ACDC: An Accuracy- and Congestion-aware Dynamic Traffic Control Method for Networks-on-Chip

Siyuan Xiao[1] Xiaohang Wang[2] Maurizio Palesi[3] Amit Kumar Singh[4] Terrence Mak[5]
[1,2]South China University of Technology, [3]University of Catania, [4]University of Essex, [5]University of Southampton
[1]syxiao1337@gmail.com, [2]xiaohangwang@scut.edu.cn, [3]maurizio.palesi@dieei.unict.it,
[4]a.k.singh@essex.ac.uk, [5]t.mak@soton.ac.uk

***Abstract*—Many applications exhibit error forgiving features. For these applications, approximate computing provides the opportunity of accelerating the execution time or reducing power consumption, by mitigating computation effort to get an approximate result. Among the components on a chip, network-on-chip (NoC) contributes a large portion to system power and performance. In this paper, we exploit the opportunity of aggressively reducing network congestion and latency by selectively dropping data. Essentially, the importance of the dropped data is measured based on a quality model. An optimization problem is formulated to minimize the network congestion with constraint of the result quality. A lightweight online algorithm is proposed to solve this problem. Experiments show that on average, our proposed method can reduce the execution time by as much as 12.87% and energy consumption by 12.42% under strict quality requirement, speed up execution by 19.59% and reduce energy consumption by 21.20% under relaxed requirement, compared to a recent work on approximate computing approach for NoCs.**

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

I. Introduction

A golden result is absent for many modern applications [15], thus some trade-offs between accuracy and other metrics (e.g. execution time) can be made. In an emerging powerful compute paradigm, approximate computing [12], the computation/communication effort is reduced by utilizing error-resilience in applications. Approximate computing approaches are found among several layers [1]–[3].

Networks-on-chip (NoCs) have a large impact on system performance and power consumption. Figure 1 shows the opportunity of improving system performance by dropping data in NoC. Legends indicate the proportion of data discarded at NI before being injected into the network. It shows that on average, blindly dropping 60% of data speeds up the execution by 25.77% compared to the case of only dropping 20% of data. However, blindly dropping data causes the application to both waste error tolerance and obtain less performance gain, since the impact on network congestion varies with different packets.

This research program is supported by Natural Science Foundation of Guangdong Province 2018A030313166, and the Science and Technology Research Grant of Guangdong Province No. 2016A010101011 and 2017A050501003, Pearl River S&T Nova Program of Guangzhou No. 201806010038

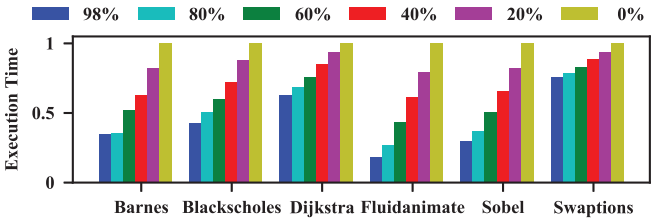

Fig. 1. Execution time at varying the proportion of data dropped for different applications.

Quality control aims at achieving design objectives by utilizing limited error resilience. Previous researches [4], [5] present quality control mechanisms for different purposes. However, quality control in NoCs is quite challenging since it involves NoC features.

Recent studies have also shown their interest in applying approximate computing to NoCs [6]–[10]. [6] allows more data to be compressed using existing pattern-based NoC compression techniques, by compressing similar patterns using the same word. [7] drops data according to runtime buffer utilization of each router. [8] proposes a dual-voltage router architecture, which tolerates bit flips running in low-power mode. In [9], similar data flits can be transferred simultaneously using a auxiliary network. In [10], approximate communication based techniques have been also proposed in the context of wireless NoCs. However, Among these works, a general formulation for quality control in NoCs is not yet presented. Therefore, in this paper, we formulate the performance optimization problem for approximate NoC, along with a lightweight control mechanism to solve it.

The main contributions of this paper are as follows:

1) We model quality loss, network congestion, packet zero load latency and a congestion minimization problem under the approximate NoC context.
2) Based on a flow predicting method, we also propose a control mechanism ACDC to first solve the congestion-minimization problem, then reduces zero load latency.
3) Based on full-system simulation, evaluation shows that ACDC significantly outperforms [7] on both system performance and energy consumption.

978-3-9819263-2-3/DATE19/©2019 EDAA

II. Problem Formulation

A. Notations

We denote n as the number of nodes, m as the number of links, ω as packet size. A flow f_{ij} is defined as communication from the i^{th} node to the j^{th} node. v_{ij}, the volume of f_{ij}, is defined as the communication bandwidth on it. r_{ijk} is a binary value, which is 1 if f_{ij} passes through the k^{th} link, or 0 otherwise. The link capacity c is defined as the maximum bandwidth on each link.

