control of ABDTR only guarantees quality safety for each approximation, whereas the quality control of ACDC optimizes performance based on global network status.

In some applications (*e.g.*, Ferret), approximation is not fully exploited. The reasons are as follows:

1) Since these applications are sensitive to input data loss, they have low error tolerance. Take Ferret as an example. In Ferret, instead of using the original image, the pre-calculated image features are used as input data. These data, however, have more critical information than the original image. Such input data also show lower locality, *i.e.*, similarity in consecutive data units, which influences the performance of the ABDTR approximator negatively. Therefore, Ferret is sensitive to input data loss.
2) Some applications do not suffer from severe NoC congestion, *e.g.*, Swaptions. There is less opportunity to improve network performance by the global optimization, while the extra overhead must be paid.

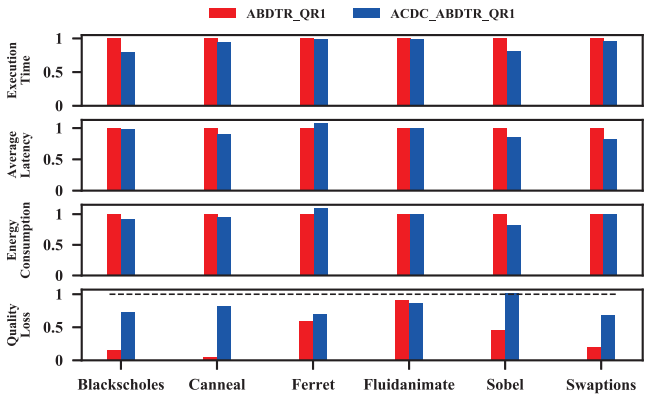


Fig. 12. Comparison of ABDTR_QR1 and ACDC_ABDTR_QR1. ABDTR is the original implementation in [8], and ACDC_ABDTR augments ABDTR with ACDC, evaluated under quality requirement QR1.

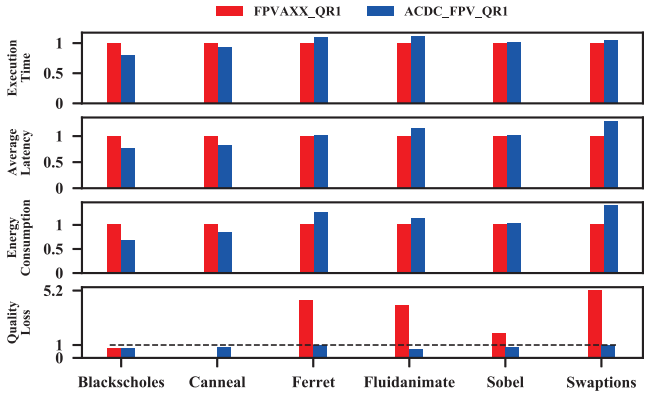


Fig. 14. Comparison of FPVAXX_QR1 and ACDC_FPV_QR1. FPVAXX is the original implementation in [7], and ACDC_FPV augments FPVAXX with ACDC, evaluated under quality requirement QR1.

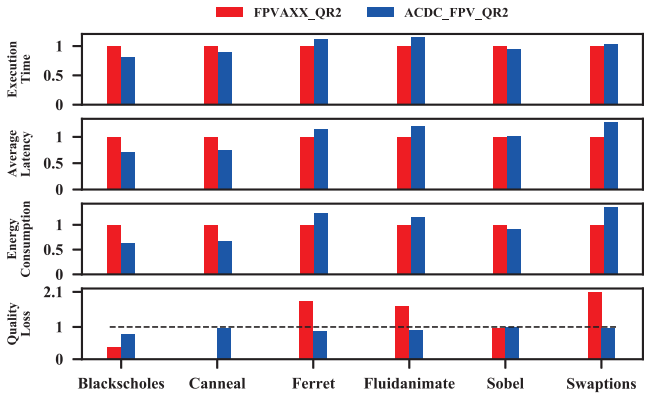


Fig. 15. Comparison of FPVAXX_QR2 and ACDC_FPV_QR2. FPVAXX is the original implementation in [7], and ACDC_FPV augments FPVAXX with ACDC, evaluated under quality requirement QR2.

3) Another factor is the approximator, *i.e.*, the compression method itself. In the ABDTR approximator (see Section 4.3), the highest data drop rate provided by this approximator is 50%. Even if the quality requirement is extremely loose, the quality control method can only approximate less than 50% of data.

