**Network-on-Chip (NoC) Performance Analysis and Optimization for Deep Learning Applications**

by

Sumit K. Mandal

A preliminary report for the degree of

Doctor of Philosophy

(Department of Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

06/10/2021

Preliminary Examination Date: 06/10/2021

Preliminary Exam Committee:

Umit Y. Ogras, Associate Professor, Electrical and Computer Engineering, University of Wisconsin-Madison
Mikko H. Lipasti, Professor, Electrical and Computer Engineering, University of Wisconsin-Madison
Karu Sankaralingam, Professor, Computer Sciences Engineering, University of Wisconsin-Madison
Chaitali Chakrabarti, Professor, Electrical Engineering, Arizona State University

CONTENTS

---

## ABSTRACT

Hardware accelerators for deep neural networks (DNNs) exhibit high volume of on-chip communication due to deep and dense connections. State-of-the-art interconnect methodologies for in-memory computing deploy a bus-based network or mesh-based Network-on-Chip (NoC). Our experiments show that up to 90% of the total inference latency of a DNN hardware is spent on on-chip communication when the bus-based network is used. To reduce the communication latency, we propose a methodology to generate an NoC architecture along with a scheduling technique customized for different DNNs. We prove mathematically that the generated NoC architecture and corresponding schedules achieve the minimum possible communication latency for a given DNN. Experimental evaluations on a wide range of DNNs show that the proposed NoC architecture enables 20%-80% reduction in communication latency with respect to state-of-the-art interconnect solutions.

Networks-on-chip (NoCs) have become the standard for interconnect solutions in DNN accelerators as well as industrial designs ranging from client CPUs to many-core chip-multiprocessors. Since NoCs play a vital role in system performance and power consumption, pre-silicon evaluation environments include cycle-accurate NoC simulators. Long simulations increase the execution time of evaluation frameworks, which are already notoriously slow, and prohibit design-space exploration. Existing analytical NoC models, which assume fair arbitration, cannot replace these simulations since industrial NoCs typically employ priority schedulers and multiple priority classes. To address this limitation, we propose a systematic approach to construct priority-aware analytical performance models using micro-architecture specifications and input traffic. Our approach decomposes the given NoC into individual queues with modified service time to enable accurate and scalable latency computations. Specifi-

cally, we introduce novel transformations along with an algorithm that iteratively applies these transformations to decompose the queuing system. Experimental evaluations using real architectures and applications show high accuracy of 97% and up to 2.5× speedup in full-system simulation.

# 1 INTRODUCTION

In recent years, deep neural networks (DNNs) have shown tremendous success in recognition and detection tasks such as image processing, health monitoring, and language processing [59, 77]. Higher accuracy in DNNs is achieved by using larger and more complex models. However, such models require a large number of weights, and consequently, traditional DNN hardware accelerators require a large number of memory accesses to fetch the weights from off-chip memory, leading to a large number of off-chip memory accesses lead to higher latency and energy consumption.

In-Memory Computing (IMC) techniques reduce memory access related latency and energy consumption through the integration of computation with memory accesses. A prime example is the crossbar-based IMC architecture which provides a significant throughput boost for DNN acceleration. At the same time, crossbar-based in-memory computing dramatically increases the volume of on-chip communication, when all weights and activations are stored on-chip. Emerging DNNs with higher accuracy, such as those derived through Neural Architecture Search (NAS) [117, 125, 126], further exacerbate the problem of on-chip communication due to larger model size and more complex connections. Therefore, designing an efficient on-chip communication architecture is crucial for the in-memory acceleration of DNNs.

State-of-the-art IMC architectures usually deploy a bus-based H-Tree interconnect [79, 105]. Figure 2.9 shows that up to 90% of the total inference latency of a DNN hardware is spent on on-chip communication when the H-Tree interconnect is used. In order to reduce on-chip communication latency, NoC-based interconnects are employed for conventional SoCs [44] and DNN accelerators [19, 100]. Eyeriss-V2 [19] proposes to use three different NoCs for weights, activations, and partial sums. Such an architectural choice allows for higher performance at the cost of both

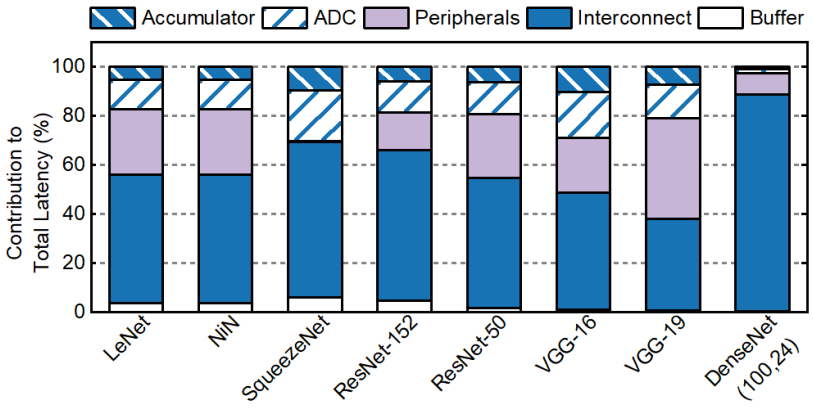



Figure 1.1: Percentage contribution of different components of the IMC hardware to total latency. 40%-90% of the total latency is spent on on-chip communication when bus-based H-Tree interconnect is used.

area and energy consumption. ISAAC [100] employs a concentrated-mesh (cmesh) NoC at the tile-level of the IMC accelerator.

In this work, we minimize the communication energy across a large number of tiles using an NoC architecture with optimized tile-to-router mapping and scheduling. We also propose an optimization technique to determine the optimal number of NoC routers required for each layer of the DNN. Next, we propose a methodology to generate a latency-optimized NoC architecture along with a scheduling technique customized for different DNNs. We prove, through induction, that the proposed NoC architecture achieves minimum possible communication latency using the minimum number of links between the routers. These two techniques together generate a custom NoC for IMC acceleration of a given DNN. We show that the proposed custom IMC architecture achieves 20%-80% improvement in overall communication latency and 5%-25% reduction in end-to-end inference latency with respect to state-of-the-art NoC based IMC architectures [100].

Since the on-chip interconnect is a critical component of DNN acceler-

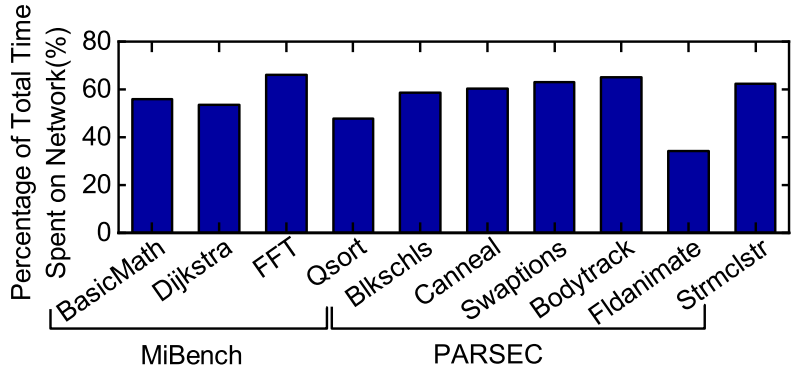



Figure 1.2: Experiments of different applications show that 40%-70% of the total simulation time is spent on the network.

ators as well as multicore architectures, pre-silicon evaluation platforms contain cycle-accurate NoC simulators [2, 45]. NoC simulations take up a significant portion of the total simulation time, which is already limiting the scope of pre-silicon evaluation (e.g., simulating even a few seconds of applications can take days). For example, Figure 2.1 shows that 40%-70% of total simulation time is spent on the network itself when performing full-system simulation using gem5 [10]. Hence, accelerating NoC simulations without sacrificing accuracy can significantly improve both the quality and scope of pre-silicon evaluations.

Several performance analysis approaches have been proposed to enable faster NoC design space exploration [83, 115, 96]. Prior techniques have assumed a round-robin arbitration policy in the routers since the majority of router architectures proposed to date have used round-robin for fairness. In doing so, *they miss two critical aspects* of the industrial priority-based NoCs [44, 98, 47]. First, routers give priority to the flits in the network to achieve predictable latency within the interconnect. flits to the local node in Figure 2.2 are already in the NoC, while flits from class-3 to the neighboring router must wait in the input buffer to be admitted. Consequently, flits in the NoC (class-1 and class-2) experience deterministic service time

at the expense of increased waiting time for new flits. Second, flits from different priority classes can be stored in the same queue. For instance, new read/write requests from the core to tag directories use the same physical and virtual channels as the requests forwarded from the directories to the memory controllers. Moreover, only a fraction of the flits in either the high or low priority queue can compete with the flits in the other queue. For example, suppose the class-2 flits in Figure 2.2 are delivered to the local node. Then, class-3 flits must compete with only class-1 flits in the high-priority queue. Analytical models that ignore this traffic split significantly overestimate the latency, as shown in Section 6.2. In contrast, analytical models that ignore priority would significantly underestimate the latency. Thus, prior approaches that do not model priority [83, 115, 96] and simple performance models for the priority queues [52, 46] are *inapplicable to priority-based industrial NoCs*.

We propose NoC performance analysis technique that *considers traffic*

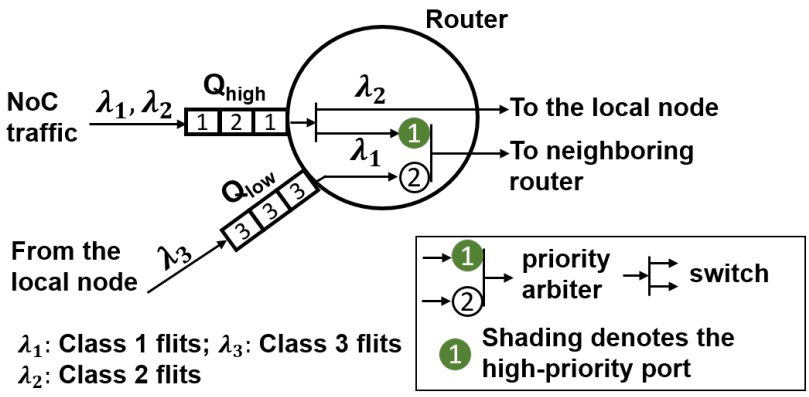



Figure 1.3: The high priority queue ($Q_{high}$) stores two different traffic classes which are already in the NoC, while the low priority queue ($Q_{low}$) stores the newly injected flits from the local node. As flits from class–2 are routed to the local node, low-priority flits compete with only class–1 flits in $Q_{high}$.

*classes with different priorities*. This problem is theoretically challenging due to the non-trivial interactions between classes and shared resources. For example, queues can be shared by flits with different priorities, as shown in Figure 2.2. Similarly, routers may *merge* different classes coming through separate ports, or act as *switches* that can disjoin flits coming from different physical channels. To address these challenges, we propose a two-step approach that consists of an analysis technique followed by an iterative algorithm. The first step establishes that priority-based NoCs can be decomposed into separate queues using traffic splits of two types. Since existing performance analysis techniques cannot model these structures with traffic splits, we develop analytical models for these canonical queuing structures. The second step involves a novel iterative algorithm that composes an end-to-end latency model for the queuing network of a given NoC topology and input traffic pattern. The proposed approach is evaluated thoroughly using both 2D mesh and ring architectures used in industrial NoCs. It achieves 97% accuracy with respect to cycle-accurate simulations for realistic architectures and applications.

The rest of the report is organized as follows. Chapter 2 describes the overview of the preliminary work. The preliminary work consists of performance analysis techniques for priority-aware NoCs under bursty traffic and deflection routing. It also spans a communication-aware hardware accelerator for DNNs. Chapter 3 describes the proposed work of analytical modeling of NoCs with weighted round robin arbitration. Chapter 4 describes the proposed work of communication-aware hardware accelerator for graph convolutional networks (GCNs). Chapter 6 – Chapter 9 provide detailed description of the preliminary work. Finally, Chapter 5 concludes the report.

## 2.1 Performance Analysis of Priority-Aware NoCs

Modern design methodologies in industries involve thorough power, performance, and area evaluations before the architectural decisions are frozen. These pre-silicon evaluations are vital for detecting functional bugs and power-performance violations, since post-silicon fixes are costly, if feasible at all. Therefore, a significant amount of resources are dedicated to pre-silicon evaluation using virtual platforms [65] or full-system simulators [10]. NoC simulations play a critical role in these evaluations, as NoCs have become the standard interconnect solution in many core chip-multiprocessors (CMPs) [44, 49, 47], client CPUs [99], and mobile systems-on-chip [103]. Moreover, there is a growing interest to use NoCs in hardware implementations of deep neural networks [21].

Since the on-chip interconnect is a critical component of multicore architectures, pre-silicon evaluation platforms contain cycle-accurate NoC simulators [2, 45]. NoC simulations take up a significant portion of the total simulation time, which is already limiting the scope of pre-silicon evaluation (e.g., simulating even a few seconds of applications can take days). For example, Figure 2.1 shows that 40%-70% of total simulation time is spent on the network itself when performing full-system simulation using gem5 [10]. Hence, accelerating NoC simulations without sacrificing accuracy can significantly improve both the quality and scope of pre-silicon evaluations.

Several performance analysis approaches have been proposed to enable faster NoC design space exploration [83, 115, 96]. Prior techniques have assumed a round-robin arbitration policy in the routers since the majority of router architectures proposed to date have used round-robin for fairness.

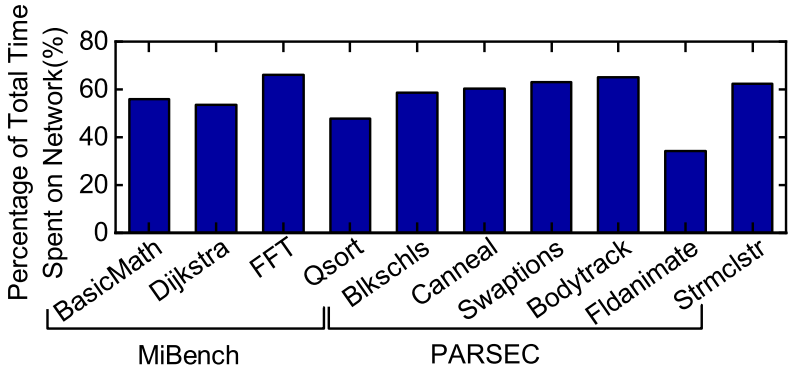



Figure 2.1: Experiments of different applications show that 40%-70% of the total simulation time is spent on the network.

In doing so, *they miss two critical aspects* of the industrial priority-based NoCs [44, 98, 47]. First, routers give priority to the flits in the network to achieve predictable latency within the interconnect. flits to the local node in Figure 2.2 are already in the NoC, while flits from class-3 to the neighboring

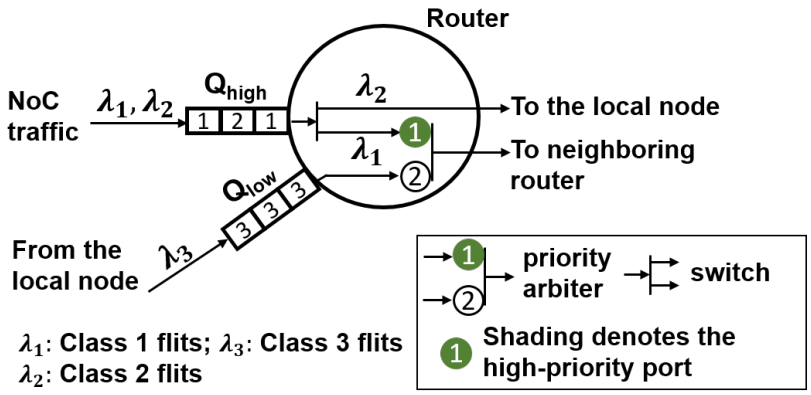



Figure 2.2: The high priority queue ($Q_{high}$) stores two different traffic classes which are already in the NoC, while the low priority queue ($Q_{low}$) stores the newly injected flits from the local node. As flits from class–2 are routed to the local node, low-priority flits compete with only class–1 flits in $Q_{high}$.

router must wait in the input buffer to be admitted. Consequently, flits in the NoC (class-1 and class-2) experience deterministic service time at the expense of increased waiting time for new flits. Second, flits from different priority classes can be stored in the same queue. For instance, new read/write requests from the core to tag directories use the same physical and virtual channels as the requests forwarded from the directories to the memory controllers. Moreover, only a fraction of the flits in either the high or low priority queue can compete with the flits in the other queue. For example, suppose the class-2 flits in Figure 2.2 are delivered to the local node. Then, class-3 flits must compete with only class-1 flits in the high-priority queue. Analytical models that ignore this traffic split significantly overestimate the latency, as shown in Section 6.2. In contrast, analytical models that ignore priority would significantly underestimate the latency. Thus, prior approaches that do not model priority [83, 115, 96] and simple performance models for the priority queues [52, 46] are *inapplicable to priority-based industrial NoCs*.

This chapter presents a novel NoC performance analysis technique that *considers traffic classes with different priorities*. This problem is theoretically challenging due to the non-trivial interactions between classes and shared resources. For example, queues can be shared by flits with different priorities, as shown in Figure 2.2. Similarly, routers may *merge* different classes coming through separate ports, or act as *switches* that can disjoin flits coming from different physical channels. To address these challenges, we propose a two-step approach that consists of an analysis technique followed by an iterative algorithm. The first step establishes that priority-based NoCs can be decomposed into separate queues using traffic splits of two types. Since existing performance analysis techniques cannot model these structures with traffic splits, we develop analytical models for these canonical queuing structures. The second step involves a novel iterative algorithm that composes an end-to-end latency model

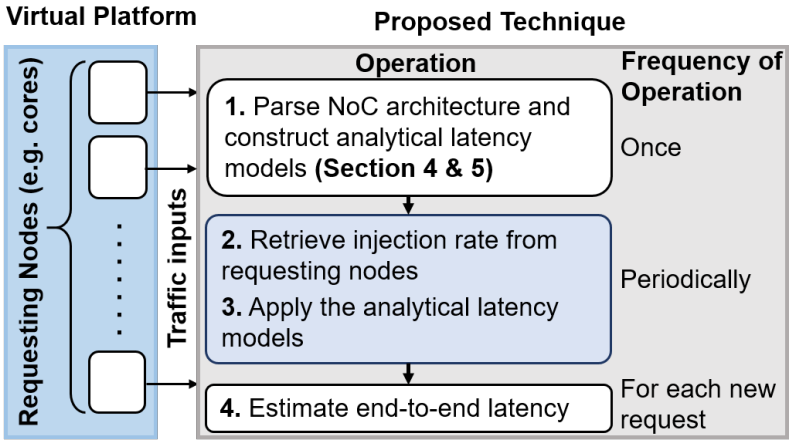



Figure 2.3: Overview of the proposed methodology.

for the queuing network of a given NoC topology and input traffic pattern. The proposed approach is evaluated thoroughly using both 2D mesh and ring architectures used in industrial NoCs. It achieves 97% accuracy with respect to cycle-accurate simulations for realistic architectures and applications.

## Background and Motivation

## Proposed Performance Analysis Flow

The primary target of the proposed model is to accelerate virtual platforms [6] and full-system simulations [10, 89, 71] by replacing time consuming NoC simulations with accurate lightweight analytical models. At the beginning of the simulation, the proposed technique parses the priority-based NoC topology to construct the analytical models, as shown in Figure 2.3. The host, such as a virtual platform, maintains a record of traffic load and the destination address for each node. It also periodically (each 10K-100K cycles) sends the traffic injections of requesting nodes, such as cores, to the proposed technique. Then, the proposed technique

applies the analytical models (steps 2 and 3 in Figure 2.3) to compute the end-to-end latency. Whenever there is a new request from an end node, the host system estimates the latency using the proposed model as a function of the source-destination pair. That is, our model replaces the cycle-by-cycle simulation of flits in NoCs.

## Basic Priority-Based Queuing Models

We assume a discrete time system in which micro-architectural events, such as writing to a buffer, arbitration and switch traversal happen in the integral number of clock cycles. Therefore, we develop queuing models based on arrival process that follows geometric distribution, *in contrast to* continuous time models that are based on Poisson (M for Markovian) arrival assumption. More specifically, we adopt the Geo/G/1 model, in which the inter-arrival time of the incoming flits to the queue follows geometric distribution (denoted by Geo), service time of the queue follows a general discrete-time distribution (denoted by G), and the queue has one server (the '1' in the Geo/G/1 notation). The proposed technique estimates the end-to-end latency for realistic applications accurately, as we

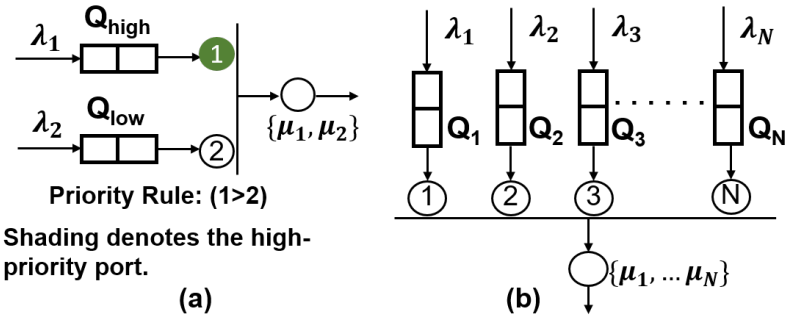



Figure 2.4: (a) A system with two queues. Flits in $Q_{high}$ have higher priority than flits in $Q_{low}$. (b) A system with N queues, where $Q_i$ has higher priority than $Q_j$ for $i < j$

demonstrate in Section 6.4. However, the accuracy is expected to drop if the NoC operates close to its maximum load since the Geometric (similar to Poisson) packet inter-arrival time assumption becomes invalid [83].

Performance analysis techniques in the literature [8, 46, 52] discuss basic priority-based networks in which each priority class has a dedicated queue, as illustrated in Figure 2.4(a). In this architecture, the flits in $Q_{high}$ have higher priority than the flits in $Q_{low}$. That is, flits in $Q_{low}$ will be served *only when* $Q_{high}$ is empty and the server is ready to serve new flits. Another example with N priority classes is shown in Figure 2.4(b). The flits in $Q_i$ have higher priority than flits in $Q_j$ if $i < j$. The average waiting time for each priority class $W_i$ for $1 \leqslant i \leqslant N$ is known for continuous time M/G/1 queues [8]. In the M/G/1 queuing system, flits arrive in the queue following Poisson distribution (M) and the service time of the queue follows general distribution (G). In this work, we first derive waiting time expressions for discrete time Geo/G/1 queues. Then, we employ these models to derive end-to-end NoC latency models.

The average waiting time of flits in a queue can be divided into two parts: (1) waiting time due to the flits already buffered in the queue, and (2) waiting time due to the flits which are in the middle of their service, i.e., the residual time. The following lemma expresses the waiting time as a function of input traffic and NoC parameters.

**Lemma 1:** Consider a queuing network with N priority classes as shown in Figure 2.4(b). Suppose that we are given the injection rates $\lambda_i$, service rates $\mu_i$, residual time $R_i$, and server utilizations $\rho_i$ for $1 \leqslant i \leqslant N$, where $N \geqslant 2$ Then, the waiting time of class-i flits $W_i$ is given as:

$$W_i = \begin{cases} \frac{\sum_{k=1}^{N} R_k}{1-\rho_1}, & \text{for } i = 1 \\ \frac{\sum_{k=1}^{N} R_k + \sum_{k=1}^{i-1} \left(\rho_k + \rho_k W_k\right)}{1-\sum_{k=1}^{i} \rho_k}, & \text{for } i > 1 \end{cases} \tag{2.1}$$

The remaining details of the work is described in Appendix A of the

report and in the reference [73].

## 2.2 Performance Analysis of NoCs with Bursty Traffic

Industrial many-core processors incorporate priority arbitration for the routers in NoC [44]. Moreover, these designs execute bursty traffic since real applications exhibit burstiness [11]. Accurate NoC performance models are required to perform design space exploration and accelerate full-system simulations [52, 96]. Most existing analysis techniques assume fair arbitration in routers, which does not hold for NoCs with priority arbitration used in manycore processors, such as high-end servers [106] and high performance computing (HPC) [44]. A recent technique targets priority-aware NoCs [73], but it assumes that the input traffic follows geometric distribution. While this assumption simplifies analytical models, *it fails to capture the bursty behavior of real applications* [11]. Indeed, our evaluations show that the geometric distribution assumption leads up to 60% error in latency estimation unless the bursty nature of applications is explicitly modeled. Therefore, there is a strong need for NoC performance analysis techniques that consider both priority arbitration and bursty traffic.

This work proposes a novel performance modeling technique for priority-aware NoCs that takes bursty traffic into account. It first models the input traffic as a generalized geometric (GGeo) discrete-time distribution that includes a parameter for burstiness.

We achieve high scalability by employing the principle of maximum entropy (ME) to transform the given queuing network into a near equivalent set of individual queue nodes of multiple-classes with revised characteristics (e.g., modifying service process). Furthermore, our solution involves transformations to handle priority arbitration of the routers across a network of queues. Finally, we construct analytical models of the transformed

queue nodes to obtain end-to-end latency.

The proposed performance analysis technique is evaluated with SYSmark® 2014 SE [5], applications from SPEC CPU® 2006 [38] and SPEC CPU® 2017 [16] benchmark suites, as well as synthetic traffic. The proposed technique has less than 10% modeling error with respect to an industrial cycle-accurate NoC simulator.

The major contributions of this work are as follows:

- Accurate and scalable high-level performance modeling of priority-based NoCs considering burstiness,
- Dynamic approximation of realistic bursty traffic via GGeo distribution,
- Thorough evaluations on industrial priority-based NoCs with synthetic traffic and real applications.