Some optional drop rates $X = \{\pi_1, \ldots, \pi_d\}$ with $\pi_1 < \cdots < \pi_d$ are provided. Using a drop rate $\pi_i \in X$ indicates that the amount of data after data dropping is $(1 - \pi_i)$ of the original. Control knob x_{ij} is defined as the data drop rate for f_{ij}. For an application a, a quality model q_a which estimates quality loss for dropping a certain proportion of data, is used. Besides, θ_a is a user-defined quality requirement, indicating the maximum acceptable quality loss for application a.

The proportion of data that can be dropped is $q_a^{-1}(\theta_a)$, and flow volume in the network can be formulated as $\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} v_{ij}$. Then the total error budget g can be calculated as

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

which is the maximum amount of flow volume that is allowed to be dropped.

B. Quality Model

For each application, a quality model is constructed. First, the error-tolerable data (e.g., pixels of an input image) is figured out in the application source code. The source code is modified to evaluate the quality loss caused by performing approximation on these error-tolerable data.

The modified application is executed for multiple times, varying the proportion of approximated data, to collect their resulting quality loss. These are some discrete samples in the form of (data drop rate, quality loss), and they are transformed into a continuous piecewise function by linear interpolation, which is the required quality model.

C. Congestion Model

The link capacity c, which is the maximum bandwidth 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 extent of congestion.

Thus for the k^{th} link, link congestion can be formulated as

$$c_k^{\text{link}} = \max\left(0, \sum_{i=1}^{n} \sum_{j=1,i\neq j}^{n} \left(r_{ijk}\left(1 - x_{ij}\right) v_{ij}\right) - c\right) \tag{2}$$

where $(1 - x_{ij})\, v_{ij}$ represents the volume of f_{ij} whose data is dropped by x_{ij}.

D. Problem Formulation

Since congestion is critical to network performance, we define an optimization problem to minimize network congestion by selecting an optimal data drop rate x_{ij} for each f_{ij}, while satisfying the quality requirement.

The optimization problem can be formulated as

$$\min \sum_{k=1}^{m} c_k^{\text{link}} \tag{3}$$

$$\text{s.t.} \sum_{i=1}^{n} \sum_{j=1,i\neq j}^{n} x_{ij} v_{ij} \leq g \tag{4}$$

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

Equation 3 is to minimize the accumulated congestion on all m links. Equation 4 guarantees that the amount of dropped data is under the total error budget g, which is the maximum allowed amount of data to be dropped. Calculation of g is stated in Equation 1, which involves the user-defined quality requirement.

E. Serialization Latency

When the network is not congestion-dominant, the most critical factor impacting the network performance is no longer the time spent queueing in a buffer, but the packet zero load latency. Typically, it can be expressed as

$$L_{\text{zero}} = L_h + L_s \tag{5}$$

where L_h is the latency for the first flit to reach the destination, which depends on communication distance (hop counts), and L_s is the serialization latency, e.g., the latency that the destination NI spent receiving all flits since the first arrival. Therefore, this serialization latency can be reduced by decreasing the size of packet.

III. The Proposed Method

A. Overview

We propose a runtime control mechanism for approximate NoC to optimize system performance, which consists of global control and local control. As shown in Figure 2 (a), it includes a global controller at the master node and local controller at each node. Each NI is equipped with a data dropper and a recoverer, to drop data at a given drop rate before injected into the network, and recover them at the destination node, respectively. Data dropper and recoverer used in [7] are adopted for evaluating our proposed control mechanism. It is performed periodically with control interval τ:

1) At the beginning of each control interval, flow prediction is performed in a distributed way that each node sends the flow prediction to the master node, as shown in the left of Figure 2 (b). After receiving flow prediction from all nodes, the global controller triggers global control.
2) In global control, a feasible combination of flow drop rates that minimizes network congestion is first searched out, which preserves a portion of the total error budget. Second, the remaining error budget is allocated to each

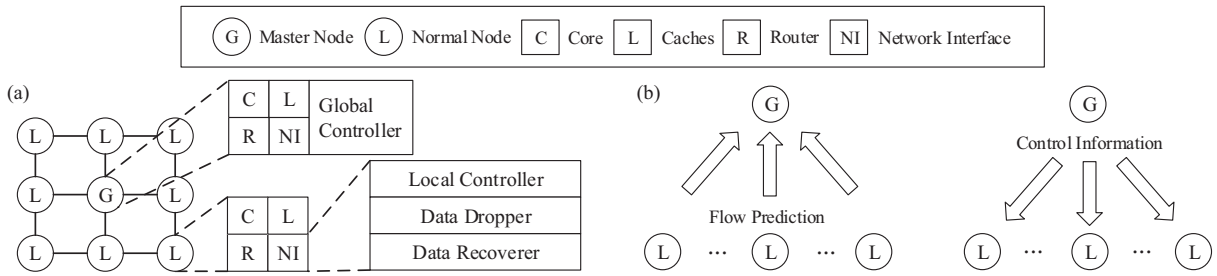

Fig. 2. An overview of ACDC. (a) the components of each node. (b) control data passing in the NoC.

node for reducing serialization latency. The control information delivered to each node, as in the right of Figure 2 (b), includes the corresponding congestion-minimized flow drop rates, the error budget for congestion minimization and the error budget for reducing serialization latency.