Figures 14 and 15 compare FPVAXX with ACDC_FPV. A severe problem in APPROX-NoC is over-approximation, *i.e.*, it fails to satisfy the quality requirement. It is mainly caused by these reasons:

1) FPVAXX does not have an application-specific output error model. It drops data according to the input data error only, without estimating the output quality loss. With a quality model, our proposed method guarantees quality safety.
2) When approximating floating point data, a small change in mantissa leads to a large variation in the data value. However, when deciding the number of approximate bits, FPVAXX considers the error in mantissa only. That is why Ferret, Fluidanimate (single-precision), and Swaptions (double-precision) are less tolerable to approximation. Even though error-resilience is low in these applications, our proposed method does not violate the given quality requirements.

For applications whose quality requirements are satisfied by both methods, on average, ACDC reduces execution time and energy consumption by 13.75% and 23.25% over FPVAXX under QR1, respectively. Similarly, on average, ACDC improves execution time and energy consumption by 11.65% and 26.43% over FPVAXX under QR2. For Blackscholes, ACDC achieves as much as 20.44% execution speedup over FPVAXX.

The sensitivity of flit size is studied. The performance evaluation under QR2 is performed with different flit sizes. ABDTR is chosen as the baseline in normalization. In this new experiment, different flit sizes are evaluated to show how flit size influences our proposed method. The experimental results are shown in Figure 16. These results indicate that ACDC achieves similar performance improvement on both longer (32 bits) and shorter packets (64 bits and 80 bits).

Moreover, an experiment is performed to evaluate congestion change caused by approximate communication. Congestion of a packet is measured by $1 - \frac{L^{\text{zero}}}{L^{\text{total}}}$, where L^{zero} is the zero load latency and L^{total} is the total latency. As shown in Figure 17, there are two configurations for application Sobel, *i.e.*, with and without approximate communication, termed as "approximate" and "lossless", respectively. The average congestion metric of finished packets is calculated every 10,000 cycles. The approximate execution uses ACDC_ABDTR and QR2. One can see from Figure 17 that congestion in the approximate execution is mitigated, compared to the lossless execution.

To see the impacts on actual output, Figure 18 shows the edge detection results from Sobel with accurate (lossless) execution, and by running ACDC_ABDTR (approximate) under QR1 and QR2, respectively. The object edges are recognizable in the two approximate outputs. They show no significant difference from the accurate one.

5.3 Overhead

The global controller is implemented as software codes running on the global controller core, whereas each local controller is designed as a hardware circuit.

The energy overhead of the ACDC control information transmission is found to be 6.92% of the total system energy consumption, on average. As the experiments have shown, this energy overhead is offset by the energy reduction obtained by our proposed optimization. The average execution time of the global controller is 419 cycles, which is much shorter than the length of control time unit.

According to the synthesis result reported by Synopsys Design Compiler and PrimePower targeting a 45 nm TSMC

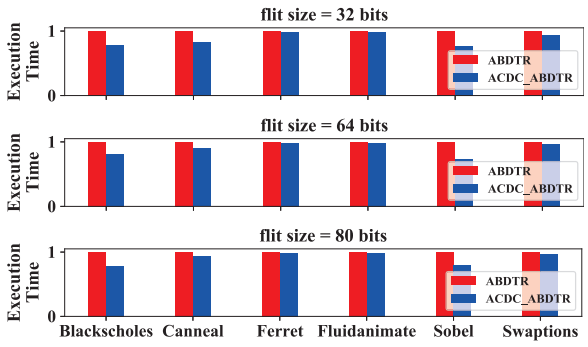


Fig. 16. Performance comparison when flit size is set to be different values.

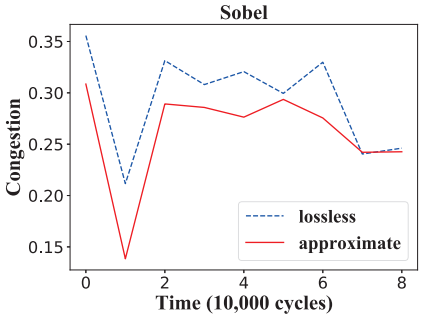
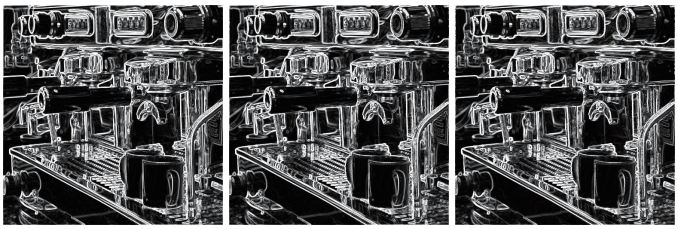


Fig. 17. Evaluation of congestion changes in lossless and approximate executions.