## Background of Generalized Geometric Distribution

The goal of this work is to construct accurate performance models for industrial NoCs under *priority-arbitration* and *bursty traffic*. We mainly target manycore processors used in servers, HPC, and high-end client CPUs [44, 106]. The proposed technique takes burstiness and injection rate of the traffic as input and then provides end-to-end latency of each traffic class.

**Input traffic model assumptions:** Applications usually produce bursty NoC traffic with varying inter-arrival times [11, 96]. We approximate the input traffic using the GGeo discrete-time distribution model, which takes both burstiness and discrete-time feature of NoCs into account [55, 96]. GGeo model includes Geometric and null (no delay) branches, as shown in Figure 2.5. Selection between branches conforms to the Bernoulli trial, where the null (upper) and Geo (lower) branches are selected with probability $p_b$ and $1 - p_b$, respectively. The Geo branch leads to geometrically

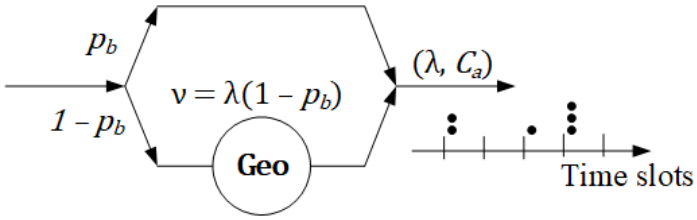



Figure 2.5: GGeo traffic model

distributed inter-arrival time, while the null branch issues additional flit in the current time slot leading to a burst. Both the number of flits in a time slot and the inter-arrival rate depend on $p_b$ [55]. Hence, we use $p_b$ as a parameter of burstiness. GGeo distribution has two important properties [55]. *First,* it is pseudo-memoryless, i.e. the remaining inter-arrival time is geometrically distributed. *Second,* it can be described by its first two moments $(\lambda, C_a)$, where $C_a^2 = 2/(1 - p_b) - \lambda - 1$. We exploit these properties to construct analytical models. The remaining details of the work is described in Appendix B of the report and in the reference [72].

## 2.3 Performance Analysis of NoCs with Deflection Routing

Pre-silicon design-space exploration and system-level simulations constitute a crucial component of the industrial design cycle [87, 31]. They are used to confirm that new generation designs meet power-performance targets before labor- and time-intensive RTL implementation starts [10]. Furthermore, virtual platforms combine power-performance simulators and functional models to enable firmware and software development while hardware design is in progress [65]. These pre-silicon evaluation environments incorporate cycle-accurate NoC simulators due to the criticality of shared communication and memory resources in overall perfor-

mance [2, 45]. However, slow cycle-accurate simulators have become the major bottleneck of pre-silicon evaluation. Similarly, exhaustive design-space exploration is not feasible due to the long simulation times. Therefore, there is a strong need for fast, yet accurate, analytical models to replace cycle-accurate simulations to increase the speed and scope of pre-silicon evaluations [122].

Analytical NoC performance models are used primarily for fast design space exploration since they provide significant speed-up compared to detailed simulators [83, 52, 48, 96]. However, most existing analytical models fail to capture two important aspects of industrial NoCs [44]. *First,* they do not model routers that employ priority arbitration. *Second,* existing analytical models assume that the destination nodes always sink the incoming packets. In reality, network interfaces between the routers and cores have finite (and typically limited) ingress buffers. Hence, packets bounce (i.e., they are deflected) at the destination nodes when the ingress queue is full. Recently proposed performance models target priority-aware NoCs [73, 72]. However, these ignore deflection due to finite buffers and uses the packet injection rate as the primary input. This is a significant limitation since the deflection probability ($p_d$) increases both the hop count and traffic congestion. Indeed, Figure 2.6 shows that the average NoC latency increases significantly with the probability of deflection. For example, the average latency varies from 6–70 cycles for an injection rate of 0.25 packets/cycle/source when $p_d$ varies from 0.1–0.5. Therefore, performance models for priority-aware NoCs have to account for deflection probability at the destinations.

This work proposes an accurate analytical model for *priority-aware NoCs with deflection routing under bursty traffic*. In addition to increasing the hop count, deflection routing also aggravates traffic congestion due to extra packets traveling in the network. Since the deflected packets also have a complex effect on the egress queues of the traffic sources, analytical mod-

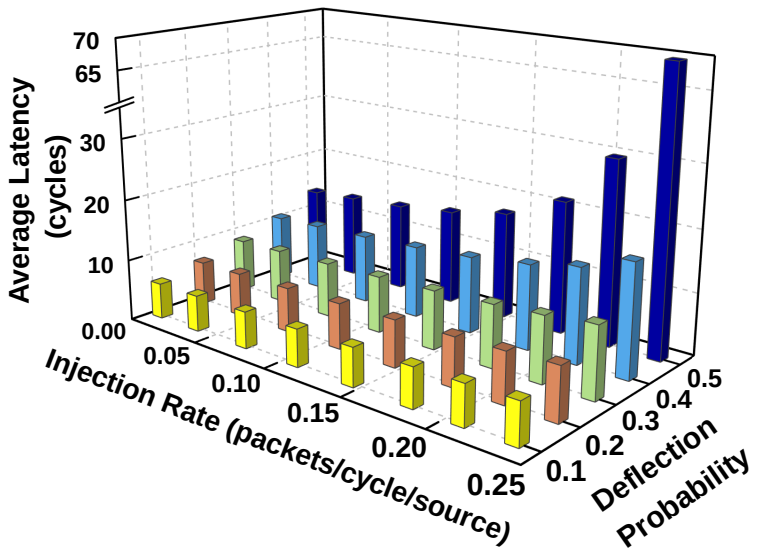



Figure 2.6: Cycle-accurate simulations on a 6×6 NoC show that the average latency increases significantly with larger deflection probability ($p_d$) at the sink.

eling of priority-aware NoCs with deflection routing is challenging. To address this problem, we first need to approximate the probability distribution of inter-arrival time of deflected packets. Specifically, we compute the first two moments of inter-arrival time of deflected packets since we consider bursty traffic. To this end, the proposed approach starts with a canonical queuing system with deflection routing. We first model the distribution of deflected traffic and the average queuing delay for this system. However, this methodology is not scalable when the network has multiple queues with complex interactions between them. Therefore, we also propose a superposition-based technique to obtain the waiting time of the packets in arbitrarily sized industrial NoCs. This technique decomposes the queuing system into multiple subsystems. The structure of these subsystems is similar to the canonical queuing system. After deriving the analytical expressions for the parameters of the distribution model of deflected packets of individual subsystems, we superimpose

the result to solve the original system with multiple queues. Thorough experimental evaluations with industrial NoCs and their cycle-accurate simulation models show that the proposed technique significantly outperforms prior approaches [52, 73]. In particular, the proposed technique achieves less than 8% modeling error when tested with real applications from different benchmark suites. *The major contributions of this chapter are as follows*:

- An accurate performance model for priority-aware NoCs with deflection routing under bursty traffic,
- An algorithm to obtain end-to-end latency using the proposed performance model,
- Detailed experimental evaluation with industrial priority-aware NoC under varying degrees of deflection.

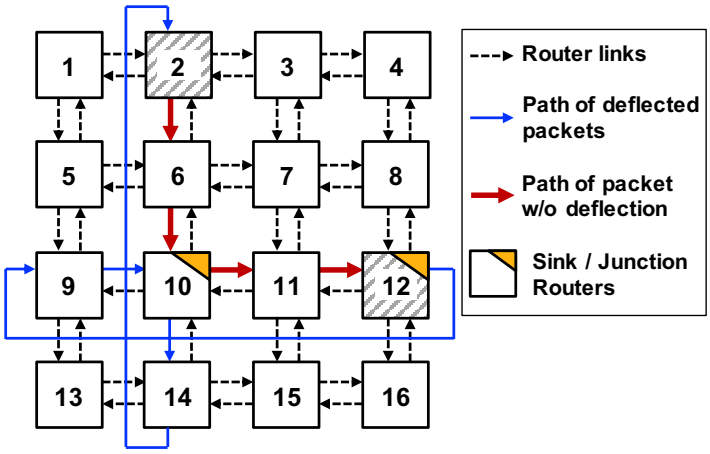



Figure 2.7: A representative 4×4 mesh with deflection routing.

# Background on Deflection Routing

## Assumptions and Notations

**Architecture:** This work considers priority-aware NoCs used in high-end servers and many core architectures [44]. Each column of the NoC architecture, shown in Figure 2.7, is also used in client systems such as Intel i7 processors [99]. Hence, the proposed analysis technique is broadly applicable to a wide range of industrial NoCs.

In priority-aware NoCs, the packets already in the network have higher priority than the packets waiting in the egress queues of the sources. Assume that Node 2 in Figure 2.7 sends a packet to Node 12 following Y-X routing (highlighted by red arrows). Suppose that a packet in the egress queue of Node 6 collides with this packet. The packet from Node 2 to Node 12 will take precedence since the packets already in the NoC have higher priority. Hence, packets experience a queuing delay at the egress queues but have predictable latency until they reach the destination or turning point (Node 10 in Figure 2.7). Then, it competes with the packets already in the corresponding row. That is, the path from the source (Node 2) to the destination (Node 12) can be considered as two segments, which consist of a queuing delay followed by a predictable latency.

Deflection in priority-aware NoCs happens when the ingress queue at the turning point (Node 10) or final destination (Node 12) become full. This can happen if the receiving node, such as a cache controller, cannot process the packets fast enough. The probability of observing a full queue increases with smaller queues (needed to save area) and heavy traffic load from the cores. If the packet is deflected at the destination node, it circulates within the same row, as shown in Figure 2.7. Consequently, a combination of regular and deflected traffic can load the corresponding row and pressure the ingress queue at the turning point (Node 10). This, in turn, can lead to deflection on the column and propagates the congestion

towards the source. Finally, if a packet is deflected more than a specific number of times, it reserves a slot in the ingress queue. This bounds the maximum number of deflections and avoids livelock.

**Traffic:** Industrial priority-aware NoCs can experience bursty traffic, which is characteristic of real applications [11, 96]. This work considers generalized geometric (GGeo) distribution for the input traffic, which takes burstiness into account [55]. GGeo traffic is characterized by an average injection rate ($\lambda$) and the coefficient of variation of inter-arrival time ($C^A$). We define a traffic class as the traffic of each source-destination pair. The average injection rate and coefficient of variation of inter-arrival time of class-$i$ are denoted by $\lambda_i$ and as $C_i^A$ respectively, as shown in Table 8.1. Finally, the mean service time and coefficient of variation of inter-departure time of class-$i$ are denoted as $T_i$ and $C_i^S$.

## Overview of the Proposed Approach

Our goal is to construct an accurate analytical model to compute the end-to-end latency for priority-aware NoCs with deflection routing. The proposed approach can be used to accelerate full system simulations and also to perform design space exploration. We assume that the parameters of the GGeo distribution of the input traffic to the NoC ($\lambda$, $C^A$) are known from the knowledge of the application. The proposed model uses the deflection probability ($p_d$) as the second major input, in contrast to existing techniques that ignore deflection. Its range is found from architecture simulations as a function of the NoC architecture (topology, number of processors, and buffer sizes). Its analytical modeling is left for future work. The proposed analytical model utilizes the distribution of the input traffic to the NoC ($\lambda$, $C^A$) and the deflection probability ($p_d$) to compute the average end-to-end latency as a sum of four components: (1) Queuing delay at the source, (2) the latency from the source router to the junction router, (3) queuing delay at the junction router, and (4) the latency from

the junction router to the destination. Note that all these components account for deflection, and it is challenging to compute them, especially under high traffic load. The remaining details of the work is described in Appendix C of the report and in the reference [74].

## 2.4 Communication-Aware Hardware Accelerators for Deep Neural Networks (DNNs)

In recent years, deep neural networks (DNNs) have shown tremendous success in recognition and detection tasks such as image processing, health monitoring, and language processing [59, 77]. Higher accuracy in DNNs is achieved by using larger and more complex models. However, such models require a large number of weights, and consequently, traditional DNN hardware accelerators require a large number of memory accesses to fetch the weights from off-chip memory, leading to a large number of off-chip memory accesses lead to higher latency and energy consumption. On average, a single off-chip memory access consumes 1,000$\times$ the energy of a single computation [39]. Therefore, there is a strong need to minimize the latency and energy consumption due to the off-chip memory accesses in DNN accelerators.

In-Memory Computing (IMC) techniques reduce memory access related latency and energy consumption through the integration of computation with memory accesses. A prime example is the crossbar-based IMC architecture which provides a significant throughput boost for DNN acceleration. Prior works have shown that both SRAM- and ReRAM-based crossbar architectures effectively improve performance and energy efficiency [100, 120, 105, 108]. Such advantages stem from the efficiency of matrix multiplication implementation in crossbar architectures. At the

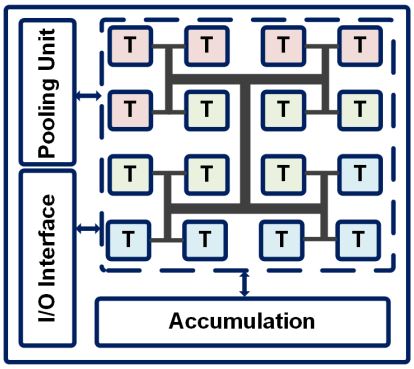



Figure 2.8: Multi-tiled IMC architecture with bus-based H-Tree interconnect [18].

same time, crossbar-based in-memory computing dramatically increases the volume of on-chip communication, when all weights and activations are stored on-chip. Emerging DNNs with higher accuracy, such as those derived through Neural Architecture Search (NAS) [117, 125, 126], further exacerbate the problem of on-chip communication due to larger model size and more complex connections. Therefore, designing an efficient on-chip communication architecture is crucial for the in-memory acceleration of DNNs.

State-of-the-art IMC architectures usually deploy a bus-based H-Tree interconnect [79, 105]. Figure 2.8 shows such a multi-tiled IMC architecture with bus-based H-Tree interconnect [18]. In this figure, the tiles in different layers shown in different colors, are connected through a bus-based H-Tree interconnect. We evaluate a range of DNNs for such an architecture using the NeuroSim [18] benchmarking tool. Figure 2.9 shows that up to 90% of the total inference latency of a DNN hardware is spent on on-chip communication when the H-Tree interconnect is used.

In order to reduce on-chip communication latency, NoC-based interconnects are employed for conventional SoCs [44, 75] and DNN accelerators [19, 100]. Eyeriss-V2 [19] proposes to use three different NoCs for weights, activations, and partial sums. Such an architectural choice allows

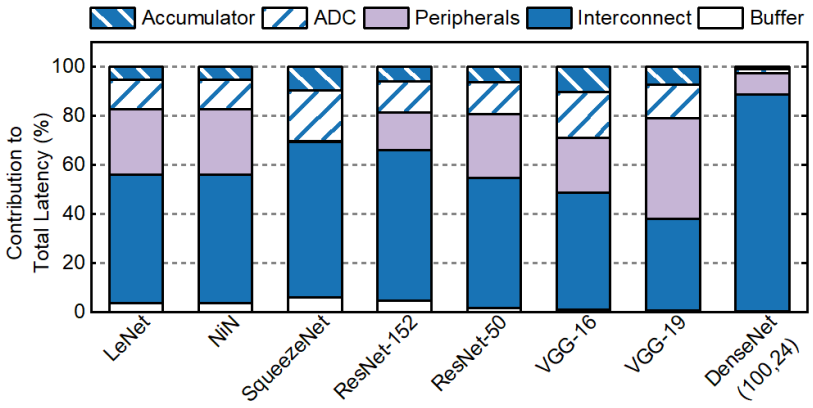



Figure 2.9: Percentage contribution of different components of the IMC hardware to total latency. 40%-90% of the total latency is spent on on-chip communication when bus-based H-Tree interconnect is used.

for higher performance at the cost of both area and energy consumption. ISAAC [100] employs a concentrated-mesh (cmesh) NoC at the tile-level of the IMC accelerator. These empirical approaches demonstrate the necessity of NoC for in-memory computing of DNNs. However, a regular NoC does not guarantee minimum possible communication latency for DNN architectures. The performance of the NoC depends on both the NoC structure and the underlying workload.

In this chapter, we minimize the communication energy across a large number of tiles using an NoC architecture with optimized tile-to-router mapping and scheduling. We also propose an optimization technique to determine the optimal number of NoC routers required for each layer of the DNN. Next, we propose a methodology to generate a latency-optimized NoC architecture along with a scheduling technique customized for different DNNs. We prove, through induction, that the proposed NoC architecture achieves minimum possible communication latency using the minimum number of links between the routers. These two techniques together generate a custom NoC for IMC acceleration of a given DNN.

We show that the proposed custom IMC architecture achieves 20%-80% improvement in overall communication latency and 5%-25% reduction in end-to-end inference latency with respect to state-of-the-art NoC based IMC architectures [100].

Constructing a custom NoC for each DNN is not a practical choice as fabricating custom hardware is expensive. The optimum DNN configuration changes due to on-line adaptation of algorithmic pruning ratio [123] and accuracy vs speed/power trade-off [119]. Consequently, communication patterns between different layers also change with the DNN configuration. Hence, the NoC needs to be configured to maintain optimality. Moreover, new DNNs are being designed at a fast rate due to the large research volume in this domain. Therefore, an exhaustive design-time exploration that considers all possible DNNs is not feasible. As a result, the NoC designed considering only the known DNNs will not be optimal for new DNNs. Hence, it must be configured at run-time to maintain the optimality. To this end, we propose a reconfigurable solution for two categories of DNNs namely, edge We categorize the DNNs based on its application. For example, authors in [17] show the extensive usage of LeNet, SqueezeNet, VGG in various edge devices, and [30] use ResNet-152 DNN for cloud-based applications such as video analytics. However, there exist multiple factors which differentiate these DNNs, namely, number of layers, number of parameters, connection density, etc. In this paper we consider LeNet, NiN, SqueezeNet, VGG-16, and VGG-19 as edge-computing based DNNs and ResNet-50, ResNet-152, and DenseNet (100,24) as cloud computing-based DNNs. We construct separate NoC architectures for these two categories offline. When a new DNN that was not known at design-time is to be executed, the NoC architecture is reconfigured to accommodate the DNN. Through leave-one-out experiments, we show that the proposed reconfigurable NoC has less than 5% performance degradation on average with respect to the customized solution. Overall, the

proposed methodology generates both custom and reconfigurable NoC-based IMC hardware architectures that provide better performance than state-of-the-art IMC hardware for DNNs.

The major contributions in this chapter are as follows:

- A methodology to construct an NoC architecture along with a scheduling technique that provides minimum communication latency for a given DNN. We prove by induction that the proposed NoC achieves minimum communication latency.
- Reconfigurable NoC architecture for edge computing-based and cloud computing-based DNNs.
- Experimental evaluation of the proposed NoC-based IMC architecture showing up to 80% reduction in communication latency with respect to state-of-the-art interconnect solution for IMC hardware of DNNs.

## Background and Overview

## Background of In-Memory Computing

Conventional architectures separate the data access from memory and the computation in the computing unit. In contrast, IMC seamlessly integrates computation and memory access in a single unit such as the crossbar [100, 105, 108, 120]. It has emerged as a promising method to address the memory access bottleneck. Both SRAM and NVM-based (e.g. ReRAM) IMC hardware architectures provide a dense and parallel structure to achieve high performance and energy efficiency. This localizes computation and data memory in a more compact design and enhances parallelism with multiple-row access, resulting in improved performance [100, 105].

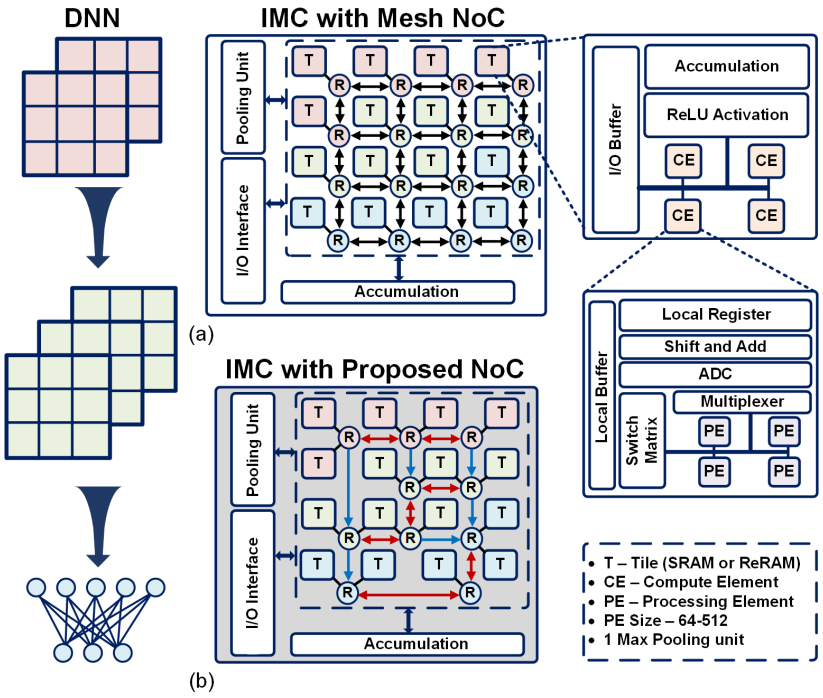



Figure 2.10: IMC architecture with an arbitrary DNN mapped onto different tiles with, (a) the mesh-NoC and (b) the proposed latency-optimized NoC.

The IMC architecture consists of Processing Elements (PE) or crossbar arrays built with IMC cells which hold the weights of the DNN. The size of the PE can vary from 64×64 to 512×512. Along with the computing unit, peripheral circuits such as ADC, Sample and Hold circuit (S&H) and accumulator circuits are used to obtain each DNN layer's computation result. The Word-Line (WL) is activated by the input activation which allows for the MAC operation to be performed along the Bit-Line (BL) via analog voltage/current accumulation. The analog MAC result is converted to digital values using an ADC, and subsequently accumulated using a shifter and adder circuit.

## Overview of the Proposed IMC Architecture with Latency-optimized NoC

Figure 2.10 shows a representative IMC hardware architecture for DNN inference acceleration [18]. The proposed IMC system utilizes a hierarchical architecture where each tile has computing elements (CEs) and each CE employs processing elements (PEs). We assume all the weights and activations are stored on-chip. Each tile consists of four CEs, input-output (IO) buffers, accumulator circuits, and a ReLU activation circuit. An H-Tree-based interconnect is used to connect the different hardware units in the tile. In addition, there is a global pooling and accumulator unit to perform pooling and inter-tile accumulation, respectively.

Each CE in a tile consists of four PEs that communicate through a bus-based interconnect. The PEs represent the crossbar structure (SRAM or ReRAM) which performs the computation. Each CE further consists of a read-out circuit that converts the MAC results from analog to the digital domain. The read-out circuit consists of a sample and hold circuit, flash ADCs, and a shift and add circuit. The choice of ADC stems from the precision of the partial sums required for the best accuracy and the physical footprint and performance of the ADC. A multiplexer circuit is employed to share the read-out circuit among different columns of the IMC crossbar. We multiplex 8 columns for each read-out circuit in a time-multiplexed manner to reduce both area and energy for the peripheral circuitry. Finally, the architecture does not utilize a DAC; it assumes sequential input signaling, i.e., an $n$-bit input signal is fed over $n$ clock cycles in a bit-serial manner.

Figure 2.10(a) shows the IMC architecture with a regular mesh NoC at the tile level. The regular mesh NoC has one router-per-tile and employs the standard X-Y routing algorithm. Figure 2.10(b) shows the IMC architecture with the proposed latency-optimized NoC. The proposed latency-optimized NoC architecture utilizes the optimal number of routers and links to perform on-chip communication. Such an architecture re-

sults in reduced energy and latency. The remaining details of the work is described in Appendix D of the report and in the reference [57, 76].

# 3 PROPOSED WORK-1: MULTI-OBJECTIVE OPTIMIZATION TO DESIGN LATENCY-OPTIMIZED NOC

Networks-on-chip continues playing a central role as many-core processors with 40 or more cores start dominating the server market [3, 26]. As the commercial solutions scale up, the latency, area, and power consumption overheads of NoCs become increasingly crucial. The designers need analytical power-performance models to guide complex design decisions during the architecture development and implementation phases. After that, the same type of models are required by virtual platforms, commonly used to develop and evaluate the software ecosystem, and applications [20]. Hence, there is a strong demand for high-fidelity analytical techniques that accurately model fundamental aspects of industrial designs across all segments ranging from systems-on-chip to client and server systems.

NoCs can be broadly classified in terms of buffer usage as buffered and bufferless architectures [80, 28, 82]. The majority of early solutions adapted buffered techniques, such as wormhole and virtual-channel switching, where the packets (or their flits) are stored in intermediate routers. Area, latency, and energy consumption of buffers have later led to bufferless architectures, where the intermediate routers simply forward the incoming flits if they can and deflect otherwise.

Bufferless NoCs successfully save significant buffer area and enable ultra-fast, as low as single-cycle routing decisions [80, 28]. Therefore, many industrial NoCs used in server and client architectures employ bufferless solutions to minimize the communication latency between the cores, last-level caches (LLC), and main memory [104, 26]. These solutions give priority to the packets already in the network to enable predictable and fast communication while stalling the newly generated packets from the processing and storage nodes. However, buffer area savings and low communication latency come at the cost of the early onset of congestion.