3) The local controller of each node adjusts drop rate for each incoming packet.

B. Flow Prediction

In this paper, we propose a flow prediction method based on autoregressive (AR) model and runtime feedback. An AR(p) model can be stated as

$$y^{AR}(t) = \lambda_1 y_{t-1} + \lambda_2 y_{t-2} + \cdots + \lambda_p y_{t-p} + \mu + \epsilon_t \quad (6)$$

where $y^{AR}(t)$ is the prediction of AR model using previous p observed values. y_t represents the observed value at time t, i.e, the volume of a flow. The AR model is trained offline, and the model order p is determined using Bayesian information criterion (BIC). By introducing runtime feedback, it becomes

$$\hat{y}_t = y^{AR}(t) + \Delta_t,\ \Delta_t = \Delta_{t-1} + \alpha\left(y_{t-1} - \hat{y}_{t-1}\right) \quad (7)$$

where α represents learning rate.

C. Global Control

1) Congestion Minimization: The optimization problem in Section II-D has a huge decision space of size $|X|^{n(n-1)}$, ($x_{ij} \in X$ and $n(n-1)$ variables). When the network size scales up, the optimal decision cannot be efficiently found at runtime. Thus in Algorithm 1, we propose a lightweight greedy heuristic.

Congestion of a flow is defined as the accumulation of link congestions along its path. Each flow is associated with a flag e_{ij} indicating whether it is processed or not. 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 flows, and the most congested link along its path is figured out.

Let $c^{\text{link}}_{k^*} = x_{i^*j^*} v_{i^*j^*}$, a drop rate $x_{i^*j^*} = c^{\text{link}}_{k^*} / v_{i^*j^*}$ is calculated for compensating the corresponding congestion on $f_{i^*j^*}$. Adding the limitation of the error budget and the maximum drop rate π_d provided, the drop rate is finally decided as in line 9. Since the flow volume after data dropping is lower, all links along this flow is affected, and so are the flows passing through those links. Their congestion statuses updated except the flows already processed.

Algorithm 1 A greedy heuristic

Input: v_{ij}: data volume of f_{ij}.
Output: x_{ij}: data drop rate for f_{ij}.

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

The loop ends when there is no unprocessed flow, the congestion is eliminated, or the error budget is used up. By analyzing the innermost loop, the worst case time complexity of Algorithm 1 is $O(mn^3)$.

The error budget preserved for congestion minimization is denoted as μ, the portion of μ for the i^{th} node as μ_i, then

$$\mu = \sum_{i=1}^{n} \mu_i,\ \mu_i = \sum_{j=1, i\neq j}^{n} x_{ij} v_{ij} \quad (8)$$

where μ_i is the amount of dropped data on all flows starting from the i^{th} node, when the congestion-minimized flow drop rates are in use.

2) Reducing Serialization Latency: The remaining error budget is denoted as ν ($\nu = g - \mu$), the portion allocated to the i^{th} node as ν_i, then we have

$$\nu_i = \nu \cdot \frac{\rho_i}{\sum_{j=1}^{n} \rho_j},\ \rho_i = \sum_{j=1, i\neq j}^{n} \left(\pi_d - x_{ij}\right) v_{ij} \quad (9)$$

TABLE I
SYSTEM CONFIGURATION

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

where ρ_i measures the opportunity of dropping more data at the i^{th} node , of which ν_i is allocated proportionally.

D. Local Control

At the i^{th} node, after receiving control information from the global controller, the two error budgets (μ_i and ν_i) and the congestion-minimizing drop rates for its $n-1$ flows are updated to the values just received.

For a packet sending to the j^{th} node, first, if μ_i is more than 0, the drop rate is temporarily set to the congestion-minimizing setting x_{ij}, and the amount of dropped data $x_{ij}w$ must be subtracted from μ_i (given the control interval τ, the bandwidth occupied by a packet in this interval is $w = \omega/\tau$). Next, if the error budget ν_i is more than 0, the maximum drop rate π_d is used for injecting this packet, and the increased amount of dropped data is subtracted from ν_i.