(a) (b) (c)

Fig. 18. The resulting approximated outputs of Sobel. (a) The accurate (lossless) result, (b) approximate result under QR1, and (c) approximate result under QR2.

library, a local controller has an area of 968 μm^2 and power consumption of 0.636 mW. Using DSENT [44] with a 45 nm CMOS technology, a router with the configurations in Table 3 consumes 145 mW power, and its area is 462,965 μm^2. The power and area of an ACDC controller are only 0.44% and 0.21% of a router, respectively. Therefore, the overhead of ACDC is low.

6 Conclusion

In this paper, we proposed an accuracy- and congestion-aware dynamic traffic control (ACDC) method for approximate NoCs. A network performance optimization problem for approximate NoCs is formulated. To solve this problem, a lightweight mechanism is proposed. An application quality model and a flow prediction model are built by offline profiling. At runtime, based on the models, ACDC manages quality loss through error budgeting. Congestion-aware data approximations are performed according to the predicted flow volumes. Experimental evaluations confirm that, ACDC significantly outperforms two state-of-the-art approaches in terms of both performance and energy consumption. ACDC improves approximation underutilization/overutilization existed in these approaches.

References

[1] Y. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nguyen, and M. Smelyanskiy, "Convergence of recognition, mining, and synthesis workloads and its implications," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 790–807, 2008.

[2] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate Computing: a survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2016.

[3] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 48–54, 2013.

[4] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. C. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 124–134, 2011.

[5] J. S. Miguel, M. Badr, and N. D. E. Jerger, "Load value approximation," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pp. 127–139, 2014.

[6] X. Wang, M. Yang, Y. Jiang, P. Liu, M. Daneshtalab, M. Palesi, and T. Mak, "On self-tuning networks-on-chip for dynamic network-flow dominance adaptation," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 2s, pp. 73:1–73:21, 2014.

[7] R. Boyapati, J. Huang, P. Majumder, K. H. Yum, and E. J. Kim, "APPROX-NoC: a data approximation framework for network-on-chip architectures," in *Proceedings of the International Symposium on Computer Architecture*, pp. 666–677, 2017.

[8] L. Wang, X. Wang, and Y. Wang, "ABDTR: approximation-based dynamic traffic regulation for networks-on-chip systems," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 153–160, 2017.

[9] Y. Chen and A. Louri, "An online quality management framework for approximate communication in network-on-chips," in *Proceedings of the ACM International Conference on Supercomputing*, pp. 217–226, 2019.

[10] S. Xiao, X. Wang, M. Palesi, A. K. Singh, and T. Mak, "ACDC: an accuracy- and congestion-aware dynamic traffic control method for networks-on-chip," in *Proceedings of the Design, Automation and Test in Europe*, pp. 630–633, 2019.

[11] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proceedings of the Design Automation Conference*, pp. 120:1–120:6, 2015.

[12] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. A. Mahlke, "SAGE: self-tuning approximation for graphics engines," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pp. 13–24, 2013.

[13] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: approximate data types for safe and general low-power computation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 164–174, 2011.

[14] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pp. 449–460, 2012.

[15] A. Yazdanbakhsh, B. Thwaites, H. Esmaeilzadeh, G. Pekhimenko, O. Mutlu, and T. C. Mowry, "Mitigating the memory bottleneck with approximate load value prediction," *IEEE Design & Test*, vol. 33, no. 1, pp. 32–42, 2016.

[16] J. S. Miguel, J. Albericio, A. Moshovos, and N. D. E. Jerger, "Doppelgänger: a cache for approximate computing," in *Proceedings of the International Symposium on Microarchitecture*, pp. 50–61, 2015.

[17] M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *Proceedings of the Design, Automation and Test in Europe*, pp. 1327–1332, 2016.