Indeed, the packets wait longer at the end nodes, and the throughput saturates faster when the NoC load increases. Moreover, all routers in the NoCs remain powered on, increasing the NoC power consumption. Therefore, there is a strong need to address these shortcomings.

Buffered NoCs with virtual channel routers have been used more commonly in academic work and most recent industry standards [90]. Shared buffering resources, such as input and output channels, require arbitrating among different requesters. For example, suppose that packets in different input channels request to be routed to the same output channel. An arbiter unit needs to resolve the conflicts and grant access to the requester to avoid packet loss. The architectures proposed to date predominantly employ a basic round-robin (RR) arbiter to provide fairness to all requesters [101, 63, 116]. Although the decisions are locally fair, the number of arbitrations a packet goes through grows with its path length. Hence, RR arbitration is globally not fair. More importantly, basic RR cannot provide preference to a particular input, which is typically desired since not all requests are equal. For example, data and acknowledgment packets can have higher priority than new requests to complete outstanding transactions, especially when the network is congested.

WRR arbitration provides flexibility in allocating bandwidth proportionally to the importance of the traffic classes, unlike basis round-robin and priority-based arbitration. Each requester has an assigned weight, which is a measure of its importance. A larger weight indicates that the requester is given more preference in arbitration. Due to its generality, WRR arbitration has been employed in several NoC proposals in the literature [94, 37, 124]. Indeed, NoCs with WRR arbitration provide better throughput compared to RR arbitration [37]. *Despite its potential, the WRR arbitration technique has not been analyzed theoretically, especially in the context of large-scale NoCs.* A large body of literature has proposed performance analysis techniques for buffered and bufferless NoCs since analytical mod-

els play a crucial role in fast design space exploration and pre-silicon evaluation [112, 96, 51]. In contrast, no analytical modeling technique has been proposed to date for NoCs with WRR arbitration. A formal analysis is required to understand the behavior of NoCs with WRR arbitration. At the same time, executable performance models are needed to guide many-core processor design and enable virtual platforms for pre-silicon evaluation.

WRR arbitration is promising for NoCs since it can tailor the communication bandwidth to different traffic classes. Furthermore, it can provide end-to-end latency-fairness to different source-destination pairs, unlike basic round-robin and priority arbitration techniques. However, these capabilities come at the expense of a vast design parameter space. An $n \times m$ mesh with $P$-port routers has $n \times m \times P$ tunable weights, e.g., an $8 \times 8$ 2D mesh would have 320 WRR weights. Due to this large design space, the current practice is limited to assigning two weights to each router (e.g., one weight to local ports and another one to packets already in the NoC).

The benefits of the proposed theoretical analysis are two-fold. *First*, it can enable accurate pre-silicon evaluations and design space exploration without time-consuming cycle-accurate simulations. *Second*, it can be used to find the combination of weights that optimizes the performance, i.e., to solve the optimization problem described in the previous paragraph.

# 4 PROPOSED WORK-2: HARDWARE ACCELERATOR FOR GRAPH CONVOLUTIONAL NETWORKS (GCNS)

Graph convolutional networks (GCNs) have shown tremendous success for various applications, including node classification, social recommendations, and link predictions [27, 121, 23]. Their powerful learning capabilities on graphs have attracted attention to additional research areas like image processing and job scheduling [111, 78]. Consequently, leading technology companies like Google and Facebook have developed libraries and systems for GCNs [64, 1], stimulating further research.

GCNs operate on graphs by preserving each node's connectivity information. They have irregular data patterns since the relation between the nodes, i.e., the edge connections, do not necessarily follow a specific pattern. In strong contrast, classical convolutional neural networks (CNNs) are optimized for regular data, which prevents them from capturing the connectivity information in the graph. GCNs use a neighbor aggregation scheme that computes each node's features using a recursive aggregation and transformation. The aggregation process depends on the graph structure, while the transformation process uses a technique similar to CNN computations. These processes repeat until embeddings for each node are generated at the end. As the data is sparse, irregular, and high dimensional, general-purpose platforms like CPU and GPU require energy-intensive memory accesses even if they use complex caching and prefetching techniques [22]. Hence, the state-of-the-art GCN models are large and complex [33, 35, 53]. Multiple software-based techniques have been proposed to reduce the computations by utilizing the sparsity of the graph [107, 68]. However, GCN execution still suffers from high latency and energy consumption.

The prevalence and computational complexity of GCNs call for high-

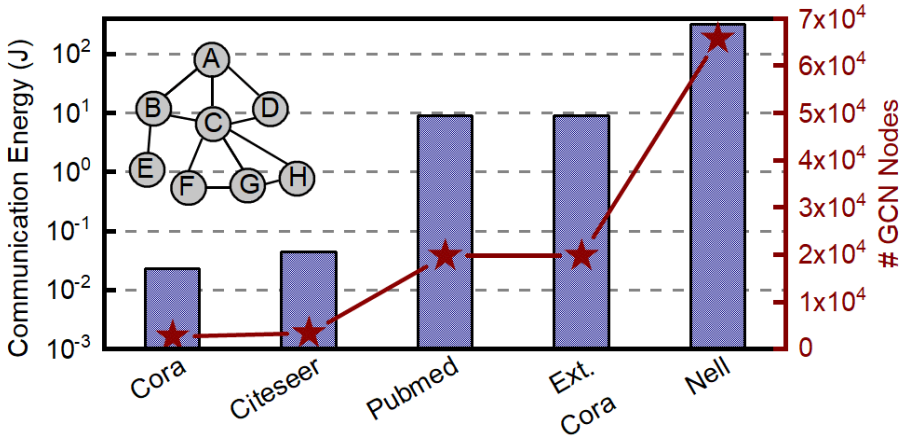



Figure 4.1: Communication energy with a baseline IMC-based GCN accelerator. In the baseline architecture, the number of compute elements is equal to the number of GCN nodes and compute elements are interconnected by a 2D mesh NoC through a dedicated router. The x-axis is sorted by increasing number of GCN nodes.

performance and energy-efficient hardware accelerators. In contrast to software implementations, hardware accelerators perform GCN computations with significantly lower latency and higher energy efficiency. Due to this potential, a couple of recent studies proposed GCN accelerators [118, 66]. These techniques implement systolic array-based architectures to perform the computations. Since this approach requires a large number of weights, the resulting GCN hardware accelerators need to make many memory accesses to fetch the weights from off-chip memory. In turn, frequent off-chip memory accesses lead to higher latency and energy consumption as single off-chip memory access consumes on average 1,000$\times$ more energy than computation [39]. Therefore, there is an urgent need to minimize the latency and energy consumption due to the off-chip memory accesses in GCN accelerators.

In-memory computing (IMC) decreases memory access-related latency and energy consumption by integrating computation with memory

accesses embedding [105]. A notable example is the crossbar-based IMC architecture, which provides a significant throughput boost for hardware acceleration by storing the weights on the chip. However, crossbar-based in-memory computing dramatically increases the volume of on-chip communication when all weights and activations are stored on-chip. In turn, the communication energy also increases exorbitantly. To quantify this effect, we implemented an IMC-based GCN accelerator baseline for popular benchmarks. Each node in the GCN is implemented using a compute element (CE) (array of IMC crossbars) that performs the required operations. The CEs that make up the design are interconnected by a 2D mesh network-on-chip (NoC) through dedicated routers. Figure 4.1 shows that the communication energy increases with the number of GCN nodes. Furthermore, with larger GCNs, the required number of compute elements as well the number of routers become very high, resulting in an increased area of the chip. *Therefore, designing an efficient on-chip communication architecture is crucial for the in-memory acceleration of GCNs.*

We propose a communication-aware in-memory computing architecture for GCN hardware acceleration. The architecture distributes the GCN computations into multiple compute elements, referred to as CEs. Each CE utilizes RRAM-based crossbars to perform computation while significantly reducing frequent off-chip memory accesses. Furthermore, it considers the intra- and inter-CE communications to design an optimized on-chip interconnection network. Specifically, we construct an objective function that represents the energy-delay product of communication. We show that the objective function is convex. Then, we minimize the objective function to obtain the number of CEs.

# 5 CONCLUSION OF THE REPORT

In this report, we propose an approach to build analytical models for priority-based NoCs with multi-class flits under bursty traffic and deflection routing. As we emphasized, no prior work has presented analytical models that consider priority arbitration and multi-class flits in a single queue simultaneously. Such a priority-based queuing network is decomposed into independent queues using novel transformations proposed in this work. We evaluate the efficiency of the proposed approach by computing end-to-end latency of flits in a realistic industrial platform and using real application benchmarks. Our extensive evaluations show that the proposed technique achieves a high accuracy of 97% accuracy compared to cycle-accurate simulations for different network sizes and traffic flows.

We also present a latency-optimized reconfigurable NoC for in-memory acceleration of DNNs. State-of-the-art interconnect methodologies include bus-based H-Tree interconnect and mesh-NoC. We show that bus-based H-Tree interconnect contributes significantly to the total inference latency of DNN hardware and are not a viable option. Mesh-NoC based IMC architectures are better than bus-based H-tree but they too do not consider the non-uniform weight distribution of different DNNs, DNN graph structure, and the computation-to-communication imbalance of the DNNs.None of the architectures holistically investigated minimization of communication latency. In contrast, our proposed latency-optimized NoC guarantees minimum possible communication latency between two consecutive layers of a given DNN. Experimental evaluations on a wide range of DNNs confirm that the proposed NoC architecture enables 60%-80% reduction in communication latency with respect to state-of-the-art interconnect solutions.

Finally, we plan to construct analytical modeling technique for NoCs with weighted round robin (WRR) arbitration, since WRR provides more

fairness than round robin arbitration or priority arbitration. We also plan to develop communication-aware hardware accelerator for graph convolutional network (GCN) in future.

# 6 APPENDIX A: PERFORMANCE ANALYSIS OF PRIORITY-AWARE NOCS

## 6.1 Related Work

Performance analysis techniques are useful for exploring design space [88] and speeding up simulations [83, 52, 115]. Indeed, there is continuous interest in applying novel techniques such as machine learning [96] and network calculus [95] to NoC performance analysis. However, these studies do not consider multiple traffic classes with different priorities. Since state-of-the-art industrial NoC designs [26, 44] use *priority-based arbitration* with multi-class traffic, it is important to develop performance analysis for this type of architectures.

Kashif et al. have recently presented priority-aware router architectures [48]. However, this work presents analytical models only for worst-case latency. In practice, analyzing the average latency is important since using worst-case latency estimation in full-system would lead to inaccurate conclusions. A recent technique proposed an analytical latency model for priority-based NoC [52]. This technique, however, assumes that each queue in the network contains a single class of flits.

Several techniques present performance analysis of priority-based queuing networks outside the NoC domain [8, 13, 42]. Nevertheless, these techniques do not consider multiple traffic classes in the same queue. The work presented in [4] considers multiple traffic classes, but it assumes that high priority packets preempt the lower priority packets. However, this is not a valid assumption in the NoC context. A technique that can handle two traffic classes, Empty Buffer Approximation (EBA), has been proposed in [7] for a priority-based queuing system. This approach was later extended to multi-class systems [46]. However, EBA ignores the residual time caused by low priority flits on high priority traffic. Hence, it

is impractical to use EBA for priority-aware industrial NoCs.

The aforementioned prior studies assume a continuous-time queuing network model, while the events in synchronous NoCs take place in discrete clock cycles. A discrete-time priority-based queuing system is analyzed in [110]. This technique forms a Markov chain for a given queuing system, then analyzes this model in z-domain through probability generating functions (PGF). PGFs deal with joint probability distributions where the number of random variables is equal to the number of traffic classes in the queuing system. This approach is not scalable for systems with large number of traffic classes because the corresponding analysis becomes intractable. For example, an industrial $8\times8$ NoC would have 64 sources and 64 destinations which will result in 4096 ($64\times64$) variables with PGF. Furthermore, our approach outperforms this technique, as demonstrated in Section 6.4.

In contrast to prior approaches, we propose a scalable and accurate closed form solution for a priority-based queuing network with multi-class traffic. The proposed technique constructs end-to-end latency models using two canonical structures identified for priority-based NoCs. Unlike prior approaches, our technique scales to any number of traffic classes. To the best of our knowledge, this is the first analytical model for priority-based NoCs that considers both (1) shared queues among multiple priority classes and (2) traffic arbitration dependencies across the queues.

## 6.2 Proposed Network Transformations

This section describes two canonical queuing structures observed in priority-based NoCs. We first describe these structures and explain why prior analysis techniques fail to analyze them. Then, we present two novel transformations and accurate analysis techniques.

## Transformation 1: Split at *High* Priority Queue

**Conceptual Illustration:** Consider the structure shown in Figure 6.1(a). As illustrated in Section **??**, flits from traffic class-1 and 2 are already in the network, while flits from traffic class-3 are waiting in $Q_{low}$ to be admitted. Since routers give priority to the flits in the network in industrial NoCs, class-1 flits have higher priority than those in $Q_{low}$. To facilitate the description of the proposed models, we represent this system by the structure shown in Figure 6.1(b). In this figure, $\mu_i$ represents the service rate of class-i for $i = 1, 2, 3$. If we use Equation 2.1 to obtain an analytical model for the waiting time of traffic class-3, the resulting waiting time will be highly pessimistic, as shown in Figure 6.2. The basic priority-based queuing model overestimates the latency, since it assumes each class in the network occupy separate queues. Hence, all flits in $Q_1$ have higher priority than those in $Q_2$.

**Proposed Transformation:** The basic priority equations cannot be applied to this system since flit distribution of class-1 as seen by class-3 flits will change depending on the presence of class-2 traffic. To address this challenge, we propose a novel structural transformation, Figure 6.1(b) to Figure 6.1(c). Comparison of the structures before and after the transformation reveals:

- The top portion ($Q_1$ with its server) is identical to the original structure,

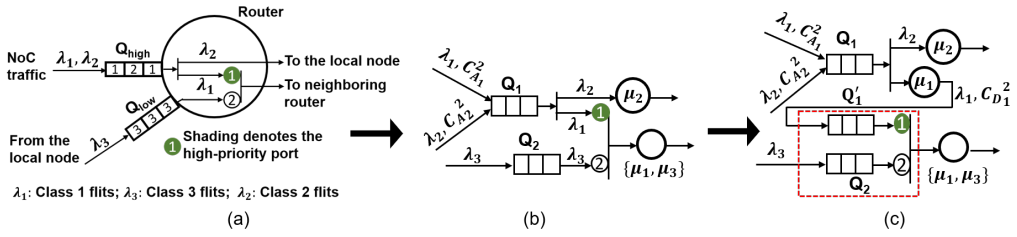



Figure 6.1: Split at high priority: Structural Transformation.

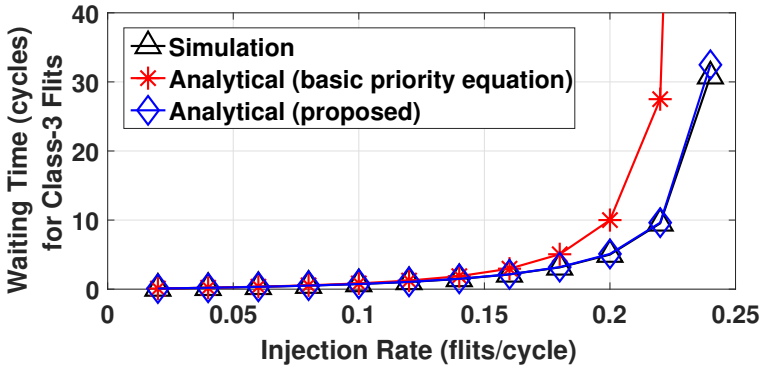



Figure 6.2: Comparison of simulation with the basic priority-based queuing model and proposed analytical model.

since $\mu_1$ and $\mu_2$ remain the same due to higher priority of class-1 over class-3.

- The bottom portion ($Q_1$ and $Q_2$) forms a basic priority queue structure, as highlighted by the red dotted box.

The basic priority queue structure is useful since we have already derived its waiting time model in Equation 2.1. However, the arrival process at $Q_1$ must be derived to apply this equation and ensure the equivalence of the structures before and after the transformation.

We derive the second order moment of inter-departure time of class-1 using the decomposition technique presented in [13]. These inter-departure distributions are functions of inter-arrival distributions of all traffic classes flowing in the same queue and service rate of the classes, as illustrated in Figure 6.3. This technique first calculates the effective coefficient of variation at the input ($C_A^2$) as the weighted sum of the coefficient of variation of individual classes ($C_{A_i}^2$ in Figure 6.3-Phase 1). Then, it finds the effective coefficient of variation for the inter-departure time ($C_D^2$) using $C_A^2$ and the coefficient of variation for the service time ($C_B^2$). In the final phase, the coefficient of variation for inter-departure time of individual

classes is found, as illustrated in Figure 6.3(Phase 3). By calculating the first two moments of the inter-arrival statistics of $Q_1$ as $\lambda_1$ and $C^2_{D_1}$, we ensure that the transformed structure in Figure 6.1(c) approximates the original system. This decomposition enables us to find the residual time for class-1 $R_1^{Q_1}$ as:

$$R_1^{Q_1} = \frac{1}{2}\frac{\rho_1}{\mu_1}\left(\frac{C^2_{D_1} + C^2_B}{2}\right) - \frac{\rho_1\mu_1}{2} \tag{6.1}$$

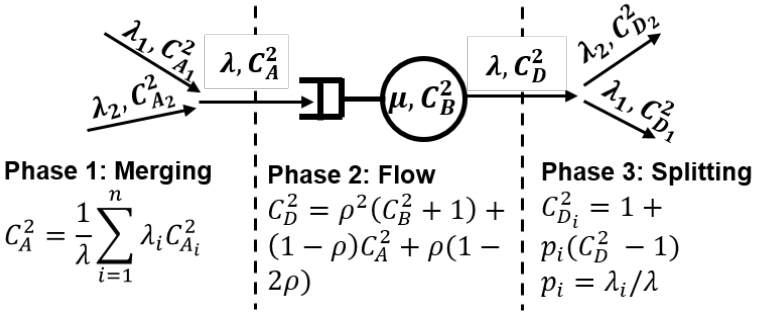



Figure 6.3: Decomposition technique: In phase 1, different traffic flows merge into a single flow with an inter-arrival time $C_A$; in phase 2, flits flow into the queue and leave the queue with an inter-departure time $C_D$; in phase 3, flits split into different flows with individual inter-departure time.

**Proposed Analytical Model:** The bottom part of the transformed system in Figure 6.1(c) is the basic priority queue (marked with the dotted red box). Therefore, the higher priority part of Equation 2.1 can be used to express the waiting time of class-1 flits as:

$$W_1^{Q_1} = \frac{R_1^{Q_1} + R_3}{1 - \rho_1} \tag{6.2}$$

where the residual time of class-1 flits $R_1^{Q_1}$ is found using Equation 6.1. Subsequently, this result is substituted in the lower priority portion of

Equation 2.1 to find the waiting time for class-3 flits:

$$W_3 = \frac{R_1^{Q_1} + R_3 + \rho_1 + \rho_1 W_1^{Q_1}}{1 - \rho_1 - \rho_3} \tag{6.3}$$

We also note the waiting time of class-2 flits, $W_2$, is not affected by this transformation. Hence, we can express it as $W_2 = \frac{R_2}{1-\rho_2}$, using Equation 2.1 for the degenerate case of $N = 1$.

Figure 6.2 shows that the waiting time calculated by the proposed analytical model for flits of traffic class-3 is quite accurate with respect to the waiting time obtained from the simulation. The average error in waiting time of traffic class-3 is 2% for the system shown in Figure 6.1(a), with a deterministic service time of two cycles.

## Transformation 2: Split at *Low* Priority Queue

**Conceptual Illustration:** Consider the queuing system shown in Figure 6.4(a). In this system, class-1 flits ($\lambda_1$) are waiting in $Q_{high}$, while class-2 flits ($\lambda_2$) and class-3 flits ($\lambda_3$) are waiting in $Q_{low}$. Class-1 and class-3 flits share the same channel and compete for the same output, while class-2 flits are sent to a separate output. Class-1 flits always win the arbitration since they have higher priority. Similar to the previous transformation, the queuing model in Figure 6.4(b) is used as an intermediate

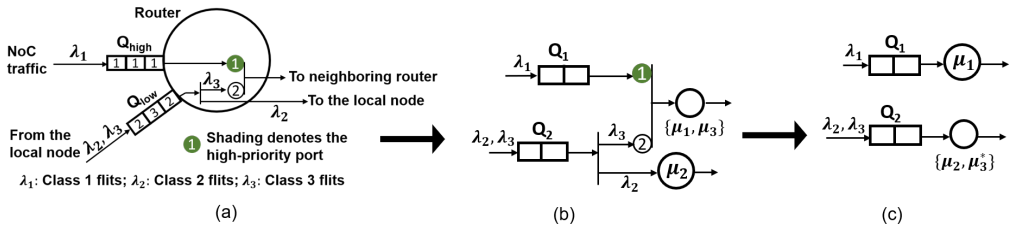



Figure 6.4: Split at low priority: Service Rate Transformation. $\mu^*$ denotes transformed service rate. The waiting time of class-1 flits depends on the residual time of the class-3 flits, as shown in Equation 6.4.

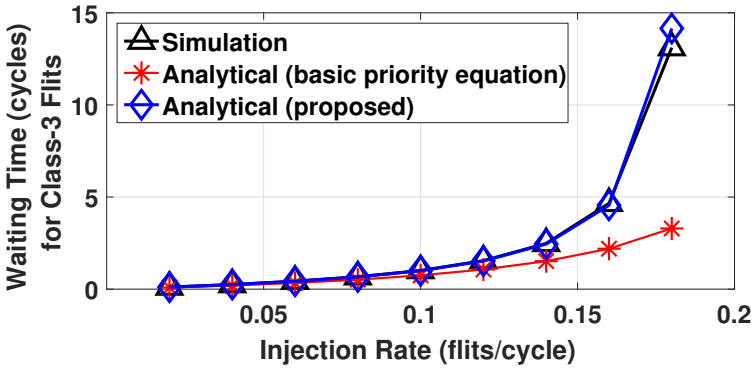



Figure 6.5: Comparison of simulation with the basic priority-based queuing model and proposed analytical model.

representation to facilitate the discussion. In this system, $Q_{high}$ and $Q_{low}$ are represented as $Q_1$ and $Q_2$ respectively.

If we ignore the impact of class-1 traffic while modeling the waiting time for class-3, the resulting analytical models will be highly optimistic, as shown in Figure 6.5. Accounting for the impact of class-1 traffic on class-2 is challenging, since only fraction of the flits in $Q_2$ that compete with class-1 are blocked. In other words, class-2 flits which go to the local node are not directly blocked by class-1 flits. Hence, there is a need for a new transformation that can address the split at the low-priority queue.

**Proposed Transformation:** The high-priority flow (class-1) is not affected by class-2 traffic since they do not share the same server. Therefore, the waiting time of class-1 flits can be readily obtained using Equation 2.1 as:

$$W_1 = \frac{R_1 + R_3}{1 - \rho_1} \tag{6.4}$$

Hence, we represent $Q_1$ as a stand-alone queue, as shown in Figure 6.4(c). However, the opposite is not true; class-1 flits affect both class-2 (indirectly) and class-3 (directly). Therefore, we represent them using a new queue with modified service rate statistics. To ensure that Figure 6.4(c) closely approximates the original system, we characterize the effect on the service

rate of class-3 using a novel analytical model.

**Proposed Analytical Model:** Both the service time and residual time of class-3 change due to the interaction with class-1. To quantify these changes, we set $\lambda_2 = 0$ such that the effect of class-2 is isolated. In this case, the waiting time of class-3 flits can be found using Equation 2.1 as:

$$W_3\Big|_{\lambda_2=0} = \frac{R_1 + R_3 + \rho_1 + \rho_1 W_1}{1 - \rho_1 - \rho_3} \tag{6.5}$$

We can find $W_3$ also by using the modified service time $(T_3^*)$ and residual time $R_3^*$ of class-3. The probability that a class-3 flit cannot be served due to class-1 is equal to server utilization $\rho_1$. Moreover, there will be extra utilization due to the residual effect of class-3 on class-1, i.e., $\lambda_1 R_3$ flits in $Q_1$. Hence, the probability that a class-3 flit is delayed due to class-1 flits is:

$$p = \rho_1 + \lambda_1 R_3 \tag{6.6}$$

Each time class-3 flit is blocked by the class-1 flits, the extra delay will be $T_1$, i.e., class-1 service time. Since each flit can be blocked multiple consecutive times, the additional busy period of serving class-3 $(\Delta T_3)$ is expressed as:

$$\begin{aligned} \Delta T_3 &= T_1 p(1-p) + 2T_1 p^2(1-p) + .... + nT_1 p^n(1-p) + \cdots \\ &= T_1 \frac{p}{1-p} \end{aligned} \tag{6.7}$$

Consequently, the modified service time $(T_3^*)$ and utilization $(\rho_3^*)$ of class-3 can be expressed as:

$$\begin{aligned} T_3^* &= T_3 + \Delta T_3 \\ \rho_3^* &= \lambda_3 T_3^* \end{aligned} \tag{6.8}$$

Suppose that the modified residual time of class-3 is denoted by $R_3^*$.