IV. EVALUATION

A. Performance Comparison

Our proposed method is evaluated using a cycle-accurate full-system many-core simulator [13]. System configuration is stated in Table I. ABDTR [7] is implemented for comparison. Six applications from PARSEC and AxBench [14] are used for evaluation. Quality metrics and error-tolerable data in [2] [14] are used. Two quality requirement settings are in use, as QR1 being strict and QR2 a relaxed one.

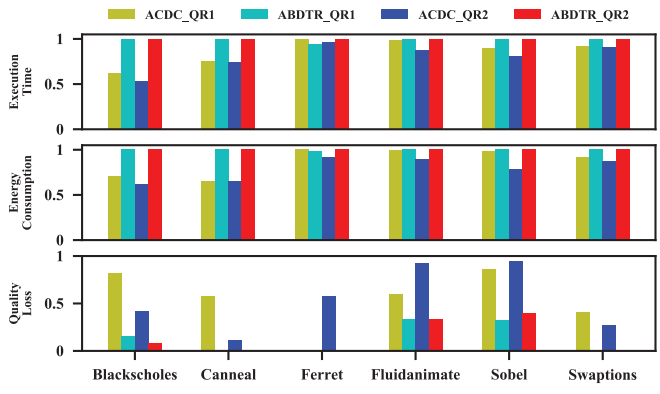

Fig. 3. Execution time, energy consumption and quality loss comparing the two control methods under QR1 and QR2.

Experimental result is shown in Figure 3, the 3 metrics are normalized for each application. Compared to ABDTR, on average, ACDC accelerates execution by 12.87% and reduces energy consumption by 12.42% respectively under QR1, and the counterpart is 19.59% and 21.20% under the more relaxed QR2. Both experiments show that on every applications, ACDC better utilizes error tolerance while not violating the quality requirement.

B. Overhead

The global controller is implemented in software, whereas local controller in hardware. Average execution time of the global control is 419 cycles, much shorter than the control interval (10000 cycles in our experiment). It is trivial since it only occupies processor in the master node and will not stall others.

According to the systhesis result reported by Synopsys Design Compiler and PrimePower targeting a 45 nm TSMC library, a local controller cost 968 μm^2 and 0.636 mW on area and power, respectively. Using DSENT with a 45 nm CMOS technology, a router for configuration in Table I has a total power of 125 mW, and a total area of 67686 μm^2. Therefore, the power and area of ACDC controller are only 0.5% and 1.43% of a router, respectively.

V. CONCLUSION

In this paper, we formulated a performance optimization problem for approximate NoC, along with a lightweight control mechanism based on flow prediction, which consists of global control and local control. Compared with a recent work, the proposed method showed significant improvement on performance, energy and error utilization.

REFERENCES

[1] S. Sidiroglou-Douskos et al. "Managing performance vs. accuracy trade-offs with loop perforation", SIGSOFT FSE 2011: 124-134.
[2] A. Yazdanbakhsh et al. "Mitigating the memory bottleneck with approximate load value prediction", IEEE Design & Test, 2016, 33(1): 32-42.
[3] D. Mohapatra et al. "Design of voltage-scalable meta-functions for approximate computing", DATE'11: 950–955.
[4] D. S. Khudia et al. "Rumba: An online quality management system for approximate computing", ISCA'15: 554-566.
[5] C. Xu et al. "On Quality Trade-off Control for Approximate Computing Using Iterative Training", DAC'17: 52.
[6] R. Boyapati et al. "APPROX-NoC: A Data Approximation Framework for Network-On-Chip Architectures", ISCA'17: 666-677.
[7] L. Wang et al. "ABDTR: Approximation-Based Dynamic Traffic Regulation for Networks-on-Chip Systems", ICCD'17: 153-160.
[8] A. B. Ahmed et al. "AxNoC: Low-power Approximate Network-on-Chips using Critical-Path Isolation", NOCS'18.
[9] V. Y. Raparti et al. "DAPPER: Data Aware Approximate NoC for GPGPU Architectures", NOCS'18.
[10] G. Ascia et al. "Approximate Wireless Networks-on-Chip", Conf. on Design of Circuits and Integrated Systems 2018: 14-16.
[11] C. Bienia et al. "The PARSEC benchmark suite: Characterization and architectural implications", PACT 2008: 72-81.
[12] Q. Xu et al. "Approximate computing: A survey", IEEE Design & Test 33.1 (2016): 8-22.
[13] X. Wang et al. "On self-tuning networks-on-chip for dynamic network-flow dominance adaptation", TECS 13.2s (2014): 73.
[14] A. Yazdanbakhsh et al. "AxBench: A multiplatform benchmark suite for approximate computing", IEEE Design & Test 34.2 (2017): 60-68.
[15] Y. Chen et al. "Convergence of recognition, mining, and synthesis workloads and its implications", Proc. of the IEEE 2008.