[18] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Proceedings of the Design, Automation and Test in Europe*, pp. 950–955, 2011.

[19] Y. Wang, J. Deng, Y. Fang, H. Li, and X. Li, "Resilience-aware frequency tuning for neural-network-based approximate computing chips," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 10, pp. 2736–2748, 2017.

[20] S. Mitra, M. K. Gupta, S. Misailovic, and S. Bagchi, "phase-aware optimization in approximate computing," in *Proceedings of the International Symposium on Code Generation and Optimization*, pp. 185–196, 2017.

[21] A. Farrell and H. Hoffmann, "MEANTIME: achieving both minimal energy and timeliness with approximate computing," in *Proceedings of USENIX Annual Technical Conference*, pp. 421–435, 2016.

[22] D. S. Khudia, B. Zamirai, M. Samadi, and S. A. Mahlke, "Rumba: an online quality management system for approximate computing," in *Proceedings of the International Symposium on Computer Architecture*, pp. 554–566, 2015.

[23] H. Hoffmann, "CoAdapt: predictable behavior for accuracy-aware applications running on power-aware systems," in *Proceedings of the Euromicro Conference on Real-Time Systems*, pp. 223–232, 2014.

[24] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. C. Rinard, "Dynamic knobs for responsive power-aware computing," in *Proceedings of the international conference on architectural support for programming languages and operating systems*, pp. 199–212, 2011.

[25] V. K. Chippa, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Dynamic effort scaling: managing the quality-efficiency tradeoff," in *Proceedings of the Design Automation Conference*, pp. 603–608, 2011.

[26] M. T. Teimoori, M. A. Hanif, A. Ejlali, and M. Shafique, "AdAM: adaptive approximation management for the non-volatile memory hierarchies," in *Proceedings of the Design, Automation and Test in Europe*, pp. 785–790, 2018.

[27] T. Wang, Q. Zhang, N. S. Kim, and Q. Xu, "On effective and efficient quality management for approximate computing," in

Proceedings of the International Symposium on Low Power Electronics and Design, pp. 156–161, 2016.
[28] C. Xu, X. Wu, W. Yin, Q. Xu, N. Jing, X. Liang, and L. Jiang, "On quality trade-off control for approximate computing using iterative training," in *Proceedings of the Design Automation Conference*, pp. 52:1–52:6, 2017.
[29] D. Mahajan, A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh, "Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration," in *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pp. 66–77, 2016.
[30] J. Park, E. Amaro, D. Mahajan, B. Thwaites, and H. Esmaeilzadeh, "AxGames: towards crowdsourcing quality target determination in approximate computing," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 623–636, 2016.
[31] X. Wang, B. Zhao, T. Mak, M. Yang, Y. Jiang, and M. Daneshtalab, "On fine-grained runtime power budgeting for networks-on-chip systems," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2780–2793, 2016.
[32] J. Ng, X. Wang, A. K. Singh, and T. Mak, "Defragmentation for efficient runtime resource management in NoC-based many-core systems," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, no. 11, pp. 3359–3372, 2016.
[33] L. Wang, X. Wang, and T. Mak, "Adaptive routing algorithms for lifetime reliability optimization in network-on-chip," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2896–2902, 2016.
[34] Z. Li, J. S. Miguel, and N. D. E. Jerger, "The runahead network-on-chip," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, pp. 333–344, 2016.
[35] A. B. Ahmed, D. Fujiki, H. Matsutani, M. Koibuchi, and H. Amano, "AxNoC: low-power approximate network-on-chips using critical-path isolation," in *Proceedings of the IEEE/ACM International Symposium on Networks-on-Chip*, pp. 6:1–6:8, 2018.
[36] V. Y. Raparti and S. Pasricha, "DAPPER: data aware approximate NoC for GPGPU architectures," in *Proceedings of the IEEE/ACM International Symposium on Networks-on-Chip*, pp. 7:1–7:8, 2018.
[37] G. Ascia, V. Catania, S. Monteleone, M. Palesi, and D. Patti, "Improving energy consumption of NoC based architectures through approximate communication," in *Proceedings of the Mediterranean Conference on Embedded Computing*, pp. 1–4, 2018.
[38] G. Ascia, V. Catania, S. Monteleone, M. Palesi, D. Patti, and J. Jose, "Approximate wireless networks-on-chip," in *Proceedings of the Conference on Design of Circuits and Integrated Systems*, pp. 1–6, 2018.
[39] G. Ascia, V. Catania, S. Monteleone, M. Palesi, D. Patti, J. Jose, and V. Salerno, "Exploiting data resilience in wireless network-on-chip architectures," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 16, no. 2, pp. 1–27, 2020.
[40] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. 2009.
[41] G. Schwarz, "Estimating the dimension of a model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.
[42] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 72–81, 2008.
[43] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: a multiplatform benchmark suite for approximate computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, 2017.
[44] C. Sun, C. O. Chen, G. Kurian, L. Wei, J. E. Miller, A. Agarwal, L. Peh, and V. Stojanovic, "DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proceedings of the IEEE/ACM International Symposium on Networks-on-Chip*, pp. 201–210, 2012.