We can plug $R_3^*$, the modified utilization $\rho_3^*$ from Equation 6.8, and the additional busy period $\Delta T_3$ from Equation 6.7 into Geo/G/1 model to express the waiting time $W_3$ as:

$$W_3 = \frac{R_3^*}{1 - \rho_3^*} + \Delta T_3 \tag{6.9}$$

When $\lambda_2$ is set to zero, this expression should give the class-3 waiting time $W_3\Big|_{\lambda_2=0}$ found in Equation 6.5. Hence, we can find the following expression for $R_3^*$ by combining Equation 6.5 and Equation 6.9:

$$R_3^* = (1 - \rho_3^*)(W_3\Big|_{\lambda_2=0} - \Delta T_3) \tag{6.10}$$

Since the modified service time and residual times are computed, we can apply the Geo/G/1 queuing model one more time to find the waiting time of class-2 and class-3 flits as:

$$W_2 = \frac{R_3^* + R_2}{1 - \rho_3^* - \rho_2}$$
$$W_3 = \frac{R_3^* + R_2}{1 - \rho_3^* - \rho_2} + \Delta T_3 \tag{6.11}$$

Figure 6.5 shows that the class-3 waiting time calculated using the proposed analytical modeling technique is very close to simulation results. The modeling error is within 4% using a deterministic service time of 2 cycles.

## 6.3 Generalization for Arbitrary Number of Queues

In this section, we show how the proposed transformations are used to generate analytical models for priority-based NoCs with arbitrary topologies

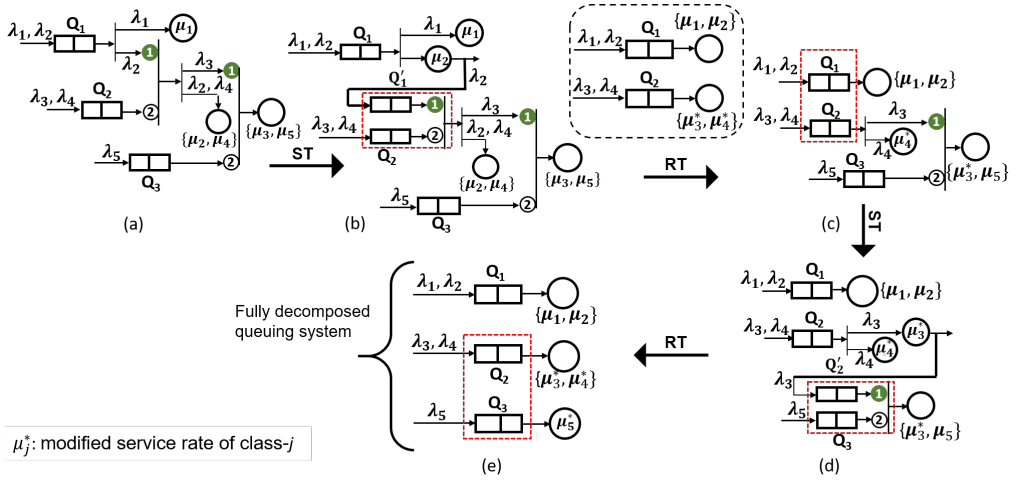



Figure 6.6: Applying the proposed methodology on a representative segment of a priority-based network. ST and RT denote Structural and Service Rate Transformation, respectively. Red-dotted squares show the transformed part from the previous step. Figure (a) shows the original queuing system. After applying ST on $Q_1$, we obtain the system shown in Figure (b). The system in Figure (c) is obtained by applying RT on $Q_2$. ST is applied again on $Q_2$ to obtain the system shown in Figure (d). Finally, RT is applied on $Q_3$ to obtain the fully decomposed queuing system shown in Figure (e).

and input traffic. Algorithm 1 describes the model generation technique, which is a part of the proposed methodology to be used in a virtual platform. This algorithm takes injection rates for all traffic classes, the NoC topology, and the routing of individual traffic classes. Then, it uses the transformations described in Section 6.2 and Section 6.2 iteratively to construct analytical performance models for each traffic class.

First, Algorithm 1 extracts all traffic classes originating from a particular queue, as shown in line 6. Next, the waiting time for each of these classes is computed separately, as each has a different dependency on other classes due to priority arbitration. At line 8, all classes that have higher priority than the current class are obtained. In lines 11–16, the

**Algorithm 1:** End-to-end queuing time calculation for different traffic classes

**Input:** Injection rates for all traffic classes, NoC topology and Traffic routing pattern
**Output:** Queuing time for all traffic classes
**for** *n = 1: no. of queues* **do**
    For queuing time expression of the current queue:
    Initialize: $num_1 \leftarrow 0$, $den_1 \leftarrow 1$
    Get all classes in current queue
    **for** *i = 1: no. of classes* **do**
        Get all higher priority classes than current class
        For reference queuing time ($W_{ref}$) of current class:
        Initialize: $num_2 \leftarrow R_i$, $den_2 \leftarrow (1 - \rho_i)$
        **for** *j = 1:no. of higher priority classes* **do**
            Calculate coefficient of variation ($C_D$) for current high priority class
            Calculate queuing time expression ($W_{ij}$) and residual time expression ($R_{ij}$) using $C_D$ using Eq. 6.1, Eq. 6.2, and Eq. 6.3
            $num_2 \leftarrow num_2 + R_{ij} + \rho_{ij} + W_{ij}\rho_{ij}$
            $den_2 \leftarrow den_2 - \rho_{ij}$
        **end**
        $W_{ref} = \frac{num_2}{den_2}$ (Eq. 6.5)
        Modify service rate ($T_i^*$) of $i^{th}$class
        Calculate residual time ($R_i^*$) using $T_i^*$ and $W_{ref}$ using Eq. 6.10
        $num_1 \leftarrow num_1 + R_i^*$
        $den_1 \leftarrow den_1 - \rho_i^*$
    **end**
    Queuing time of class-i in $n^{th}$ queue = $\frac{num_1}{den_1} + \Delta T_i$
**end**

structural transformation as described in Section 6.2 is applied. For that, the coefficient of variation of inter-departure time ($C_D$) for each of the higher priority classes is computed. Through structural transformation,

reference waiting time ($W_{ref}$) for the current class is obtained, as depicted in line 17 of the algorithm. At line 18, we compute the modified service time ($T_i^*$) of the current class following the method described in Section 6.2. Using $T_i^*$ and $W_{ref}$, the residual time ($R_i^*$ in line 19) is computed. Using residual time expressions for all classes in a queue, we obtain waiting time expressions for each class separately, as shown in line 23 of the algorithm.

Figure 6.6 illustrates the proposed approach on a representative example of a priority-based network to decompose the system. Figure 6.6(a) shows the original queuing network. This network consists of three queues: $Q_1$, $Q_2$, and $Q_3$. $Q_1$ stores flits from class-1 and class-2 flows, while $Q_2$ buffers class-3 and class-4 flits. Flits of class-2 have higher priority than both class-3 and class-4, as denoted by the first port of the switch that connects these flows. Finally, class-5 flits are stored in $Q_3$. We note that class-5 flits have lower priority than that of class-3, while they are independent of class-2 and class-4 flits. To solve this queuing system, we first apply the structural transformation on class-1 and class-2 of $Q_1$ by bypassing class-2 flits to $Q_1$ as shown in Figure 6.6(b). Next, the service rate transformation on class-3 and class-4 is applied to obtain modified service time ($\mu^*$). This transformation allows us to form the network by decomposing $Q_1$ and $Q_2$, as depicted in Figure 6.6(c). After that, structural transformation is applied on class-3 as flits of class-3 have higher priority than those of class-5. Finally, service rate transformation is performed on class-5 to achieve a fully decomposed system, which is shown in Figure 6.6(e).

**Automation of Model Generation Technique:** We developed a framework to automatically generate the analytical performance model for NoCs with arbitrary size 2D Mesh and ring topologies. The proposed framework operates in two steps. In the first step, we extract all architecture-related information of the NoC. This includes information about the traffic classes in each queue and priority relations between classes. In the second step, the automation framework uses this architecture information to generate

analytical models.

## 6.4 Experimental Evaluations

### Experimental Setup

We applied the proposed analytical models to a widely used priority-based industrial NoC design [44]. We implemented the proposed analytical models in C and observed that on average it takes 0.66µs to calculate latency value per source-to-destination pair. At each router of the NoC, there are queues in which tokens wait to be routed. This NoC design incorporates deterministic service time across all queues. We compared average latency values in the steady state found in this approach against an industrial cycle-accurate simulator written in SystemC [85, 84]. We ran each simulation for 10 million cycles to obtain steady state latency values, with a warm-up period of 5000 cycles. Average latency values are obtained by averaging latencies of all flits injected after the warm-up period. Injection rates are swept from $\lambda_1$ to $\lambda_{max}$. Beyond $\lambda_{max}$, server utilization becomes greater than one, which is not practical. We show the average latency of flits as a function of the flit injection rate for different NoC topologies. We also present experimental results considering the cache coherency protocol with different hit rates, network topologies, and floorplans. With a decreasing hit rate, traffic towards the memory controller increases, leading to more congestion in the network.

### Full-System Simulations on gem5

Applications are profiled in the full-system simulator gem5 [10] using Linux 'perf' tools [24]. The 'perf' tool captures the time taken by each function call and their children in the gem5 source. It represents the statistics through a function call graph. From this call graph, we obtain

the time taken by the functions related to Garnet2.0, which is the on-chip interconnect for gem5. Figure 6.7 shows components of Garnet2.0, which takes up a significant portion of the total simulation time while running Streamcluster application on gem5. These components are router, network-link, and functional write. The 'other components' shown in Figure 6.7 consists of the functions not related to network simulation. We observe that the functional write takes 50%, and the whole network takes around 60% of the total simulation time in this case.

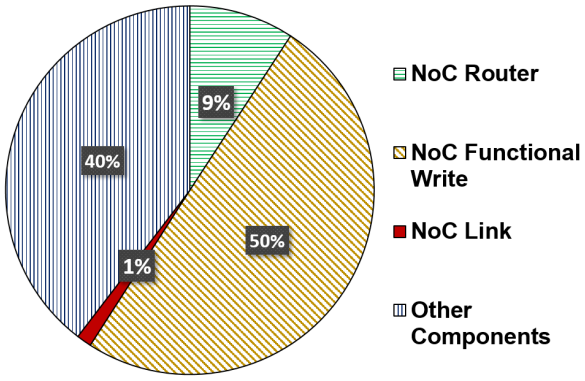



Figure 6.7: The fraction of simulation time spent by different functions while running Streamcluster in gem5. NoC-related functions take 60% of simulation time.

**Simulation Time:** To evaluate the decrease in simulation time with the proposed approach, we first run the Streamcluster application with a 16-core CPU on gem5 in full system mode using Garnet2.0, a cycle-accurate network simulator. Then, we repeat the same simulation by replacing the cycle-accurate simulation with the proposed analytical model. The total simulation time is reduced from 12,466 seconds to 4986 seconds when we replace the cycle-accurate NoC simulations with the proposed analytical models. Hence, we achieve a 2.5$\times$ speedup in cycle-accurate full-system simulation with the proposed NoC performance analysis technique.

## Validation on Ring Architectures

This section evaluates the proposed analytical models on priority-based ring architecture that consists of eight nodes. In this experiment, all nodes inject flits with an equal injection rate. Flits injected from a node go to other nodes with equal probability. We obtain the latency between each source-destination pair using the proposed analytical models. The simulation and analysis results are compared in Figure 6.8. The proposed analysis technique has only 2% error on average. The accuracy is higher at lower injection rates and degrades gradually with increasing injection rates, as expected. However, the error at the highest injection rate is only 5.2%.

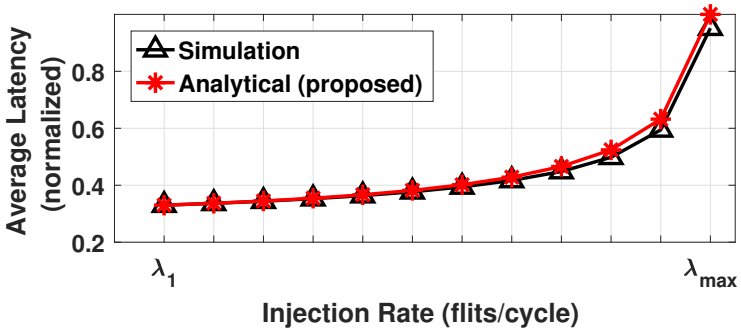



Figure 6.8: Evaluation of the proposed model on a ring with eight nodes.

## Validation on Mesh Architectures

This section evaluates the proposed analytical model for 6×6 and 8×8 priority-based mesh NoCs with Y-X routing. As described in [44], a mesh is a combination of horizontal and vertical half rings. The analytical model generation technique for priority-based NoC architecture is applied to horizontal and vertical rings individually. Then, these latencies, as well as the time it takes to switch from one to the other are used to obtain the latency for each source-destination pair. We first consider uniform random all-to-all traffic, as in Section 6.4. The comparison with the cycle-accurate

simulator shows that the proposed analytical models are on average 97% and 96% accurate for 6×6 and 8×8 mesh, as shown in Figure 6.9 and Figure 6.10, respectively. At the highest injection rate, the analytical models show 11% error for both cases.

**Comparison to Prior Techniques:** We compare the proposed analytical models to the existing priority-aware analytical models in literature [110]. Since these techniques do not consider multiple priority traffic classes in the network, they fail to accurately estimate the end-to-end latency. For example, Figure 6.9 and Figure 6.10 show that they overestimate NoC latency at high injection rates for 6×6 and 8×8 mesh networks, respectively. In contrast, since it captures the interactions between different classes, the proposed technique is able to estimate latencies accurately. Finally, we analyze the impact of using each transformation individually. If we apply only the Structural Transformation (ST), then the latency is severely underestimated at higher injection rates, since contentions are not captured accurately. In contrast, applying only Service Rate Transformation (RT) results in overestimating the latency at higher injection rates as the model becomes pessimistic.

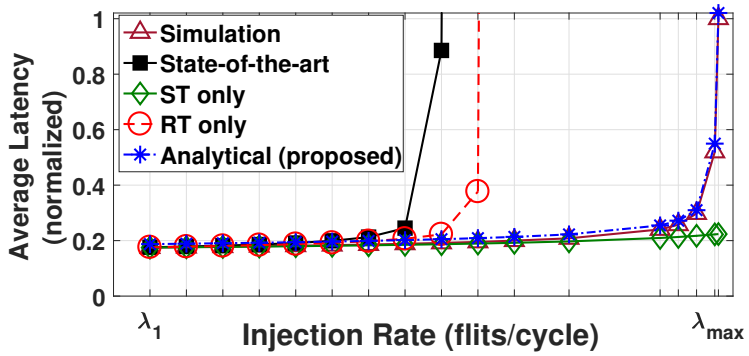



Figure 6.9: Evaluation of the proposed model on a 6×6 mesh.

**Impact of coefficient of variation:** One of the important parameters in

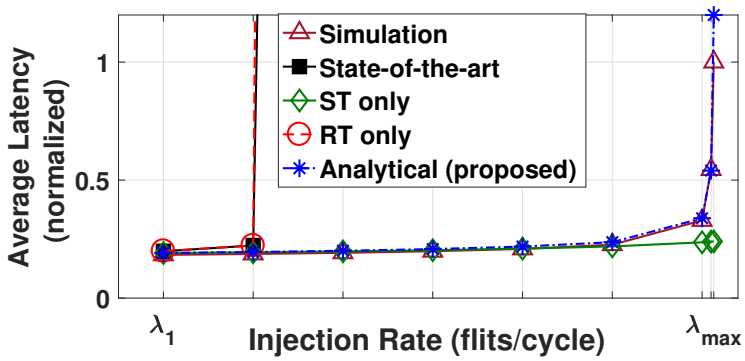



Figure 6.10: Evaluation of the proposed model on an 8×8 mesh.

our analytical model is the coefficient of variation of inter-arrival time. When the inter-arrival time between the incoming flits follows geometric distribution, increasing coefficient of variation implies larger inter-arrival time. Hence, the average flit latency is expected to decrease with an increasing coefficient of variation. Indeed, the simulation and analysis results demonstrate this behavior for a 6×6 mesh in Figure 6.11. We observe that the proposed technique accurately estimates the average latency in comparison to cycle-accurate simulation. On average, the analytical models are 97% accurate with respect to latency obtained from the simulation in this case.

**Evaluation with Intel® Xeon® Scalable Server Processor Architecture:**

This section evaluates the proposed analytical model with the floorplan of a variant of the Intel® Xeon® Scalable Server Processor Architecture [26] architecture. This version of the Xeon server has 26 cores, 26 banks of the last level cache (LLC), and 2 memory controllers. The cores and LLC are distributed on a 6×6 mesh NoC. The comparison of simulation and proposed analytical models with this floorplan is shown in Figure 6.12. On average, the accuracy is 98% when all cores send flits to all caches with equal injection rates. Similar to the evaluations on 6×6 mesh and

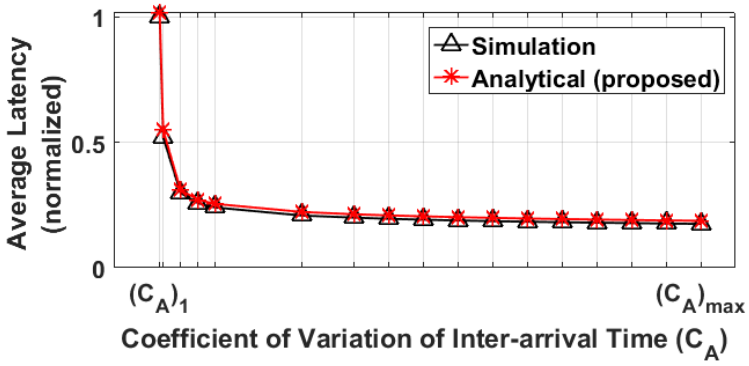



Figure 6.11: Effect of coefficient of variation of inter-arrival time on average latency for a 6×6 mesh.

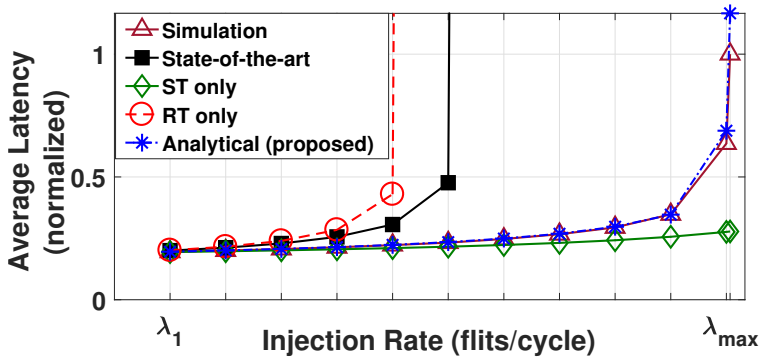



Figure 6.12: Evaluation of the proposed model on one variant of the Xeon server architecture.

8×8 mesh, the state-of-the-art NoC performance analysis technique [110] highly overestimates the average latency for this server architecture, as shown in Figure 6.12. Applying only ST underestimates the average latency and applying only RT overestimates the average latency.

The NoC latency is a function of the traffic class, since higher priority classes experience less contention. To demonstrate the latency for different classes, we present the NoC latencies for 9 representative traffic classes of the server architecture described above. Figure 6.13 shows the latency of each class of the server architecture described above normalized

Table 6.1: Accuracy for cache-coherency traffic flow

| LLC Hit Rate (%) | Accuracy for Address Network (%) | Accuracy for Data Network (%) |
|---|---|---|
| 100 | 98.8 | 93.9 |
| 50 | 97.7 | 98.1 |
| 0 | 97.7 | 98.0 |

with respect to the average latency obtained from the simulation. Higher priority classes experience lower latency, as expected. The proposed performance analysis technique achieves 91% accuracy on average for the classes which have the lowest priority in the NoC. For the classes having medium priority and highest priority, the accuracy is 99% on average. Therefore, the proposed technique is reliable for all classes with different levels of priority.

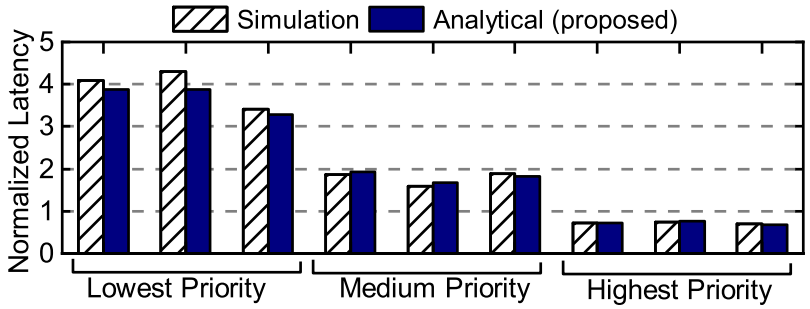



Figure 6.13: Per-class latency comparison for the server example.

Finally, we evaluate the proposed technique with different LLC hit rates. Table 6.1 shows that the proposed approach achieves over 97% accuracy in estimating the average latency of the address network for all hit rates. Similarly, the latencies in the data network are estimated with 98% or greater accuracy for 0% and 50% hit rates. The accuracy drops to 93.9% for 100% hit rates, since this scenario leads to the highest level of congestion due to all-to-all traffic behavior.

## Evaluation with Real Applications

In this section, evaluations of the proposed technique with real applications are shown. We use gem5 [10] to extract traces of applications in Full-System (FS) mode. Garnet2.0 [2] is used as the network simulator in gem5 with the Ruby memory system. Table 6.2 shows the various configuration settings we used for FS simulation in gem5.

We collect traces of six 16-threaded applications from PARSEC [9] benchmark suites: Blackscholes, Canneal, Swaptions, Bodytrack, Fluidanimate, and Streamcluster. We selected applications that show relatively higher network utilization as discussed in [114]. The accuracy obtained for these applications is an important indicator of the practicality of the proposed technique since real applications do not necessarily comply with a known inter-arrival time distribution [12], such as the geometric distribution used in this work. The traces are parsed and simulated through our custom in-house simulator with priority-based router model. For each application, a window of one million cycles with the highest injection rate is chosen for simulation. From the traces of these applications, we get the average injection rate of each source and destination pair. These injection

Table 6.2: Configuration settings in the gem5 simulation

| | | |
|---|---|---|
| **Processor** | Number of Cores | 16 |
| | Frequency of Cores | 2 GHz |
| | Instruction Set | x86 |
| **Interconnect Network** | Topology | 4x4 Mesh |
| | Routing Algorithm | X-Y deterministic |
| **Memory System** | L1 Cache | 16KB of instruction and data cache for each core |
| | Memory Size | 3 GB |
| **Kernel** | Type | Linux |
| | Version | 3.4.112 |

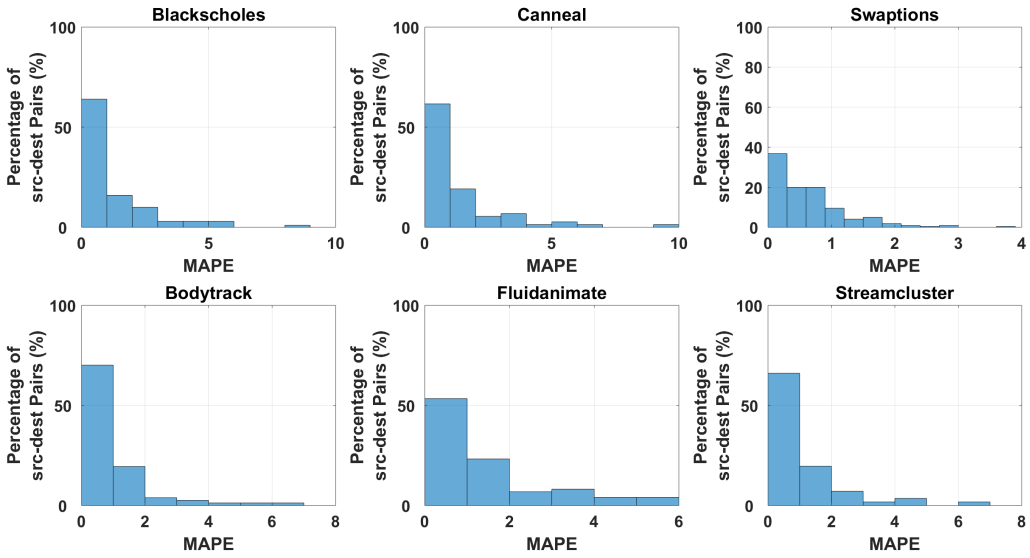



Figure 6.14: Model comparison for different applications from PARSEC suite.

rates are fed to our analytical models to obtain average latency.

Figure 6.14 shows the comparison of the average latency between the proposed analytical model and the simulation. The x-axes represent mean absolute percentage error (MAPE) between the average simulation latency ($L_{sim}$) and average latency obtained from analytical models ($L_{analytical}$). MAPE is defined by the following equation:

$$\text{MAPE} = 100\left(\frac{|L_{sim} - L_{analytical}|}{L_{sim}}\right) \tag{6.12}$$

The y-axes in the plots represent the percentage of source to destination pairs having the corresponding MAPE. From this figure, we observe that the latency obtained from the proposed analytical model is always within 10% of the latency reported by the cycle-accurate simulations. In particular, only 1% source-destination pair has MAPE of 10% for the Canneal appli-

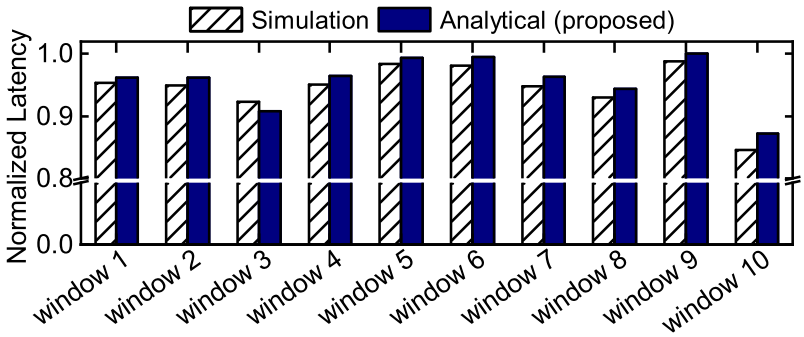



Figure 6.15: Evaluation of the proposed model under a finer level of time granularity (100K cycles) for Streamcluster application.

cation. On average, the analytical models have 3% error in comparison to latency obtained from the simulation for real applications. These results demonstrate that our technique achieves high accuracy for applications which may have arbitrary inter-arrival time distributions.

We further divide the window of one million cycles into 10 smaller windows containing 100,000 cycles each. Average latency comparison for Streamcluster application in these smaller windows is shown in Figure 6.15. The largest MAPE between latency obtained from the simulation and analytical model is observed for window 10, which is 7%. On average, the proposed analytical models are 98% accurate for these 10 windows. This confirms the reliability of the proposed analytical models at an even more granular level for the application. Finally, we note that the experiments with synthetic traffic shown in Section 6.4 and Section 6.4 exercise higher injection rates than these applications. Hence, the proposed technique performs well both under real application traces and heavy traffic scenarios.

Prior work showed that the deviation from Poisson distribution becomes larger as the network load approaches saturation [83]. Similar to this result, we also observe that the Geometric distribution assumption is a

good approximation until the NoC operates near saturation point. Therefore, we obtain high accuracy for real application workloads. Since this accuracy can degrade with increasing traffic load, we plan to generalize the proposed models by relaxing the assumption of Geometric distribution in our future work.

## 6.5 Conclusion

In this work, we propose an approach to build analytical models for priority-based NoCs with multi-class flits. As we emphasized, no prior work has presented analytical models that consider priority arbitration and multi-class flits in a single queue simultaneously. Such a priority-based queuing network is decomposed into independent queues using novel transformations proposed in this work. We evaluate the efficiency of the proposed approach by computing end-to-end latency of flits in a realistic industrial platform and using real application benchmarks. Our extensive evaluations show that the proposed technique achieves a high accuracy of 97% accuracy compared to cycle-accurate simulations for different network sizes and traffic flows.

# 7 APPENDIX B: PERFORMANCE ANALYSIS OF NOCS WITH BURSTY TRAFFIC

## 7.1 Related Work

NoC analytical performance analysis techniques primarily target fast design space exploration and accelerating full-system simulations. Most of the existing techniques consider NoC routers with fair arbitration [86, 96], but this assumption does not hold for NoCs that employ priority arbitration [44, 106].

Several performance analysis techniques target priority-aware NoCs [52, 73]. The technique presented in [52] assumes that each class of traffic in the NoC occupies different queues. This assumption is not practical since most of the industrial NoCs share queues between multiple traffic classes. Analytical model for industrial NoCs, which estimates average end-to-end latency is proposed in [73]. However, these models assume that the input traffic follows geometric distribution, which is not applicable for workloads with bursty traffic.

Analytical modeling of priority-based queuing networks has also been studied outside of the realm of the on-chip interconnect [13, 110]. Analytical models constructed in [13] considers a queuing network in the continuous-time domain. This assumption is not valid for NoCs, as events happen in discrete clock cycles. In [110], performance analysis models are constructed in the discrete-time domain. Since the number of random variables required in this technique is equal to the number of classes (exponential on the number of routers) present in the NoC, this approach does not scale. In contrast, the analytical models presented in this paper use the discrete-time domain and scale to thousands of traffic classes.

## 7.2 Proposed Approach to Handle Bursty Traffic

In industrial NoCs, flits already in the network have higher priority than new injections to achieve predictable latency [44]. This leads to nontrivial timing dependencies between the multi-class flits in the network. Hence, we propose a systematic approach for accurate and scalable performance analysis. We note that the proposed technique can be extended to NoCs with fair arbitration if we assume that all classes have the same priority. However, we do not focus on non-priority NoCs since this domain has been studied in the past [86].

### Maximum entropy for queuing networks

We apply the principle of ME to queuing systems to find the probability distribution of desired metrics (e.g., queue occupancy) [55]. According to this principle, the selected distribution should be *the least biased* among all feasible distributions satisfying the prior information in the form of mean values. The optimal distribution is found by maximizing the corresponding entropy function: we formulate a nonlinear programming problem and solve it analytically via the Lagrange method of undetermined multipliers as discussed next.

### Decomposition of basic priority queuing

In a non-preemptive priority queuing system, the router does not preempt a higher priority flit while processing a lower priority flit. An example system with two queues and a shared server is shown in Figure 7.1(a). There are two flows arriving at a priority-based arbiter and a shared server. The shaded circle corresponds to high priority input (class 1) to the arbiter. We denote this structure as *basic priority queuing*. Our goal is to decompose

this system into individual queue-nodes with modified servers, as shown in Figure 7.1(b). The combination of a queue and its corresponding server is referred to as a *queue-node*. The effective expected service time of class 2 flits, $\hat{T}_2$, is larger than the original mean service time $T_2$, since class 2 flits wait for the higher priority (class 1) flits in the original system. We calculate the effective service time in the transformed network using Little's Law as:

$$\widehat{T}_m = \frac{1 - p_m(0)}{\lambda_m} \tag{7.1}$$

where $p_m(0)$ is the marginal probability of having no flits of class $m$ in the queue-node, as listed in Table 7.1.

**Computing $p_m(0)$ using ME:** We find $p_m(0)$ using the ME principle by maximizing the entropy function $H(p(\mathbf{n}))$ given in (7.2) subject to the

Table 7.1: Summary of the notations used in this paper

| | |
|---|---|
| $\lambda, \lambda_m$ | Mean arrival rate of total traffic and class m |
| $p_b$ | Probability of burstiness |
| $T_m$, $\widehat{T}_m$ | Original and modified mean service time of class m flits |
| $R, R_{mk}$ | Total residual time and residual time of class m while class k is served |
| $\rho_m$ | Mean server utilization of class m flits ($=\lambda_m T_m$) |
| $C_a$, $C_{a_m}$ | Coeff. of variation of interarrival time of total traffic and class m flits |
| $C_{s_m}$, $\widehat{C}_{s_m}$ | Coeff. of variation of original and modified service time of class m flits |
| $C_d$, $C_{d_m}$ | Coeff. of variation of interdeparture time of total traffic and class m flits |
| $W_m$ | Mean waiting time of class m flits |
| $\overline{n}_m, n_m$ | Mean and current occupancy of class m flits in a queue-node |
| $\beta_m$ | Mean number of bursty arrivals of class m |
| $\overline{n}_{mk}$ | Mean queue-node occupancy of class $m$ with serving class $k$ |
| $\mathbf{n}$ | State vector, $\mathbf{n} = (n_1, n_2, ..., n_M)$ of priority queue-nodes |
| $p(\mathbf{n})$ | Probability that a queue-node is in state $\mathbf{n}$ |
| $p_m(0)$ | Marginal probability of zero flits of class m in a queue-node. |
| $\alpha_m(\mathbf{n})$ | $\alpha_m(\mathbf{n}) = 1$ if class $m$ in service and 0 otherwise |
| $M$ | Number of classes that share same server |

constraints listed in (7.3):

$$\underset{p}{\text{maximize}} \qquad H(p(\mathbf{n})) = -\sum_{\mathbf{n}} p(\mathbf{n}) \log(p(\mathbf{n})) \tag{7.2}$$

$$\begin{aligned} \text{subject to} \qquad & \sum_{\mathbf{n}=\mathbf{0}}^{\infty} p(\mathbf{n}) = 1, \\ & \sum_{\substack{\mathbf{n}=\mathbf{0} \\ \text{except} \\ n_m=1}}^{\infty} \alpha_m(\mathbf{n}) p(\mathbf{n}) = \rho_m, \ m = 1, \ldots, M \\ & \sum_{\substack{\mathbf{n}=\mathbf{0} \\ \text{except} \\ n_m=n_k=1}}^{\infty} n_m \alpha_k(\mathbf{n}) p(\mathbf{n}) = \bar{n}_{mk}, m, k = 1, .., M \end{aligned} \tag{7.3}$$

The notation $\infty$ means a state vector $\mathbf{n}$ with all elements set to $\infty$, and $(\mathbf{n} = \mathbf{0} \text{ except } n_m = 1)$ refers to a vector $\mathbf{n}$ with the $m^{th}$ element set to 1 and other elements set to 0. The constraints in (7.3) comprise three types: normalization, mean server utilization and mean occupancy. We introduced an extended set of mean occupancy constraints compared to [55] to provide further information about the underlying system. When a flit of a certain class arrives at the system, it may find the server busy with its own class or other classes since the server is a shared resource, as shown in Figure 7.1(a). Therefore, the mean occupancy of each class can be partitioned according to the contribution of each class occupying the server. We exploit this inherent partitioning to generate $M$ additional occupancy constraints. The occupancy related constraints depend on three components, $\beta_m$, $R_{mk}$ and $W_m$ (defined in Table 7.1) derived in [55, 73].

We solve the nonlinear programming problem in (7.2, 7.3) to find $p(\mathbf{n})$ which we use to determine the probability of having zero flits of class $m$, $p_m(0)$. The convergence of this solution is guaranteed when the queuing system is in a stable region. We derived the general expression for $M$

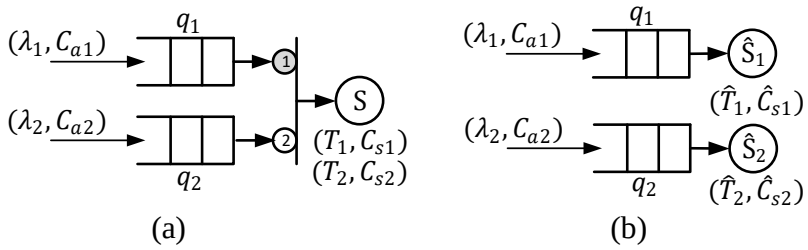



Figure 7.1: Decomposition of a basic priority queuing

queues in a priority structure with a single class per queue as:

$$p_m(0) = 1 - \rho_m - \sum_{k=1, k\neq m}^{M} \rho_k \frac{\overline{n}_{mk}}{\rho_k + \overline{n}_{mk}} \tag{7.4}$$

Plugging the expression of $p_m(0)$ from (7.4) into (7.1), we obtain the first moment of the service process.

**Computing second moment of the service time:** Since we also need the second moment to characterize the GGeo traffic, we calculate the modified squared coefficient of variation of the service time for class $m$ ($\widehat{C}^2_{s_m}$). We utilize the queuing occupancy formulation of GGeo/G/1 [55] and the modified server utilization $\widehat{\rho}_m = \lambda_m \widehat{T}_m$ to obtain the following expression for $\widehat{C}^2_{s_m}$:

$$\widehat{C}^2_{s_m} = \frac{(1 - \widehat{\rho}_m)(2\overline{n}_m - \widehat{\rho}_m) - \widehat{\rho}_m C^2_{a_m}}{\widehat{\rho}^2_m} \tag{7.5}$$

## Decomposition of priority queuing with partial contention

Priority-aware NoCs involve complex queuing structures that cannot be modeled accurately using only the models for basic priority queuing. The complexity is primarily attributed to the partial priority contention

across queues. We identified two basic structures with partial priority dependency that constitute the building blocks of practical priority-aware NoCs.

The first basic structure is shown in Figure 7.2(a) where high priority class 1 is in contention with a portion of the traffic in $q_2$ (class 2) through server $S_A$. Class 2 and 3 flits have the same priority and share $q_2$ before entering the traffic splitter that assigns class 2 and 3 flits to server $S_A$ and $S_B$ respectively, following a notation similar to the one adopted in [34]. We denote this structure as *contention at low priority*. To decompose $q_1$ and $q_2$, we need to calculate the first two moments of the modified service process of class 1 and 2. The decomposed structure is shown in Figure 7.2(b). First, we set $\lambda_3$ to zero which leads to a basic priority structure. Then, we apply the decomposition method discussed in Section 7.2 to obtain $(\widehat{T}_1, \widehat{C}_{s_1})$ and $(\widehat{T}_2, \widehat{C}_{s_2})$. We derived mean queuing time $(W_m)$ of individual classes of $q_2$ in the decomposed form as:

$$W_m = \frac{R + \sum_{k=1}^{M} \widehat{\rho}_k \widehat{T}_k \beta_k}{1 - \sum_{k=1}^{M} \widehat{\rho}_k} + \widehat{T}_m(\beta_m + 1) - T_m \tag{7.6}$$

where $R = \sum_{k=1}^{M} \frac{1}{2}\widehat{\rho}_k(\widehat{T}_k - 1 + \widehat{T}_k \widehat{C}_{s_k}^2)$ and $\beta_m = \frac{1}{2}(C_{A_m}^2 + \lambda_m - 1)$.

The other basic structure, *contention at high priority*, is shown in Figure 7.3(a). In this scenario, only a fraction of the classes in $q_1$ (class 2) has higher priority than class 3 since class 1 in $q_1$ is served by $S_A$. Determining $\widehat{T}_3$ is challenging due to class 1 that influences the inter-departure time of class 2. To incorporate this effect, we calculate the squared coefficient of variation of inter-departure time, $C_{d_2}^2$, of class 2 using the split process formulation of GGeo streams given in [55]. We introduce a virtual queue, $q_v$ and feed it with the flits of class 2. Therefore, $q_v$ and $q_2$ form a basic priority structure, as shown in Figure 7.3(b). Subsequently, we apply the decomposition method described in Section 7.2 to calculate $(\widehat{T}_3, \widehat{C}_{s_3})$ as well as $(\widehat{T}_2, \widehat{C}_{s_2})$. The decomposed structure is shown in Figure 7.3(c).

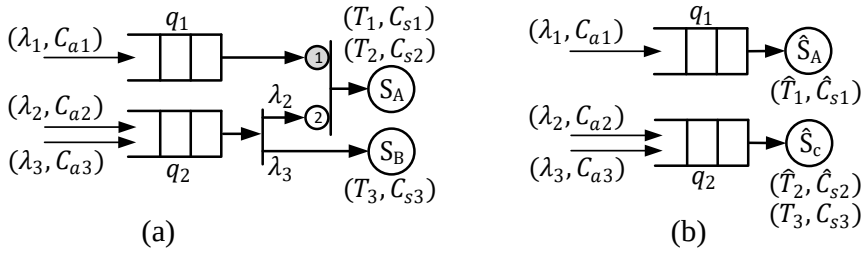



Figure 7.2: Decomposition of flow contention at low priority

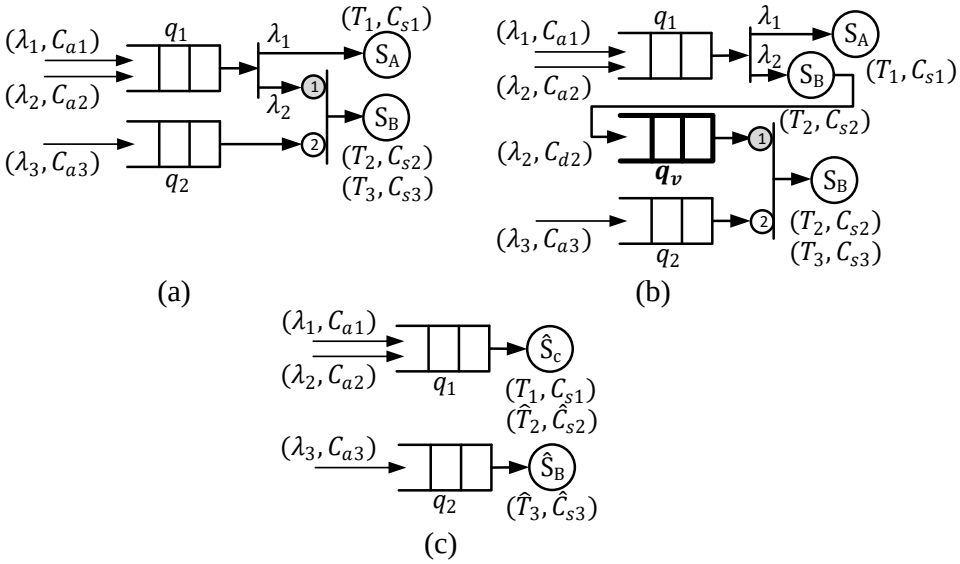



Figure 7.3: Decomposition of flow contention at high priority

## Iterative decomposition algorithm

Algorithm 2 shows a step-by-step procedure to obtain the analytical model using our approach described in Section 7.2. The inputs to the algorithm are NoC topology, routing algorithm and server process. The analytical models presented for the canonical queuing system are independent of

the NoC topology. Therefore, the analytical models are valid for any NoC, including irregular topologies. First, we identify priority dependencies between different classes in the network. Next, we apply decomposition for contention at high and low priority, as shown in line 7 – 8 of Algorithm 2. Subsequently, we calculate the modified service process ($\widehat{T}$, $\widehat{C}_s^2$) using (7.1, 7.4) and (7.5). Then, we compute the waiting time per class following (7.6). Finally, we obtain the average waiting time in each queue ($W_q$), as shown in line 12.

## 7.3 Experimental Results with Bursty Traffic

The proposed technique is implemented in C++ to facilitate integration with system-level simulators. Analysis takes 2.7 ms for a 6×6 NoC and the worst-case complexity is $O(n^3)$, where $n$ is the number of nodes. In all experiments, 200K cycles of warm-up period is considered. The accuracy of

**Algorithm 2:** Iterative Decomposition Algorithm

1 **Input:** NoC topology, routing algorithm, server process, ($\lambda$) and ($p_b$) for each class as parameters
2 **Output:** Average waiting time for each queue ($W_q$)
3 N = number of queues in the network
4 $S_q$ = set of classes in queue q
5 **for** $q$ = 1:N **do**
6  **for** $m = 1{:}|S_q|$ **do**
7  Apply decomp. for contention at high priority (if found)
8  Apply decomp. for contention at low priority (if found)
9  Compute $\widehat{T}$, $\widehat{C}_s^2$ using (7.1, 7.4) and (7.5)
10  Compute queuing time ($W_{q,m}$) using (7.6)
11  **end**
12  $W_q = \frac{\sum_{i=1}^{|S_q|} \lambda_{q,m} W_{q,m}}{\sum_{i=1}^{|S_q|} \lambda_{q,m}}$
13 **end**

Table 7.2: Comparisons against existing alternatives (Reference [52] and Reference [73]). *H* denotes errors over 100%.

| Topology | | 6×1 Ring | | | | | | | | | 8×1 Ring | | | | | | | | | 4×4 Mesh | | | | | | | | | 6×6 Mesh | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_b$ | | 0.2 | | | 0.4 | | | 0.6 | | | 0.2 | | | 0.4 | | | 0.6 | | | 0.2 | | | 0.4 | | | 0.6 | | | 0.2 | | | 0.4 | | | 0.6 | | |
| λ | | 0.1 | 0.4 | 0.6 | 0.1 | 0.4 | 0.6 | 0.1 | 0.4 | 0.6 | 0.1 | 0.3 | 0.5 | 0.1 | 0.3 | 0.5 | 0.1 | 0.3 | 0.5 | 0.2 | 0.5 | 0.8 | 0.2 | 0.5 | 0.8 | 0.2 | 0.5 | 0.8 | 0.1 | 0.4 | 0.6 | 0.1 | 0.4 | 0.6 | 0.1 | 0.3 | 0.6 |
| Err(%) | **Prop.** | **0.2** | **5.6** | **12** | **0.8** | **0.6** | **12** | **0.2** | **4.3** | **14** | **0.5** | **3.7** | **7.3** | **0.9** | **5.1** | **12** | **0.5** | **3.1** | **12** | **2.3** | **5.0** | **11** | **2.9** | **7.5** | **13** | **2.0** | **9.1** | **12** | **4.7** | **0.6** | **11** | **4.3** | **8.2** | **10** | **6.1** | **7.9** | **12** |
| | Ref[52] | *17* | *H* | *H* | *30* | *H* | *H* | *54* | *H* | *H* | *66* | *H* | *H* | *H* | *H* | *H* | *H* | *H* | *H* | *30* | *H* | *H* | *10* | *H* | *H* | *12* | *H* | *H* | *28* | *H* | *H* | *54* | *H* | *H* | *78* | *H* | *H* |
| | Ref[73] | 8.5 | 12 | 18 | 20 | 30 | 55 | 36 | 47 | 79 | 7.5 | 8.8 | 11 | 18 | 24 | 39 | 33 | 42 | 85 | 10 | 21 | 40 | 21 | 38 | 82 | 37 | 56 | 88 | 7.2 | 13 | 45 | 14 | 34 | 64 | 28 | 48 | 76 |

the models is evaluated against an industrial cycle-accurate simulator [84] under both real applications and synthetic traffic that models uniformly distributed core to last-level cache traffic with 100% hit rate.

## Evaluation on Architectures with Ring NoCs

This section analyzes the accuracy of the proposed analytical models using uniform traffic on a priority-based 6×1 and 8×1 ring NoCs, similar to those used in high-end client CPUs with integrated GPU and memory controller. Table 7.2 shows that the average errors between our technique and simulation are 6%, 4% and 6% for burst probability of 0.2, 0.4 and 0.6, respectively. These errors hardly reach 14% even at the highest injection, which is hard to model. Table 7.2 also shows that priority-based analytical models *which do not* consider burstiness [73] significantly underestimate the latency by 33% on average (highlighted with the shaded row). In contrast, the work without the proposed decomposition technique [52] leads to over 100% overestimation even at low traffic loads (highlighted with text in italics). In this case, GGeo models can not handle partial contention, since it assumes all packets in the high-priority queue have higher priority than each packet in the low priority queue. These results demonstrate that the proposed priority-aware NoC performance models have significantly higher accuracy than the existing alternatives.

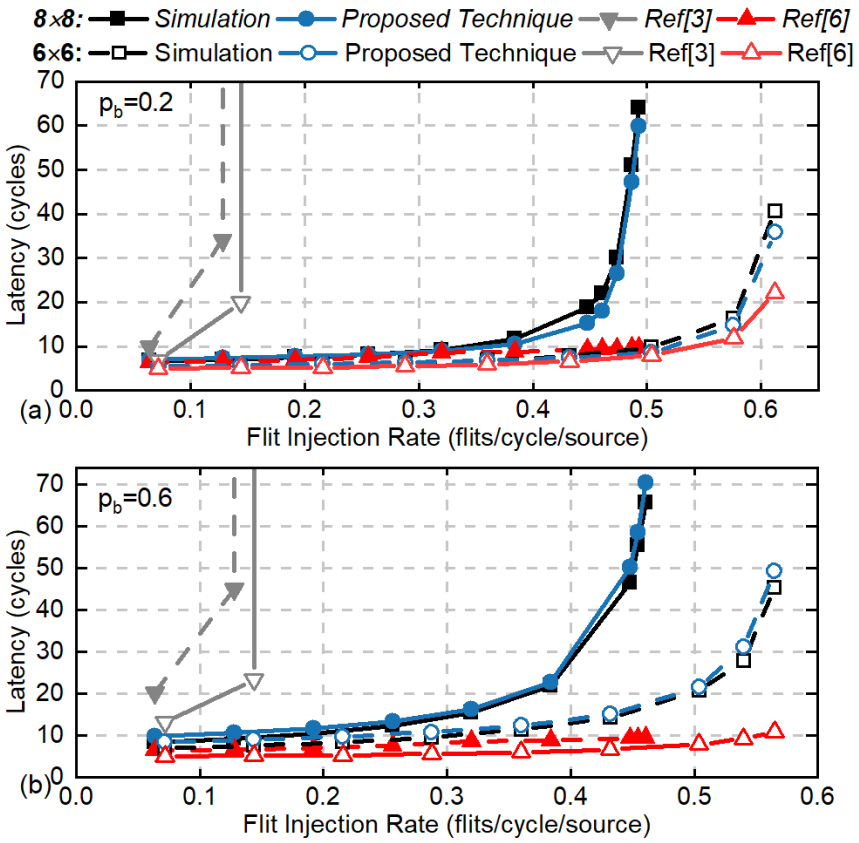



Figure 7.4: Comparison of a proposed analytical model with cycle-accurate simulation for 8×8 and 6×6 mesh for (a) $p_b = 0.2$ and (b) $p_b = 0.6$.

## Evaluation on Architectures with Mesh NoCs

Table 7.2 compares the analytical model and simulation results for a priority-based 4×4 and 6×6 mesh NoC, similar to those used in high-end servers [44]. Our technique incurs on average 6%, 7% and 10% error for burst probability of 0.2, 0.4 and 0.6, respectively. Priority-based analytical models which neglect burstiness [73] underestimate the latency by 60% on average similar to the results on the ring architectures. Likewise, GGeo models without the proposed decomposition technique lead to overestimation. We also provide detailed comparison of proposed an-

Table 7.3: Modeling Error (%) with Real Applications

| | | xalan-cbmk | mcf | gcc | bwaves | Gems FDTD | omnet-pp | perl-bench | SYSmark 14se |
|---|---|---|---|---|---|---|---|---|---|
| 6×6 Mesh | Prop | 2.17 | 4.97 | 0.92 | 0.15 | 0.38 | 5.10 | 3.63 | 0.73 |
| | Ref [52] | 14.62 | 11.99 | 7.69 | 12.29 | 5.18 | 13.64 | 11.46 | 7.25 |
| | Ref [73] | 17.36 | 23.29 | 7.71 | 22.02 | 6.99 | 14.11 | 12.95 | 11.13 |
| 8×8 Mesh | Prop | 3.59 | 4.08 | 3.81 | 4.87 | 0.44 | 7.48 | 3.67 | 1.10 |
| | Ref [52] | 10.33 | 12.73 | 12.07 | 22.90 | 19.17 | 9.93 | 5.99 | 19.04 |
| | Ref [73] | 12.15 | 29.99 | 10.00 | 19.65 | 5.44 | 10.78 | 14.74 | 7.94 |

alytical models on 6×6 and 8×8 NoC for burst probability of 0.2 and 0.6 in Figure 7.4(a) and Figure 7.4(b), respectively. The proposed models significantly outperform the other alternatives and lead to less than 10% error on average.