Siyuan Xiao received the bachelor degree in Computer Science and Technology from Guangdong Polytechnic Normal University, Guangzhou, China. He is pursuing his master degree in the School of Software Engineering, South China University of Technology. His research interests include approximate computing and many-core systems.
E-mail: syxiao1337@gmail.com

Xiaohang Wang received the B.Eng. and Ph.D degree in communication and electronic engineering from Zhejiang University, in 2006 and 2011. He is currently an associate professor at South China University of Technology. He was the receipt of PDP 2015 and VLSI-SoC 2014 Best Paper Awards. His research interests include many-core architecture, power efficient architectures, optimal control, and NoC-based systems.

Maurizio Palesi received the MSc and PhD degrees in computer science engineering from the University of Catania, Italy, in 1999 and 2003, respectively. He is an associate professor with Department of Electrical, Electronics and Computer Engineering University of Catania, Catania, Italy. He has published one book, six book chapters, and more than 110 refereed international journals and conference papers. His current research interests include embedded systems design and single-chip implementations of complete embedded systems known as system-on-chip. He was a recipient of the Best Paper Award from DATE 2011. He is an associate editor of eight publications. He has served as a guest editor for several international journals. He is a senior member of the IEEE.

Amit Kumar Singh is a Lecturer (Assistant Professor) at University of Essex, UK. He received the B.Tech. degree from IIT, Dhanbad, India, in 2006, and the Ph.D. degree from Nanyang Technological University (NTU), Singapore, in 2013. His current research interests are design and optimisation of multi-core based computing systems with focus on performance, energy, temperature, reliability and security. He has published over 90 papers in reputed journals/conferences, and received several best paper awards, e.g. IEEE TC February 2018 Featured Paper, ICCES 2017, ISORC 2016, PDP 2015, HiPEAC 2013 and GLSVLSI 2014 runner up. He has served on the TPC of IEEE/ACM conferences like DAC, DATE, CASES and CODES+ISSS.

Liang Wang received the BEng and MSc degree in electronics engineering from Harbin Institute of Technology, China, in 2011 and 2013 respectively, and the Ph.D degree in Computer Science and Engineering from The Chinese University of Hong Kong, Hong Kong, in 2017. He is currently a postdoctoral research fellow at Institute of Microelectronics, Tsinghua University, China. His research interests include power-efficient and reliability-aware design for network-on-chip and many-core system.

Terrence Mak is an Associate Professor at Electronics and Computer Science, University of Southampton. Supported by the Royal Society, he was a Visiting Scientist at Massachusetts Institute of Technology during 2010, and also, affiliated with the Chinese Academy of Sciences as a Visiting Professor since 2013. Previously, He worked with Turing Award holder Prof. Ivan Sutherland, at Sun Lab in California and has awarded Croucher Foundation scholar. His newly proposed approaches, using runtime optimisation and adaptation, strengthened network reliability, reduced power dissipations and significantly improved overall on-chip communication performances. Throughout a spectrum of novel methodologies, including regulating traffic dynamics using network-on-chips, enabling unprecedented MTBF and to provide better on-chip efficiencies, and proposed a novel garbage collections methods, defragmentation, together led to three prestigious best paper awards at DATE 2011, IEEE/ACM VLSI-SoC 2014 and IEEE PDP 2015, respectively. More recently, his newly published journal based on 3D adaptation and deadlock-free routing has awarded the prestigious 2015 IET Computers & Digital Techniques Premium Award. He has published more than 100 papers in both conferences and journals and jointly published 4 books.