## Evaluation with Real Applications

This section validates the proposed analytical models using SYSmark® 2014 SE [5], and applications from SPEC CPU® 2006 [38] and SPEC CPU® 2017 [16] benchmark suites. These applications are chosen since they show different levels of burstiness. First, we run these applications on gem5 [10] and collect traces with timestamps for each packet injection. Then, we use the traces to compute the injection rate ($\lambda$) and $p_b$.

**Computing** $p_b$: For each source, we feed traffic arrivals with timestamps over a 200K clock cycle window into a virtual queue with the same service rate as the NoC to determine the queue occupancy. At the end of the window, we compute the average occupancy. Then, we employ the model described in [55] to find the occupancy and then $p_b$ of each class.

The proposed analytical models are used to estimate the latency using the injection rate and burst parameters, as well as the NoC architecture and routing algorithm. The applications show burstiness in the range of 0.2 – 0.5. As shown in Table 7.3, the proposed technique has on average 2% and 4% error compared to cycle-accurate simulations for 6×6 mesh and 8×8 mesh, respectively. In contrast, the analytical models presented in [52] and [73] incur significant modeling error.

## 7.4 Conclusion

We presented analytical models for priority-aware NoCs under bursty traffic. We model bursty traffic as generalized geometric distribution and applied the maximum entropy method to construct analytical models. Experimental evaluations show that the proposed technique has less 10% modeling error with respect to cycle-accurate NoC simulation for real applications.

# 8 APPENDIX C: PERFORMANCE ANALYSIS OF NOCS WITH DEFLECTION ROUTING

## 8.1 Related Work

Deflection routing was first introduced in the domain of optical NoC as hot-potato routing [14]. Later, it was adapted for the NoCs used in high-performance SoCs to minimize buffer requirements and increase energy efficiency [80, 28, 29]. This routing mechanism always assigns the packets to a free output port of a router, even if the assignment does not result in minimum latency. This way, the buffer size requirement in the routers is minimized. Authors in [69] perform a thorough study on the effectiveness of deflection routing for different NoC topology and routing algorithm. Deflection routing is also used in industrial priority-aware NoC [44]. Since arbitrary deflections can cause livelocks and unpredictable latency, industrial priority-aware NoCs deflect the packets only at the destination nodes when the ingress buffer is full. Furthermore, the deflected packets always remain within the same row or column, and they are guaranteed to be sunk after a fixed number of deflections.

NoC performance analysis techniques have been used for design space exploration and architectural studies such as buffer sizing [83, 115, 92]. However, most of these techniques do not consider NoCs with priority arbitration and deflection routing, which are the key features of industrial NoCs [44]. Performance analysis of priority-aware queuing networks has also been studied for off-chip networks [8, 13, 110]. These analytical models consider the queuing networks in continuous time. However, each transaction in NoC happens at each clock cycle. Therefore, the underlying queuing system needs to be considered in the discrete time domain. A performance analysis technique for a priority-aware queuing network in discrete time domain is presented in [110]. However, this technique

suffers from high complexity for a complex queuing network, hence not applicable to industrial priority-aware NoCs.

A recent technique targets priority-aware NoCs [52], but it considers only a single class of packets in each queue of the network. In contrast, industrial priority-aware NoCs have multiple classes of packets that can exist in the same queue. NoCs with multiple priority traffic classes has recently been analyzed in [73]. However, this analysis assumes that the input traffic follows a geometric distribution. This technique has limited applications since industrial NoCs can experience bursty traffic. Furthermore, it does not consider deflection routing. Since deflection routing increases traffic congestion, it is crucial to incorporate this aspect while constructing performance models. An analytical bound on maximum delay in networks with deflection routing is presented in [15]. However, evaluating maximum delay is not useful since it leads to significant overestimation. Another analytical model for NoCs with deflection routing is proposed in [32]. The authors first compute the blocking probability at each port of a router using an M/G/1 queuing model. Then, they compute the contention matrix at each router port. The average waiting time of packets at each port is computed using the contention matrix. However, this analysis ignores different priority classes and applies to only continuous-time queuing systems.

In contrast to prior work, we propose a performance analysis technique that considers ***both priority-aware NoCs with deflection routing under bursty and high traffic load.*** The proposed technique applies the superposition principle to obtain the statistical distribution of the deflected packets. Using this distribution, it computes the average waiting time for each queue. To the best of our knowledge, this is the first analytical model for priority-aware industrial NoCs with deflection routing under high traffic load.

## 8.2 Proposed Superposition-based Approach

This section presents the proposed performance analysis technique for estimating the end-to-end latency for priority-aware NoCs with deflection routing. We first construct a model for a canonical system with a single traffic class, where the deflected traffic distribution is approximated using a GGeo distribution (Section 8.2). Subsequently, we introduce a scalable approach for a network with multiple traffic classes. In this approach, we first develop a solution for the canonical system. Then, employ the principle of superposition to extend the analytical model to larger and realistic NoCs with multiple traffic classes (Section 8.2). Finally, we propose an algorithm that uses our analytical models to compute the average end-to-end latency for a priority-aware NoC with deflection routing (Section 17).

### An Illustration with a Single Traffic Class

Figure 8.1(a) shows an example of a single class input traffic and egress queue that inject traffic to a network with deflection routing. The input packets are buffered in the egress queue $Q_i$ (analogous to the packets stored in the egress queue of Node 2 in Figure 2.7). We denote the traffic of $Q_i$ as class-i, which is modeled using GGeo distribution with two parameters $(\lambda_i, C_i^A)$. The packets in $Q_i$ are dispatched to a priority arbiter

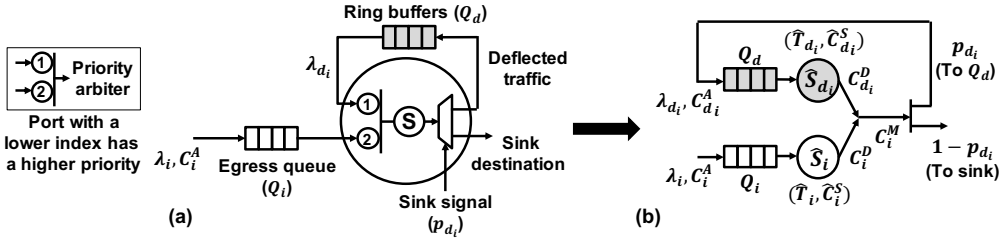



Figure 8.1: (a) Queuing system of a single class with deflection routing (b) Approximate queuing system to compute $C_{d_i}^A$.

Table 8.1: Summary of the notations used in this paper.

| | |
|---|---|
| $\lambda_i$ | Arrival rate of class-i |
| $p_{d_j}$ | Deflection probability at sink-j |
| $T_i$, $\widehat{T}_i$ | Original and modified mean service time of class-i |
| $\rho_i$ | Mean server utilization of class-i ($=\lambda_i T_i$) |
| $C_i^A$ | Coefficient of variation of inter-arrival time of class-i |
| $C_i^S$, $\widehat{C}_i^S$ | Coefficient of variation of original and modified service time of class-i |
| $C_i^D$ | Coefficient of variation of inter-departure time of class-i |
| $C_i^M$ | Coefficient of variation of inter-departure time of merged traffic of class-i |
| $W_i$ | Mean waiting time of class-i |

and assigned a low priority, marked with ②. In contrast, the packets already in the network have a high priority, which are routed to the port marked with ①. The packet traverses a certain number of hops (similar to the latency from the source router to the junction router in Figure 2.7) and reaches the destination. Since the number of hops is constant for a particular traffic class, *we omit these details in Figure 8.1(a) for simplicity*. If the ingress queue at the destination is full (with probability $p_{d_i}$), the packet is deflected back into the network. Otherwise, it is consumed at the destination (with probability $1 - p_{d_i}$). Deflected packets travel through the NoC (within the column or row as illustrated in Figure 2.7) and pass through the source router, but this time with higher priority. The profile of the deflected packets in the network is modeled by a buffer ($Q_d$) in Figure 8.1(a), since they remain in order and have a fixed latency from the destination to the original source. This process continues until the destination can consume the deflected packets.

Our goal is to compute the average waiting time $W_i$ in the source queue, i.e., components 1 and 3 of the end-to-end latency described in Section 2.3.

To obtain $W_i$, we first need to derive the analytical expression for the rate of deflected packets of class-$i$ ($\lambda_{d_i}$) and the coefficient of variation of inter-arrival time of the deflected packets ($C_{d_i}^A$) as follows.

**Rate of deflected packets** ($\lambda_{d_i}$): $\lambda_{d_i}$ is obtained by calculating the average number of times a packet is deflected ($N_{d_i}$) until it is consumed at the destination as:

$$\begin{aligned} N_{d_i} &= p_{d_i}(1-p_{d_i}) + 2p_{d_i}^2(1-p_{d_i}) + \ldots + np_{d_i}^n(1-p_{d_i}) + \ldots \\ &= \sum_{n=1}^{\infty} np_{d_i}^n(1-p_{d_i}) = \frac{p_{d_i}}{1-p_{d_i}} \end{aligned} \tag{8.1}$$

Therefore, $\lambda_{d_i}$ can be expressed as:

$$\lambda_{d_i} = \lambda_i N_{d_i} = \lambda_i \frac{p_{d_i}}{1-p_{d_i}} \tag{8.2}$$

**Coefficient of variation of inter-arrival time of deflected packets** ($C_{d_i}^A$)**:** To compute $C_{d_i}^A$, the priority related interaction between the deflected traffic of $Q_d$ and new injections in $Q_i$ must be captured. This computation is more involved due to the priority arbitration between the packets in $Q_d$ and $Q_i$ that involve a circular dependency. We tackle this problem by transforming the system in Figure 8.1(a) into an approximate representation shown in Figure 8.1(b) to simplify the computations. The idea here is to transform the priority queuing with a shared resource into separate queue nodes (queue + server) with a modified server process. This transformation enables the decomposition of $Q_d$ and $Q_i$ and their shared server into individual queue nodes with servers $\widehat{S}_d$ and $\widehat{S}_i$ respectively. The departure traffic from these two nodes merge at the destination, consumed with a probability $1-p_{d_i}$ and deflected otherwise.

The input traffic to the egress queue, as well as the deflected traffic, may exhibit bursty behavior. Indeed, the deflected traffic distribution can be bursty because of the server-process effect and the priority interactions

between the input traffic and the deflected traffic, even when the input traffic is not bursty. Therefore, we approximate the distribution of the deflected traffic via GGeo distribution. To compute the parameters of the GGeo traffic, we need to apply the principle of maximum entropy (ME) as shown in [55]. To obtain the modified service process of class-$i$, we first calculate the probability of no packets in $Q_i$ and in its corresponding server (i.e., $p_{Q_i}(0)$) using ME as,

$$p_{Q_i}(0) = 1 - \rho_i - \rho_{d_i} \frac{\overline{n}_i}{\overline{n}_i + \rho_i + \rho_{d_i}} \tag{8.3}$$

where $\rho_i$ and $\rho_{d_i}$ denote the utilization of the respective servers, and $\overline{n}_i$ is the occupancy of class-$i$ in $Q_i$. Next, we apply Little's law to compute the first order moment of modified service time ($\widehat{T}_i$) as:

$$\widehat{T}_i = \frac{1 - p_{Q_i}(0)}{\lambda_i} \tag{8.4}$$

Subsequently, we obtain the effective coefficient of variation $\widehat{C}_i^S$ as:

$$(\widehat{C}_i^S)^2 = \frac{(1 - \widehat{\rho}_i)(2\overline{n}_i + \widehat{\rho}_i) - \widehat{\rho}_i (C_i^A)^2}{\widehat{\rho}_i^2} \tag{8.5}$$

where $\widehat{\rho}_i = \lambda_i \widehat{T}_i$. We follow similar steps (Equation 8.3 – Equation 8.5) for the deflected traffic to obtain $\widehat{T}_{d_i}$ and $\widehat{C}_{d_i}^S$. With the modified service process, the coefficients of variation of inter-departure time of the packets in $Q_d$ ($C_{d_i}^D$) and $Q_i$ ($C_i^D$) are computed using the process merging method [93]. Then, we find the coefficient of variation ($C_i^M$) of the merged traffic from queues $Q_d$ and $Q_i$ as:

$$(C_i^M)^2 = \frac{1}{\lambda_{d_i} + \lambda_i} (\lambda_{d_i} (C_{d_i}^D)^2 + \lambda_i (C_i^D)^2) \tag{8.6}$$

We note that $C_i^M$ is a function of the coefficient of variation of the inter-arrival time of deflected traffic $C_{d_i}^A$. Since part of this merged traffic is

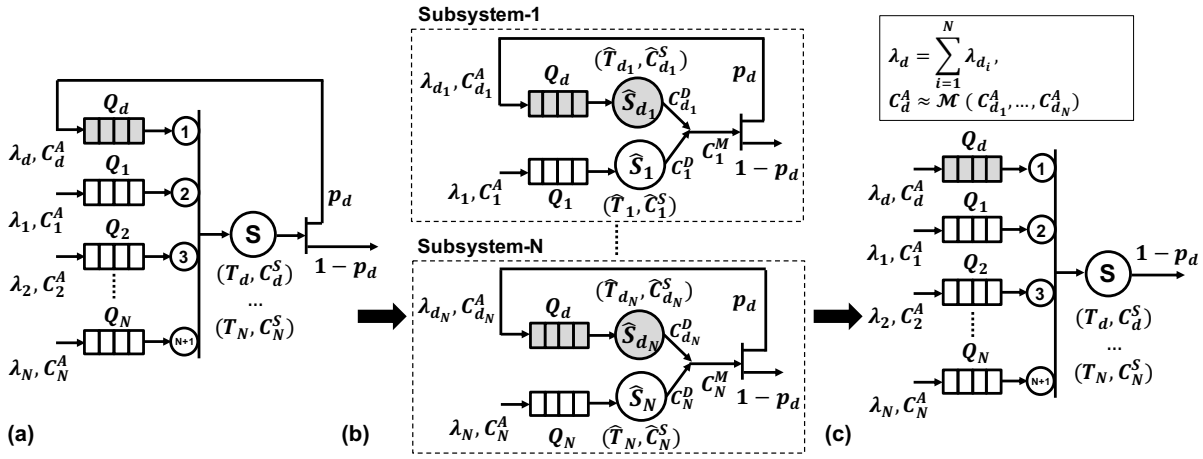



Figure 8.2: (a) Queuing system with N classes with deflection routing, (b) Decomposition into N subsystems to calculate GGeo parameters of deflected traffic per class, (c) Applying superposition to obtain the GGeo parameters of overall deflected traffic. $\mathcal{M}$ denotes the merging process.

consumed at the sink, we apply the traffic splitting method from [93] to approximate $C_{d_i}^A$ as:

$$(C_{d_i}^A)^2 = 1 + p_{d_i}((C_i^M)^2 - 1) \tag{8.7}$$

Finally, we extend the priority-aware formulations in continuous time domain [13] to discrete time domain to obtain the average waiting time of the packets in $Q_{d_i}$ and $Q_i$:

$$W_{d_i} = \frac{\rho_{d_i}(T_{d_i} - 1) + \rho_i(T_i - 1) + T_{d_i}((C_{d_i}^A)^2 + \lambda_{d_i} - 1)}{2(1 - \rho_{d_i})} \tag{8.8}$$

$$W_i = \frac{\rho_{d_i}(T_{d_i} + 1) + 2\rho_{d_i}W_{d_i} + \rho_i(T_i - 1) + T_i((C_i^A)^2 + \lambda_i - 1)}{2(1 - \rho_i - \rho_{d_i})} \tag{8.9}$$

## Queuing System with Multiple Traffic Classes

The analytical model for the system with a single class presented in Section 8.2 becomes intractable with a higher number of traffic classes. This section introduces a scalable approach based on the superposition principle that builds upon our canonical system used in Section 8.2.

Figure 8.2(a) shows an example with priority arbitration and N egress queues, one for each traffic class. *We note that this queuing system is a simplified representation of a real system*. The packets routed to port ⓘ have higher priority than those routed to port ⓙ for $i < j$. The deflected traffic in the network is buffered in $Q_d$, which has the highest priority in the queuing system. The primary goal is to model the queuing time of the packets of each traffic class. Modeling the coefficient of variations of the deflected traffic becomes harder since deflected packets interact with all traffic classes rather than a single class. These interactions complicate the analytical expressions significantly.

Priority arbitration enables us to sort the queues in the order at which the packets are served. The queue of the deflected packets has the highest priority, while the rest are ordered with respect to their indices. Due to this inherent order between the priority classes, their impact on the deflected traffic distribution can be approximated as being independent of each other. This property enables us to decompose the queuing system into multiple subsystems and model each subsystem separately, as illustrated in Figure 8.2(b). Then, we apply the principle of superposition to obtain the parameters of the GGeo distribution of the deflected traffic. Note that *each of these subsystems is identical to the canonical system analyzed in Section 8.2*. Hence, we first compute $\lambda_{d_i}$ and $C_{d_i}^A$ of each subsystem-$i$ following the procedure described in Section 8.2. Subsequently, we apply the superposition principle to $\lambda_{d_i}$ and $C_{d_i}^A$ for $i = 1 \dots N$ to obtain the GGeo distribution parameters of the deflected traffic $(\lambda_d, C_d^A)$.

In general, we obtain the GGeo distribution parameters of the deflected

traffic corresponding to class-$i$ by setting all traffic classes to zero expect class-$i$, ($\lambda_j = 0$, $j = 1 \ldots N, j \neq i$). The values of $\lambda_{d_i}$ and $C_{d_i}^A$ can be expressed as:

$$\lambda_{d_i} = \lambda_d\Big|_{\lambda_j=0, j\neq i; \lambda_i>0} \quad \text{and} \quad C_{d_i}^A = C_d^A\Big|_{\lambda_j=0, j\neq i; \lambda_i>0} \tag{8.10}$$

Subsequently, we apply the principle of superposition to obtain the distribution parameters of $Q_d$ as shown in Figure 8.2(c). First, we compute $\lambda_d$ by adding all $\lambda_{d_i}$ as:

$$\lambda_d = \sum_{i=1}^{N} \lambda_{d_i} \tag{8.11}$$

The value of $C_d^A$ is approximated by applying the superposition-based traffic merging process [93] for each $C_{d_i}^A$, as shown below:

$$(C_d^A)^2 = \sum_{i=1}^{N} \frac{\lambda_{d_i}}{\lambda_d} (C_{d_i}^A)^2 \tag{8.12}$$

Next, we use these distribution parameters $(\lambda_d, C_d^A)$ of the deflected packets to calculate the waiting time of the traffic classes in the system. The formulation of the priority-aware queuing system is applied to obtain the waiting time of each traffic class-$i$ ($W_i$) [8]:

$$\begin{aligned} W_i = & \frac{\rho_d(T_d+1) + 2\rho_i W_d}{2(1-\rho_d - \sum_{n=1}^{i}\rho_n)} + \frac{\sum_{n=1}^{i-1}(\rho_n(T_n+1) + 2\rho_i W_n)}{2(1-\rho_d - \sum_{n=1}^{i}\rho_n)} + \\ & \frac{\rho_i(T_i-1) + T_i((C_i^A)^2 + \lambda_i - 1)}{2(1-\rho_d - \sum_{n=1}^{i}\rho_n)} \end{aligned} \tag{8.13}$$

The first term in Equation 8.13 denotes the effect of deflected traffic on class-$i$; the second term denotes the effect of higher priority classes (class-$j$, $j < i$) on class-$i$; and the last term denotes the effect of class-$i$ itself. For

more complex scenarios that include traffic splits, we apply an iterative decomposition algorithm [72] to obtain the queuing time of different classes.

Figure 8.3 shows the average latency comparison between the proposed analytical model and simulation for the system in Figure 8.2. In this setup, we assume the number of classes is 5 ($N = 5$), $p_d = 0.3$, and input traffic distribution is geometric. The results show that the analytical model performs well against the simulation, with only 4% error on average. In contrast, the analytical model from [52] highly overestimates the latency as it does not consider multiple traffic classes. The performance model of the priority-aware NoC in [73] accounts for multiple traffic classes, but it does not model deflection. Hence, it severely underestimates the average latency.

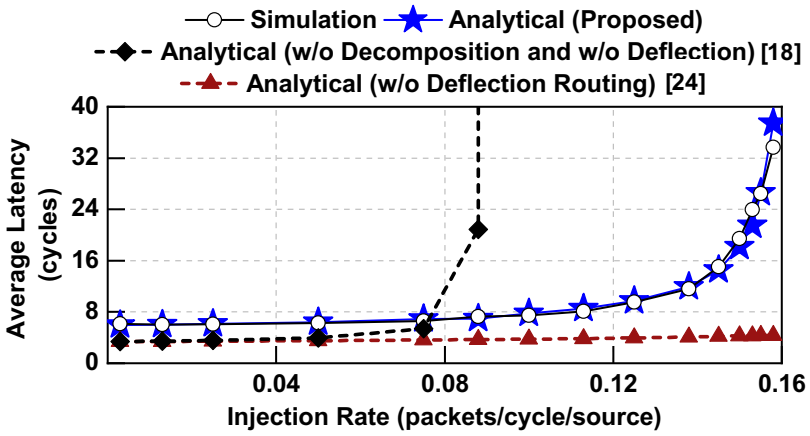



Figure 8.3: Comparison of average latency between simulation and analytical model for the canonical example shown in Figure 8.2 with $p_d = 0.3$ and $N = 5$.

**Algorithm 3:** End-to-end latency computation

**Input:** NoC topology, routing algorithm, service process, input distribution for each class, $(\lambda, C_A)$, deflection probability $(p_d)$ for each sink
**Output:** Average end-to-end latency $(L_{avg})$
$\mathcal{S}$ = set of all classes in the network
$N$ = number of queues in the network
$\mathcal{S}_n$ = set of classes in queue $n$
`/* Distribution of deflected traffic */`
**for** $i = 1$: $|\mathcal{S}|$ **do**
    Compute $\lambda_{di}$ and $C_{d_i}^A$ using Equation 8.10
    Compute $\lambda_d$ and $C_d^A$ using Equation 8.11 and Equation 8.12
**end**
Compute $W_d$ using $\lambda_d$ and $C_d^A$
`/* Average waiting time of each class */`
**for** $n = 1{:}N$ **do**
    **for** $s = 1{:}|\mathcal{S}_n|$ **do**
        Compute $W_{ns}$ using Equation 8.13 (if $|\mathcal{S}_n| = 1$)
        Compute $W_{ns}$ following the decomposition method in [72] (if $|\mathcal{S}_n| > 1$)
    **end**
**end**
$L_{avg} = \frac{\sum_{n=1}^{N}\sum_{s=1}^{\mathcal{S}_n}(W_{ns}+L_{ns})\lambda_{ns}}{\sum_{n=1}^{N}\sum_{s=1}^{\mathcal{S}_n}\lambda_{ns}}$ (*For mesh this term includes the latency both on the rows and the columns.*)

## Summary & End-to-End Latency Estimation

**Summary of the analytical modeling:** We presented a scalable approach for the analytical model generation of end-to-end latency that handles multiple traffic classes of priority-aware NoCs with deflection routing. It applies the principle of superposition on subsystems where each subsystem is a canonical queuing system of a single traffic class to significantly simplify the approximation of the GGeo parameters of deflected traffic and in turn, the latency calculations.

**End-to-End latency computation:** Algorithm 3 describes the end-to-end latency computation with our proposed analytical model. The input parameters of the algorithm are the NoC topology, routing algorithm, service process of each server, input traffic distribution for each class, and deflection probability per sink. It outputs the average end-to-end latency ($L_{avg}$). First, the queuing system is decomposed into multiple subsystems as shown in Figure 8.2(b) and $\lambda_{d_i}$ and $C_{d_i}^A$ for each subsystem-i are computed. Subsequently, the proposed superposition methodology is applied to compute $\lambda_d$ and $C_d^A$, shown in lines 6–9 of the algorithm. Then, $\lambda_d$ and $C_d^A$ are used to compute the average waiting time of the deflected packets ($W_d$). Then, the average waiting time for class-s in $Q_n$ ($W_{ns}$) is computed as shown in lines 13–14. The service time combined with static latency from source to destination ($L_{ns}$) is added to $W_{ns}$ to obtain the end-to-end latency. Finally, the average end-to-end latency ($L_{avg}$) is computed by taking a weighted average of the latency of each class, as shown in line 16 of the algorithm.

## 8.3 Experimental Results with Deflection Routing

This section validates the proposed analytical model against an industrial cycle-accurate NoC simulator under a wide range of traffic scenarios. The experiment scenarios include real applications and synthetic traffic that allow evaluations with varying injection rates and deflection probabilities. The evaluations include a 6×6 mesh NoC and a 6×1 ring as representative examples of high-end server CPUs [44] and high-end client CPUs [99], respectively. In both cases, the traffic sources emulate high-end CPU cores with a 100% hit rate on the shared last level cache (LLC) to load the NoCs. The target platforms are more powerful than experimental [109] and special-purpose [113] platforms with simple cores, although the mesh

size is smaller. To further demonstrate the scalability of the proposed approach, we also present results with mesh sizes up to 16×16. All cycle-accurate simulations run for 200K cycles, with a warm-up period of 20K cycles, to allow the NoC to reach the steady-state.

## Estimation of Deflected Traffic

One of the key components of the proposed analytical model is estimating the average number of deflected packets. This section evaluates the accuracy of this estimation compared to simulation with a 6×6 mesh. To perform evaluations under heavy load, we set the deflection probability at each junction and sink to $p_d = 0.3$ and injection rates at each source to 0.33

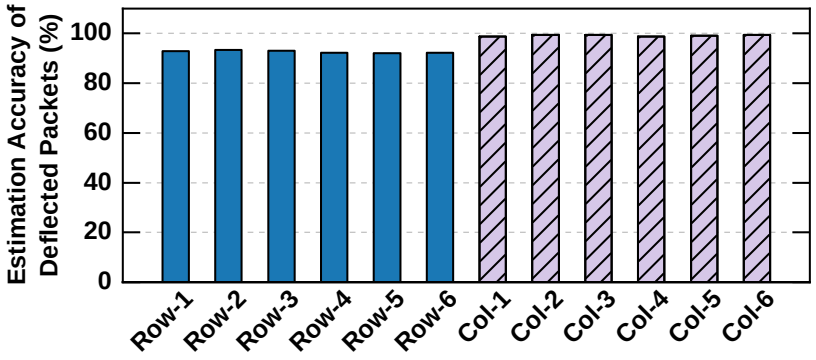



Figure 8.4: Estimation accuracy of average number of packets deflected for each row and column in a 6×6 mesh with $p_d$=0.3.

Table 8.2: Validation of the proposed analytical model for 6×6 mesh and 6×1 ring with bursty traffic arrival, and comparisons against prior work [52, 73]. 'E' signifies error >100%.

| **Topo.** | | **6×6 Mesh** | | | | | | | | | | | | | | | | | | **6×1 Ring** | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_d$ | | 0.1 | | | | | | 0.2 | | | | | | 0.3 | | | | | | 0.1 | | | | | | 0.2 | | | | | | 0.3 | | | | | |
| $p_{br}$ | | 0.2 | | | 0.6 | | | 0.2 | | | 0.6 | | | 0.2 | | | 0.6 | | | 0.2 | | | 0.6 | | | 0.2 | | | 0.6 | | | 0.2 | | | 0.6 | | |
| λ | | 0.1 | 0.3 | 0.4 | 0.1 | 0.3 | 0.4 | 0.1 | 0.3 | 0.4 | 0.1 | 0.3 | 0.4 | 0.1 | 0.2 | 0.3 | 0.1 | 0.2 | 0.3 | 0.1 | 0.3 | 0.4 | 0.1 | 0.3 | 0.4 | 0.1 | 0.3 | 0.4 | 0.1 | 0.3 | 0.4 | 0.1 | 0.2 | 0.3 | 0.1 | 0.2 | 0.3 |
| Err.(%) | **Prop.** | **7.3** | **9.6** | **8.1** | **14** | **13** | **14** | **8.9** | **8.0** | **7.7** | **13** | **12** | **12** | **9.6** | **9.2** | **6.5** | **11** | **12** | **13** | **1.0** | **4.1** | **5.8** | **4.6** | **5.2** | **5.5** | **0.7** | **2.3** | **4.2** | **6.3** | **7.3** | **8.6** | **0.7** | **0.9** | **3.3** | **6.3** | **8.5** | **8.6** |
| | Ref[52] | 2.6 | E | E | 26 | E | E | 22 | E | E | 39 | E | E | 35 | 18 | E | 57 | E | E | 7.0 | E | E | 34 | E | E | 23 | E | E | 45 | E | E | 42 | E | E | 54 | E | E |
| | Ref[73] | 12 | 15 | 23 | 3.1 | 18 | 23 | 28 | 41 | 65 | 19 | 33 | 49 | 42 | 45 | 55 | 39 | 35 | 31 | 15.3 | 18 | 22 | 18 | 24 | 33 | 30 | 38 | 67 | 31 | 44 | 54 | 41 | 50 | 73 | 42 | 50 | 58 |

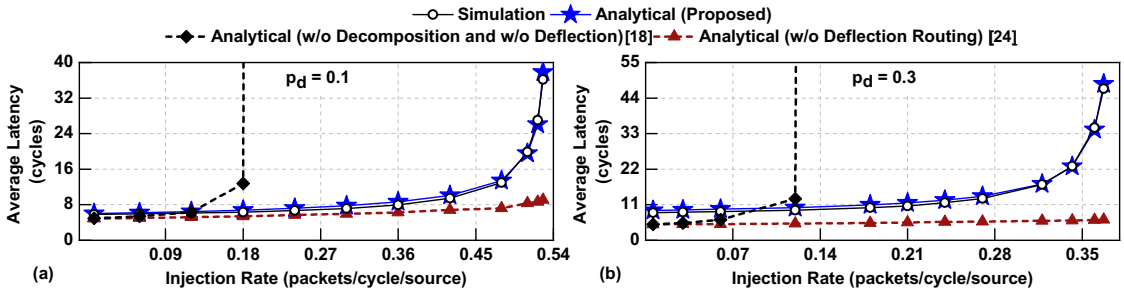



Figure 8.5: Comparison of average latency between simulation, the analytical model proposed in this work, and analytical models proposed in [52, 73] for a 6×6 mesh with deflection probability (a) 0.1 and (b) 0.3.

packets/cycle/source, which are relatively large values seen in actual systems. We first run cycle-accurate simulations to obtain the average number of deflected packets at each row and column of the mesh. Then, the analytical model estimates the same quantities for the 6×6 mesh. Figure 8.4 shows the estimation accuracy for all rows and columns. The average estimation accuracy across all rows and columns is 96% and the worst-case accuracy is 92%. Overall, this evaluation shows that the proposed model accurately estimates the average number of deflected packets.

## Evaluations with Geometric Traffic Input

This section evaluates the accuracy of our latency estimation technique when the sources inject packets following a geometric traffic distribution. We note that our technique can also handle bursty traffic, which is significantly harder. However, we start with this assumption to make a fair comparison to two state-of-the-art techniques from the literature [52, 73]. The model presented in [52] does not incorporate multiple traffic classes and deflection routing. On the other hand, the model presented in [73] considers multiple traffic classes but does not consider bursty traffic and deflection routing.

The evaluations are performed first on the server-like 6×6 mesh for deflection probabilities $p_d = 0.1$ and $p_d = 0.3$ while sweeping the packet injection rates. Figure 8.5(a) and Figure 8.5(b) show that the proposed technique follows the simulation results closely for all injections. More specifically, the proposed analytical model has only 7% and 6% percentage error on average for deflection probabilities of 0.1 and 0.3, respectively. In sharp contrast, the analytical model proposed in [52] significantly overestimates the latency starting with moderate injection rates, since it does not consider multiple traffic classes. Its performance degrades even further with larger deflection probability, as depicted in Figure 8.5(b). We note that it also slightly underestimates the latency at low injection rates since it ignores deflection. Unlike this approach, the technique presented in [73] considers multiple traffic classes in the same queue, but it ignores deflected packets. Consequently, it severely underestimates the latency impact of deflection, as shown in Figure 8.5.

We repeated the same evaluation on a 6×1 priority-aware ring NoC which follows a high-end industrial quad-core CPU with an integrated GPU and memory-controller [44]. The average error between the proposed analytical model and simulations are 7% and 4% for deflection probabilities of 0.1 and 0.3, respectively. In contrast, the model presented in [52] underestimates the latency at low injection rates and significantly overestimates it under high traffic load similar to the 6×6 results in Figure 8.5. Similarly, the analytical model presented in [73] severely underestimates the average latency. It leads to an average 43% error with respect to simulation. The plots of these results are not included for space considerations since they closely follow the results in Figure 8.5.

## Latency Estimation with Bursty Traffic Input

Since real applications exhibit burstiness, it is crucial to perform accurate analytical modeling under bursty traffic. Therefore, this section presents

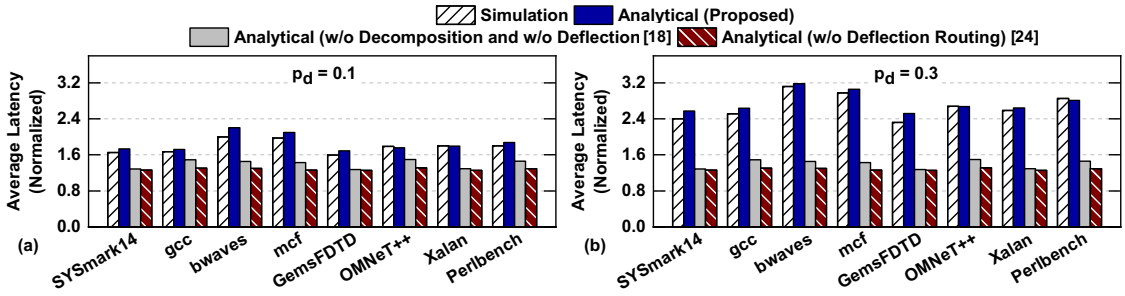



Figure 8.6: Average latency comparison between simulation, the analytical model proposed in this work, and analytical models proposed in [52, 73] for a 6×6 mesh with (a) $p_d = 0.1$ and (b) $p_d = 0.3$. modify the reference in the figure

the comparison of our proposed analytical model with respect to simulation under bursty traffic. For an extensive and thorough validation, we sweep the packet injection rate ($\lambda$), probability of burstiness ($p_{br}$), and deflection probability ($p_d$). The injection rates cover a wide range to capture various traffic congestion scenarios in the network. Likewise, we report evaluation results for two different burstiness ($p_{br} = \{0.2, 0.6\}$), and three different deflection probabilities ($p_d = \{0.1, 0.2, 0.3\}$). The coefficient of variation for the input traffic ($C_A$), the final input to the model, is then computed as a function of $p_{br}$ and $\lambda$ [54]. We simulate the 6×6 mesh and 6×1 ring NoCs using their cycle-accurate models for all input parameter values mentioned above. Then, we estimate the average packet latencies using the proposed technique, as well as the most relevant prior work [52, 73].

The estimation error of all three performance analysis techniques is reported in Table 8.2 for all input parameters. The mean and median estimation errors of our proposed technique are 9.3% and 9.5%, respectively. Furthermore, we do not observe more than 14% error even with relatively higher traffic load, probability of deflection, and burstiness than seen in real applications (presented in the following section). In strong contrast,

the analytical models proposed in [52] severely overestimate the latency similar to the results presented in Section 8.3. The estimation error is more than 100% for most cases since the impact of multiple traffic classes and deflected packet become more significant under these challenging scenarios. Similarly, the model proposed in [73] underestimates the latency because it does not model bursty traffic.

The right-hand side of Table 8.2 summarizes the estimation errors obtained on the 6×1 ring NoC that follows high-end client systems. In most cases, the error with the proposed analytical model is within 10% of simulation, and the error is as low as 1%. With $p_d = 0.1$, $p_{br} = 0.6$ and $\lambda = 0.4$, the error is 14%, which is acceptable, considering that the network is severely congested. In contrast, the analytical models proposed in [52] overestimate the latency, whereas the models in [73] underestimate the latency which conforms the results with geometric traffic, as in the 6×6 mesh results.

## Experiments with Real Applications

In addition to the synthetic traffic, the proposed analytical model is evaluated with applications from SPEC CPU®2006 [38], SPEC CPU®2017 benchmark suites [16], and the SYSmark®2014 application [5]. Specifically, the evaluation includes SYSmark14, gcc, bwaves, mcf, GemsFDTD, OMNeT++, Xalan, and perlbench applications. The chosen applications represent a variety of injection rates for each source in the NoC and different levels of burstiness. Each application is profiled offline to find the input traffic parameters. Of note, the probability of burstiness for these applications ranges from $p_b = 0.25$ to $p_b = 0.55$, which is aligned with the evaluations in Section 8.3.

The benchmark applications are executed on both 6×6 mesh and 6×1 ring architectures. The comparison of average latency between simulation and proposed analytical model for the 6×6 mesh is shown in Figure 8.6.

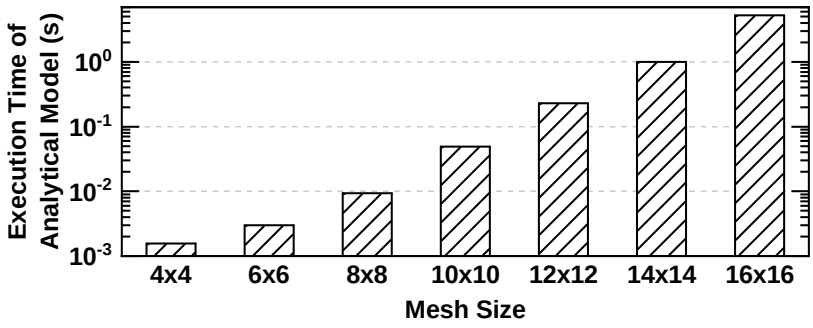



Figure 8.7: Execution time of the proposed analytical model (in seconds) for different mesh sizes.

The proposed model follows the simulation results very closely for deflection probability $p_d = 0.1$ and $p_d = 0.3$, as shown in Figure 8.6(a) and Figure 8.6(b), respectively. These plots show the average packet latencies normalized to the smallest latency from the $6\times1$ ring simulations due to the confidentiality of the results. On average, the proposed analytical model achieves less than 5% modeling error. In contrast, the analytical models which do not consider deflection routing [52, 73] underestimate the latency, since the injection rates of these applications are in the range of 0.02–0.1 flits/cycle/source (low injection region).

We observe similar results for the $6\times1$ ring NoC. The average estimation error of our proposed technique is less than 8% for all applications. In contrast, the prior techniques underestimate the latency by more than $2\times$ since they ignore deflected packets, and the average traffic loads are small. In conclusion, the proposed technique outperforms state-of-the-art for real applications and a wide range of synthetic traffic inputs.

## Scalability Analysis

Finally, we evaluate the scalability of the proposed technique for larger NoCs. We note that accuracy results for larger NoCs are not available

since they do not have detailed cycle-accurate simulation models. We implemented the analytical model in C. Figure 8.7 shows that the analysis completes in the order of seconds for up to 16×16 mesh. In comparison, cycle-accurate simulations take hours with this size, even considering linear scaling. When we scale the mesh size aggressively to 16×16, the analysis completes in about 5 seconds, which is orders of magnitude faster than cycle-accurate simulations of NoCs of this size.

## 8.4 Conclusion

Industrial NoCs incorporate priority-arbitration and deflection routing to minimize buffer requirement and achieve predictable latency. Analytical performance modeling of these NoCs is needed to perform design space exploration, fast system simulations, and tuning architectural parameters. However, state-of-the-art performance analysis models for NoCs do not incorporate priority arbitration and deflection routing together. This paper presented a performance analysis technique for industrial priority aware NoCs with deflection routing under heavy traffic. Experimental evaluations with industrial NoCs show that the proposed technique significantly outperforms existing analytical models under both real application and a wide range of synthetic traffic workloads.

# 9 APPENDIX D: COMMUNICATION-AWARE HARDWARE ACCELERATORS FOR DEEP NEURAL NETWORKS (DNNS)

## 9.1 Related Work

IMC-based hardware architectures have emerged as a promising alternative to conventional von-Neumann architectures. Prior works have proposed IMC hardware based on both SRAM and nanoscale non-volatile memory (e.g. resistive RAM or ReRAM) [100, 105, 79, 50, 120]. Authors in [100] proposed a ReRAM-based IMC architecture for DNN inference. The architecture utilizes a crossbar of size $128 \times 128$ to perform the Multiply-and-Accumulate (MAC) operations. They employ a parallel read-out method to accelerate the MAC computations in the analog domain. In addition, a Digital-to-Analog Converter (DAC) and an Analog-to-Digital Converter (ADC) is employed to switch between digital and analog domains. In contrast, [105] utilized spike-based computation to perform MAC operations in the time domain. Such an architecture does not need DAC and ADC units. An atomic computation-based ReRAM IMC architecture for both training and inference of DNNs is proposed in [97]. Authors in [79] proposed a systolic array-based ReRAM IMC design for DNN inference. Other works proposed in the past have explored SRAM-based IMC [120, 50].

All of the prior works focus on accelerating the computation while giving less importance to inter-layer data movement. Crossbar-based IMC hardware designs for DNNs significantly increase the volume of on-chip communications, making the role of on-chip interconnect crucial. Different on-chip interconnect solutions have been used for IMC-based DNN accelerators in the literature. Bus-based H-Tree interconnect is proposed in [18, 91]. However, bus-based interconnect contributes up to 90% of the total inference latency, as shown in Figure 2.9. Hence, a bus-based intercon-

nect does not provide a scalable solution for DNN accelerators. Shafiee *et al.* employs a concentrated mesh (cmesh)-based NoC to connect multiple tiles on-chip [100]. An IMC-based DNN accelerator for high-precision training is proposed in [43]. Ni *et al.* proposed a distributed in-memory computing architecture with a binary RRAM-based crossbar [81]. However, all these techniques assume a fixed interconnect architecture for different DNNs, i.e. these techniques do not cater to the communication needs of different DNNs.

The DNN accelerators presented in MAERI [61] and Eyeriss-v2 [19] use a flexible interconnect for DNN accelerators. In MAERI [61], a fat-tree-based programmable interconnect is used to support various sparse and non-sparse DNN dataflows. The work in [19] proposes a hierarchical mesh-NoC for DNNs with different operating modes to support different levels of data reuse in different dataflow patterns. However, these works do not consider the non-uniform weight distribution of different DNNs [70], DNN graph structure, and the computation-to-communication imbalance of the DNNs. A communication-centric IMC architecture is proposed in [57]. In this work, an optimization-based technique is incorporated to construct the schedules of a given DNN on mesh interconnect. Nonetheless, this architecture does not guarantee minimum possible communication latency for different DNNs.

In contrast to prior works, we propose an NoC architecture along with a scheduling technique that achieves the minimum possible communication latency for a given DNN. Specifically, our proposed approach takes different DNN parameters such as graph structure and weights distribution as input and constructs an NoC architecture with schedules ensuring minimum possible communication latency. This NoC is customized to a given DNN and does not inherit any of state-of-the-art topology (e.g. tree, mesh, torus, etc.). Furthermore, we propose a reconfigurable NoC architecture for two representative class of DNNs, namely, edge computing-based and

cloud computing-based DNNs. The reconfigurable architecture assumes that the IMC resources (IMC tiles and associated peripheral circuits) are available on-chip. Then, based on the DNN, the NoC architecture is reconfigured to obtain the lowest possible communication latency.

## 9.2 Area-aware NoC Optimization

**Energy-Aware Optimization for NoC:** As a result of the proposed area-aware optimization, the total number of tiles in an IMC architecture can be very high. For example, DenseNet (100,24) requires 1,088 tiles [18]. For such an architecture, one-to-one mapping of a router to tile [60] will require a large number of NoC routers and consume high power, as shown in Figure 9.1. Therefore, in our framework we introduce an energy-aware optimization for the NoC.

*Mapping Tiles to Routers:* We first construct an objective function that represents the NoC energy consumption. Let $n_k$ be the number of routers required for the $k^{th}$ layer of the DNN. The number of activations communi-

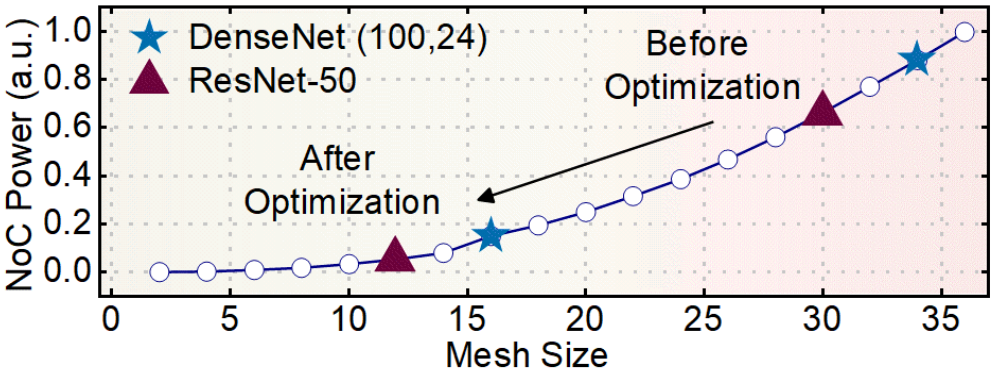



Figure 9.1: NoC optimization effectively reduces the power consumption because of its non-linear dependence on the mesh size. We obtain NoC power through BookSim [45] simulations.

cated between $n_k$ and $n_{k+1}$ routers is $I_k$. Hence, the number of activations between each source-destination pair is given by $I_k/(n_k \times n_{k+1})$. The total amount of communication volume can be found by adding this across all $K$ layers and routers:

$$E(\bar{\mathbf{n}}) = \Big(\sum_{k=1}^{K-1} \frac{I_k}{n_k n_{k+1}}\Big)\Big(\sum_{k=1}^{K} n_k\Big) \tag{9.1}$$

$E(\bar{\mathbf{n}})$ is proportional to the total communication energy of the DNN assuming that all transactions have a uniform size. We minimize this objective function with an upper bound on the total number of routers, $N$ as:

$$\begin{aligned} \underset{\bar{\mathbf{n}}}{\text{minimize}} \quad & E(\bar{\mathbf{n}}) \\ \text{subject to} \quad & n_k \geqslant 1;\ k = 1, \ldots, K, \\ & \sum_{k=1}^{K} n_k < N. \end{aligned} \tag{9.2}$$

where the first constraint ensures that each layer of the DNN is associated with at least one router. $N$ is a user-defined constraint (input to the optimization framework) that represents the maximum number of routers in the IMC architecture. At the end of this optimization, we obtain the number of routers needed for each layer ($n_k$) of the DNN.

*Packet Scheduling in NoC:* If the activations of a layer are injected into the NoC in the order of computation, there is a high possibility of congestion resulting in high communication latency in the NoC. Therefore, we propose a scheduling technique for the NoC to schedule the activations between two layers of the DNN. The scheduling technique is applied on top of the optimal tile-to-router mapping for the NoC. This scheduling technique provides a starting time for activations from each source to destination pair in the NoC. Without loss of generality, we assume that all activations for a particular source-destination pair can be injected back-to-back.

Using the NoC topology and the routing algorithm, we first find the source-destination pairs which contend for the same link in the NoC. We model each source-destination pair $(sd)$ as an individual task. The start time of the task corresponding to the pair $sd$ is denoted by $t_{sd}$ and the duration of the task equals to the number of packets for that pair $(n_{sd})$. Next, we put constraints on the start time of each task so that there is no contention between two transactions for the same link. The set of all tasks is denoted by $\mathcal{T}$ and the set of all non-overlapping tasks is denoted by $\mathcal{C}$. (9.3) shows the formulation of the non-overlap constraint, where the start time of two tasks is separated by the duration. Furthermore, the start time of all tasks are integers and greater than zero. We add one terminal task with the constraint that the start time of the terminal task $(t_{terminal})$ is greater than the start time of any of the source-destination pairs. We minimize $t_{terminal}$ to obtain the optimal schedule for all source-destination pairs.

$$
\begin{aligned}
\text{minimize} \quad & t_{terminal} \\
\text{subject to} \quad & t_{mn} > t_{pq} + n_{pq} \vee t_{pq} > t_{mn} + n_{pq}, \\
& \forall t_{mn}, t_{pq} \in \mathcal{C}, \\
& t_{xy} \geqslant 0, \forall\, t_{xy} \in \mathcal{T} \\
& t_{terminal} > t_{xy} + n_{xy}, \forall\, t_{xy} \in \mathcal{T}.
\end{aligned}
\tag{9.3}
$$

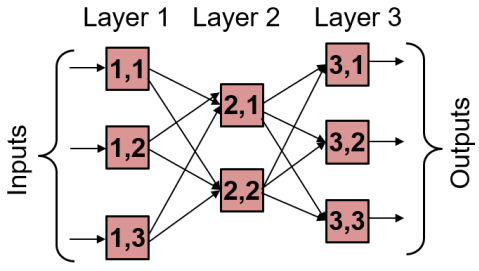
## 9.3 Latency-aware NoC optimization

### Determining Minimum Communication Latency

We aim to construct an NoC architecture, customized for different DNNs, which achieves minimum possible communication latency. To this end, we first show how the minimum possible communication latency between two consecutive layers of a given DNN for one round of communication

is achieved. In one round of communication, each source router of a layer sends one packet to each destination router in the next layer. Since each router sends/receives maximum one packet per cycle, the minimum possible communication latency to finish all the transactions is $\max(N_k, N_{k+1})$, where $N_k$ is the number of routers in $k^{th}$ layer.

Figure 9.2(a) represents an IMC hardware of a neural network with one hidden layer, where each square represents a tile. There are three tiles in the input layer (Layer 1), two tiles in the hidden layer (Layer 2), and three tiles in the output layer (Layer 3). We assume that there is an NoC router associated with each tile, and all routers have one input and one output port. We also assume layer-by-layer operation for the DNN, i.e. after all packets reach layer 2 from layer 1, then the communication between layer 2 and layer 3 will start. If each router of a layer is connected to every other router of the next layer (as shown in Figure 9.2(a)), then the minimum possible communication latency is achieved. We utilize below key insights to construct the schedules for obtaining minimum




(a)

| Layer 1 – Layer 2 | | Layer 2 – Layer 3 | |
|---|---|---|---|
| Link | Cycle | Link | Cycle |
| $L_{1,1} - L_{2,1}$ | 1 | $L_{2,1} - L_{3,1}$ | 4 |
| $L_{1,2} - L_{2,2}$ | 1 | $L_{2,2} - L_{3,2}$ | 4 |
| $L_{1,1} - L_{2,2}$ | 2 | $L_{2,1} - L_{3,2}$ | 5 |
| $L_{1,3} - L_{2,1}$ | 2 | $L_{2,2} - L_{3,1}$ | 5 |
| $L_{1,2} - L_{2,1}$ | 3 | $L_{2,2} - L_{3,2}$ | 6 |
| $L_{1,3} - L_{2,2}$ | 3 | $L_{2,1} - L_{3,3}$ | 6 |

(b)

Figure 9.2: (a) Communication between layers in a DNN and (b) The schedules for obtaining minimum communication latency between layers. Without loss of generality, it is assumed that the computation time in the tile is 0 cycles.

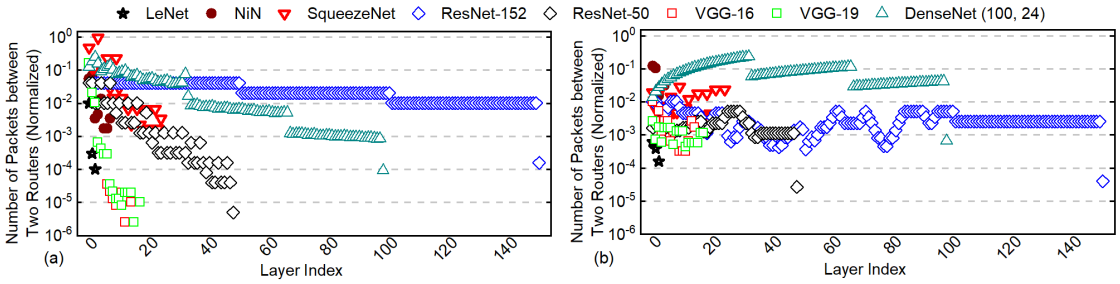



Figure 9.3: The number of packets between two routers with (a) one router per tile, and (b) the proposed technique. All the numbers are normalized with a factor of $4 \times 10^5$ (the highest number of packets per router between two layers with one router per tile, which occurs between $4^{th}$ and $5^{th}$ layer of SqueezeNet). The number of packets between routers decreases significantly with the proposed approach for all DNNs except DenseNet (100,24) and ResNet-50. However, the number of routers used for DenseNet with our proposed technique (300) is less than the number of routers required (1088) if one router is allocated per tile. Our technique achieves around 72% reduction in the NoC area. Similar improvement is seen for ResNet-50 as well.

communication latency.

*Key insight 1*: A single router can not send/receive more than one packet in a cycle. However, multiple routers can send/receive packets in parallel.

*Key insight 2*: Since a router can send/receive only one packet in a cycle, the congestion is minimum i.e., no more than a transaction is scheduled through a particular link.

Figure 9.2(b) shows the schedules for each round of communication between two layers to achieve the minimum possible communication latency. With these schedules, there is no congestion in the NoC, since no link is scheduled to carry more than one packet in a particular cycle. With this NoC topology, the number of links required (to achieve minimum possible communication latency) between the $k^{th}$ layer and $(k+1)^{th}$ layer is $N_k \times N_{k+1}$. Thus the number of links is $O(N^2)$, where N is the number of routers in a layer. The total number of links can be very large for DNNs

Table 9.1: Summary of the notations used in this paper.

| | |
|---|---|
| $A_k$ | Number of output activations in the $k^{th}$ layer |
| $N_k$ | Number of routers in the $k^{th}$ layer |
| $R_{k,n}$ | $n^{th}$ router in the $k^{th}$ layer |
| $R_{k_1,n_1} - R_{k_2,n_2}$ | The packet between $R_{k_1,n_1}$ and $R_{k_2,n_2}$ |
| $L_{k_1,n_1} - L_{k_2,n_2}$ | The link between $R_{k_1,n_1}$ and $R_{k_2,n_2}$ |

with a large number of routers per layer. For example, with this NoC topology, more than $1.5 \times 10^4$ links are required for DenseNet (100,24). Since the number of links increases the NoC area, this NoC topology is impractical to implement. To overcome this challenge, we first propose a technique to determine the optimal number of routers required for each layer of a DNN. In addition, we propose an NoC architecture that achieves the minimum possible communication latency between two consecutive layers of DNNs with minimum number of links.

## Determining the Optimal Number of Routers

The optimal number of routers in each layer of a DNN is a function of the number of packets that are sent from one layer to the next. Since a higher number of packets between two source-destination router pairs increase communication latency (due to higher congestion), determining the optimal number of routers for each DNN layer is crucial. We perform an analysis which shows that the inefficient distribution of routers can cause higher communication latency. In Figure 9.3(a), we show the number of packets between two routers for each layer of different DNNs when one router is allocated per tile. The number of packets is normalized with respect to the highest number of packets ($4 \times 10^5$), which occurs between two routers in the $4^{th}$ and $5^{th}$ layers of SqueezeNet. High number of packets between two routers increases congestion in the NoC, resulting in high communication latency.

Therefore, we propose a technique to determine the optimal number of routers required for each layer of the DNN. The objective function is shown in Equation (9.4). It is a function of the number of routers in each layer which is denoted by $\bar{\mathbf{N}}$, where $\bar{\mathbf{N}} = \{N_1, N_2, \ldots, N_K\}$. The number of activations between the $k^{th}$ and the $(k+1)^{th}$ layer is denoted by $A_k$. Therefore, if we divide $A_k$ by the product of $N_k$ and $N_{k+1}$, we obtain the number of activations between a pair of routers.

$$L(\bar{\mathbf{N}}) = \sum_{k=1}^{K-1} \left( \left\lceil \frac{A_k}{N_k N_{k+1}} \frac{Q}{W} \right\rceil \right) \max(N_k, N_{k+1}) \tag{9.4}$$

$$\begin{aligned} \underset{\bar{\mathbf{N}}}{\text{minimize}} \quad & L(\bar{\mathbf{N}}) \\ \text{subject to} \quad & N_k > 0; k = 1, \ldots, K, \\ & \sum_{k=1}^{K} N_k < N. \end{aligned} \tag{9.5}$$

Furthermore, after multiplying the expression by the bit precision $(Q)$ and dividing it by the NoC-bus width $(W)$, we obtain the number of packets between a pair of routers $(\frac{A_k}{N_k N_{k+1}} \frac{Q}{W})$. To convert the value to integer we take the ceiling of the expression. The minimum possible latency to finish each round of communication is $\max(N_k,\ N_{k+1})$ cycles as shown in Section 9.3, and the number of rounds equals the number of packets between a pair of routers. Therefore, multiplying these two terms we obtain the total communication latency of the DNN with the proposed NoC architecture (Equation (9.4)). At this point, the number of routers for each layer is unknown. Therefore, we minimize the objective function by setting all elements of $\bar{\mathbf{N}}$ positive and a user-defined upper bound on the total number of routers $(N)$ for the DNN, as shown in Equation (9.5). We solve the optimization problem using the gradient-based interior point algorithm. A sub-gradient based methodology is incorporated in the region where the function is not differentiable. In this

work, we assume that packet transmissions happen only in two consecutive layers. Therefore ensuring local minimum (minimum number of links between two consecutive layers) is sufficient to ensure global minimum latency with a minimum number of links.

Next, we show how the minimum communication latency for the configuration shown in Figure 9.2(a) is achieved. For each round of communication, one packet is communicated between a pair of routers in two consecutive layers. Therefore, $\left\lceil \frac{A_k}{N_k N_{k+1}} \frac{Q}{W} \right\rceil = 1$ in Equation 9.4. For the configuration shown in Figure 9.2(a), $N_1 = 3, N_2 = 2, N_3 = 3$. Putting this in Equation 9.4 we obtain, $L(\bar{\mathbf{N}}) = \max(3,2) + \max(2,3) = 3 + 3 = 6$. Therefore, minimum communication latency for this configuration is 6 cycles, which supports the schedules shown in Figure 9.2(b).

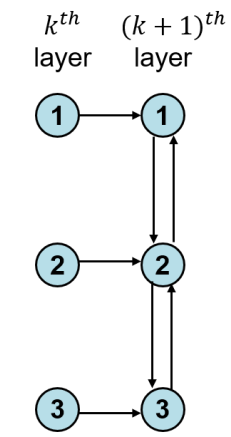
Figure 9.3(b) shows the number of normalized packets between two routers for different DNNs with our proposed tile-to-router mapping methodology. We observe that with the proposed mapping methodology, the number of packets between two routers for different layers of different DNNs is always less than 0.25. We observe an increase in the number of packets between two routers in the case of DenseNet and ResNet-50 due to a decrease in the number of routers with our proposed methodology. The number of routers used for DenseNet with our proposed technique (300) is less than the number of routers required (1088) if one router is allocated per tile. This provides 72% reduction in the NoC area. Similar improvement is seen for ResNet-50 as well.

## Constructing the Custom NoC

In this sub-section, we construct our proposed latency-optimized NoC architecture. The optimal number of routers required for each layer of a given DNN is computed using the methodology described in Section 9.3. We analyze a given DNN layer-by-layer to obtain the latency-optimized NoC architecture and the corresponding schedules. We show, by induction,

that the proposed NoC architecture along with the schedules achieves the minimum communication latency using the minimum number of links for one round of communication. Without loss of generality, let us assume $N_k$ and $N_{k+1}$ are the number of routers in two consecutive layers. We consider three cases: 1) $N_k = N_{k+1}$, 2) $N_k < N_{k+1}$ and 3) $N_k > N_{k+1}$. The minimum possible latency for one round of communication between the two layers is $\max(N_k, N_{k+1})$. For each of these cases, we develop the latency-optimized NoC architecture and the corresponding schedules. We also prove that the constructed NoC architecture achieves the minimum possible latency for all cases.

**Case 1 ($N_k = N_{k+1}$):** We first consider a case where two consecutive layers of a DNN have three routers each, i.e. $N_k = N_{k+1} = 3$. Since each router of $(k+1)^{th}$ layer can receive at most one packet per cycle, the minimum number of cycles required to receive packets from all routers in $k^{th}$ layer is $N_k = \max(N_k, N_{k+1})$. Therefore, the minimum possible latency for one round of communication between each of the three source routers to each of the three destination routers is three cycles. Figure 9.4(a) shows our proposed NoC architecture and Figure 9.4(b) shows the corresponding schedule to finish all transactions in three cycles. We assume that the communication starts at cycle-1. In one round, each router in $k^{th}$ layer sends the same packet (output activation) to all routers in $(k+1)^{th}$ layer. We also assume that when a packet reaches a router, the associated tile computes on the packets, and the transaction is considered to be completed. In the next cycle, the router can send the received packet to other routers if necessary. We denote the packet to be transmitted from $i^{th}$ router of $k^{th}$ layer to $j^{th}$ router of $(k+1)^{th}$ layer as $R_{k,i} - R_{k+1,j}$ as shown in Table 9.1. At cycle-1, all routers in $k^{th}$ layer send the packet to a router in $(k+1)^{th}$ layer through the horizontal links as shown in Figure 9.4(a). First three rows in Figure 9.4(b) show the transaction in cycle-1. In the next cycle, each router in $(k+1)^{th}$ layer transmits the packet received in cycle-1 both




(a)

(b)

| Link | Cycle | Transaction |
|---|---|---|
| $L_{k,1} - L_{(k+1),1}$ | 1 | $R_{k,1} - R_{(k+1),1}$ |
| $L_{k,2} - L_{(k+1),2}$ | 1 | $R_{k,2} - R_{(k+1),2}$ |
| $L_{k,3} - L_{(k+1),3}$ | 1 | $R_{k,3} - R_{(k+1),3}$ |
| $L_{(k+1),1} - L_{(k+1),2}$ | 2 | $R_{k,1} - R_{(k+1),2}$ |
| $L_{(k+1),2} - L_{(k+1),3}$ | 2 | $R_{k,2} - R_{(k+1),3}$ |
| $L_{(k+1),3} - L_{(k+1),2}$ | 2 | $R_{k,3} - R_{(k+1),2}$ |
| $L_{(k+1),2} - L_{(k+1),1}$ | 2 | $R_{k,2} - R_{(k+1),1}$ |
| $L_{(k+1),2} - L_{(k+1),3}$ | 3 | $R_{k,1} - R_{(k+1),3}$ |
| $L_{(k+1),2} - L_{(k+1),1}$ | 3 | $R_{k,3} - R_{(k+1),1}$ |

Figure 9.4: (a) NoC architecture which achieves the minimum possible latency when two consecutive layers each have three routers, (b) Schedule to achieve the minimum possible latency.

through upward and downward vertical link if the links exist. In the subsequent cycles, each router in $(k+1)^{th}$ layer sends the packet received from north to south, and the packet received from south to north through the downward and upward vertical link, respectively. All transactions are finished in three cycles. Since no link is scheduled to transmit more than one packet at a particular cycle, there is no contention in the NoC. We note that if any of the links shown in Figure 9.4(a) is removed, then some of the transactions will not be possible. Therefore, the NoC architecture in Figure 9.4(a) achieves the minimum possible latency using the minimum number of links.

Next, we prove (by induction) that, if the NoC architecture with $N_k = N_{k+1} = N - 1$ achieves minimum latency, then the architecture with $N_k = N_{k+1} = N$ also achieves minimum latency. Figure 9.5 shows the architecture with $N_k = N_{k+1} = N$. The dotted box shows the architecture which is assumed to achieve minimum possible latency, i.e. all transactions

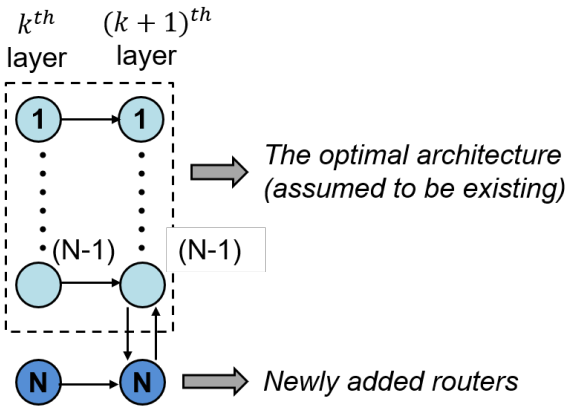



Figure 9.5: NoC architecture which achieves the minimum possible latency when two consecutive layers consist of N routers each.

will finish at $(N-1)$ cycles. The dark blue circles indicate the newly added routers $(N^{th})$ in each layer. By adding the new routers and corresponding links, new transactions are introduced. Our goal is to schedule the new transactions in a way that there is no contention with any transaction scheduled with the architecture having $(N-1)$ routers in each layer. This can be achieved by scheduling the new transaction in the links in the manner shown below.

- Horizontal Link: $(L_{k,N} - L_{(k+1),N})$ carries the packet $R_{k,N} - R_{(k+1),N}$ in cycle-1.
- Upward vertical link: New transactions occur for the packet sent by the router N of the $(k+1)^{th}$ layer. The link $(L_{(k+1),N} - L_{(k+1),(N-1)})$ carries this packet at cycle-2. All other upward vertical links carry this packet after the last transaction through the links with the architecture consisting of $N-1$ routers in each layer. The packet reaches the $1^{st}$ router of the $(k+1)^{th}$ layer at cycle-N.
- Downward vertical link: New transactions occur only through the link $(L_{(k+1),(N-1)} - L_{(k+1),N})$. Since this is a newly added link, the

Table 9.2: Schedules for $N_k = N_{k+1}$

| Link Type | Cycle | Link | Transaction | Range |
|---|---|---|---|---|
| Horizontal | 1 | $L_{k,n}-$ $L_{(k+1),n}$ | $R_{k,n}-$ $R_{(k+1),n}$ | $n = 1 \dots N$ |
| Upward Vertical | m | $L_{(k+1),n}-$ $L_{(k_1),(n-1)}$ | $R_{k,(n+m-2)}-$ $R_{(k+1),(n-1)}$ | $n = 2 \dots N$ |
| Downward Vertical | m | $L_{(k+1),n}-$ $L_{(k+1),(n+1)}$ | $R_{k,(n-m+2)}-$ $R_{(k+1),(n+1)}$ | $n = 1 \dots (N-1)$ |

transaction does not contend with any other link. Through this link, the transaction which occurs last is $R_{k,1} - R_{(k+1),N}$ at cycle-N.

Therefore, all transactions for the architecture with N routers finish at cycle-N, which is the minimum possible latency with N routers in each layer. Table 9.2 shows the schedules for this case.

**Case 2 ($N_k < N_{k+1}, N_k = N$):** Next, we consider the case where $N_k < N_{k+1}$. Without loss of generality, we assume that $N_k = N$. The latency-optimized NoC architecture for this case is shown in Figure 9.9(a).

Table 9.3 shows the schedules for this case. The transactions in the horizontal link and upward vertical link are same as the Case 1, since there is no change in the configuration of these links. The downward vertical links in the $(k+1)^{th}$ layer carries a packet in each cycle till all the transactions are finished.

**Proof for Case 2 ($N_k < N_{k+1}, N_k = N$):** First, we consider the configuration with $N_{k+1} = N + 1$ as shown in Figure 9.6(a). Since each router of $k^{th}$ layer can send at most one packet per cycle, the minimum number of cycles required to send packets to all routers in the $(k+1)^{th}$ layer is $N_{k+1} = \max(N_k, N_{k+1})$. The NoC configuration shown in the dotted box is optimal as it has an equal number of routers in both layers. By adding the router $R_{(k+1),(N+1)}$, we add the downward vertical link $L_{(k+1),N} - L_{(k+1),(N+1)}$. The new transactions due to the newly added router will only happen through this link. Specifically, the router $R_{(k+1),(N+1)}$ will

Table 9.3: Schedules for $N_k < N_{k+1}$

| Link Type | Cycle | Link | Transaction | Range |
|---|---|---|---|---|
| Horizontal | 1 | $L_{k,n}-$ $L_{(k+1),n}$ | $R_{k,n}-$ $R_{(k+1),n}$ | $n = 1 \dots N_k$ |
| Upward Vertical | m | $L_{(k+1),n}-$ $L_{(k+1),(n-1)}$ | $R_{k,(n+m-2)}-$ $R_{(k+1),(n-1)}$ | $n = 2 \dots N_k$ |
| Downward Vertical | m | $L_{(k+1),n}-$ $L_{(k+1),(n+1)}$ | $R_{k,(n-m+2)}-$ $R_{(k+1),(n+1)}$ | $n = 1 \dots N_{k+1} - 1$ |

receive one packet at each cycle starting from cycle-2 only from the router $R_{(k+1),N}$. The last transaction through this link is $R_{(k+1),1} - R_{(k+1),(N+1)}$ and that will happen in cycle-(N+1). Therefore, the NoC configuration shown in Figure 9.6(a) completes all transactions in minimum possible cycles and therefore it is optimal.

Next, we assume that the architecture with $N_{k+1} = N + j$ is optimal as shown in the dotted box in Figure 9.6(b). We will prove by induction that if the architecture with $N_{k+1} = N + j$ is optimal, then the architecture with $N_{k+1} = N + j + 1$ is also optimal which proves the general case. By introducing the router $R_{(k+1),(N+j+1)}$, the downward vertical link $L_{(k+1),(N+j)} - L_{(k+1),(N+j+1)}$ is introduced. The new transactions due to the newly added router will only happen through this link. Specifically, the router $R_{(k+1),(N+j+1)}$ will receive one packet at each cycle starting from cycle-2 only from the router $R_{(k+1),(N+j)}$. The last transaction through this link is $R_{(k+1),1} - R_{(k+1),(N+j+1)}$ and that will happen in cycle-(N+j+1). Therefore, the NoC configuration shown in Figure 9.6(b) completes all transactions in minimum possible cycles and therefore it is optimal.

**Case 3 ($N_k > N_{k+1}, N_{k+1} = N$):** Next, we consider the case where $N_k < N_{k+1}$. Without loss of generality, we assume that $N_{k+1} = N$. The latency-optimized NoC architecture for this case is shown in Figure 9.9(b). Table 9.4 shows the schedules for this case. The transactions in the horizontal links $L_{k,n} - L_{(k+1),n}$ to $L_{k,(N-1)} - L_{(k+1),(N-1)}$ happens in cycle-1.

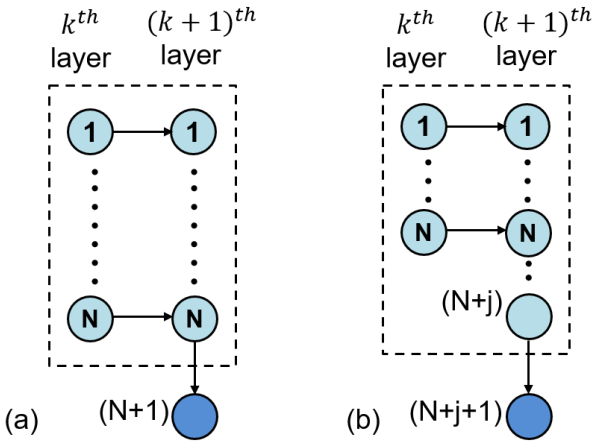



Figure 9.6: NoC architecture to achieve minimum latency for Case 2. (a) shows the case when there is one more router in $(k+1)^{th}$ layer than $k^{th}$ layer. (b) shows the general case. The dotted box shows the optimal architecture (already proved) and the circles filled with dark color represent the newly added router.

Apart from carrying a packet in cycle-1, the link $L_{k,N} - L_{(k+1),N}$ also carries packets from routers $R_{k,(N+j)}$ to $R_{(k+1),N}$ in subsequent cycles, where $j = 1 \ldots (N_k - N)$.

In Appendix C, we show the operation of the proposed NoC architec-

Table 9.4: Schedules for $N_k > N_{k+1}$

| Link Type | Cycle | Link | Transaction | Range |
|---|---|---|---|---|
| Horizontal | 1 | $L_{k,n}-$ $L_{(k+1),n}$ | $R_{k,n}-$ $R_{(k+1),n}$ | $n = 1 \ldots N_{k+1}$ |
| Horizontal | m | $L_{k,N_{k+1}}-$ $L_{(k+1),N_{k+1}}$ | $R_{k,(N_{k+1}+m-1)}-$ $R_{(k+1),N_{k+1}}$ | - |
| Upward Vertical | m | $L_{k,N_{k+1}+n}-$ $L_{k,N_{k+1}+n-1}$ | $R_{k,(N_{k+1}+n+m-1)}-$ $R_{(k+1),N_{k+1}}$ | $n = 1 \ldots$ $(N_k - N_{k+1})$ |
| Upward Vertical | m | $L_{(k+1),n}-$ $L_{(k_1),(n-1)}$ | $R_{k,(n+m-2)}-$ $R_{(k+1),(n-1)}$ | $n = 2 \ldots N_{k+1}$ |
| Downward Vertical | m | $L_{(k+1),n}-$ $L_{(k+1),(n+1)}$ | $R_{k,(n-m+2)}-$ $R_{(k+1),(n+1)}$ | $n = 1 \ldots$ $(N_{k+1} - 1)$ |

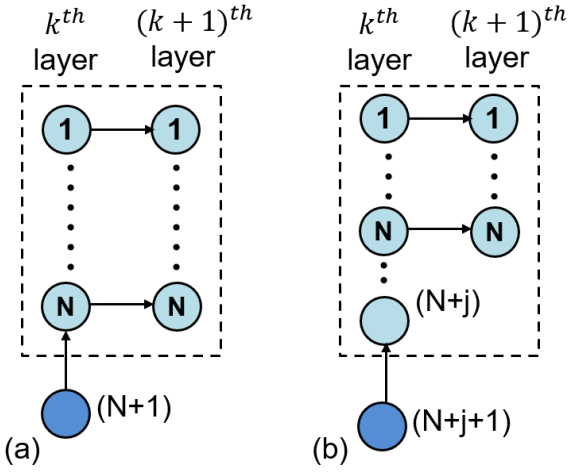



Figure 9.7: NoC architecture to achieve minimum latency for Case 3. (a) shows the case when there is one more router in $(k+1)^{th}$ layer than $k^{th}$ layer, (b) shows the general case. The dotted box shows the optimal architecture (already proved) and the circles filled with dark color represent the newly added router.

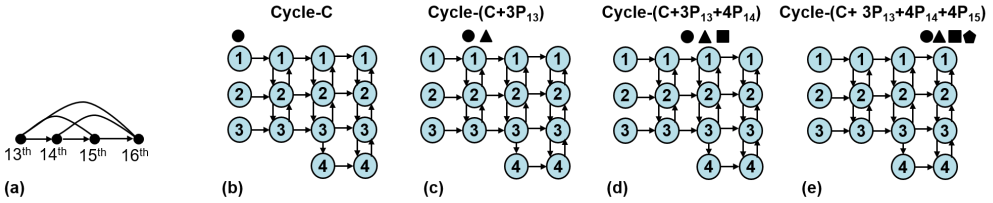



Figure 9.8: Operation of the proposed NoC for a section of DenseNet (100,24) [40]. (a) A representative section of DenseNet (100,24). (b)–(d) show the communication between the layers of DenseNet (100,24).

ture for densely connected DNNs such as DenseNet [40].

**Proof for Case 3** ($N_k > N_{k+1}, N_{k+1} = N$)**:** First, we consider the configuration with $N_{k+1} = N + 1$ as shown in Figure 9.7(a). Since each router of $(k+1)^{th}$ layer can receive at most one packet per cycle, the minimum number of cycles required to receive packets from all routers in $k^{th}$ layer is $N_k = \max(N_k, N_{k+1})$. The NoC configuration shown in the dotted box is

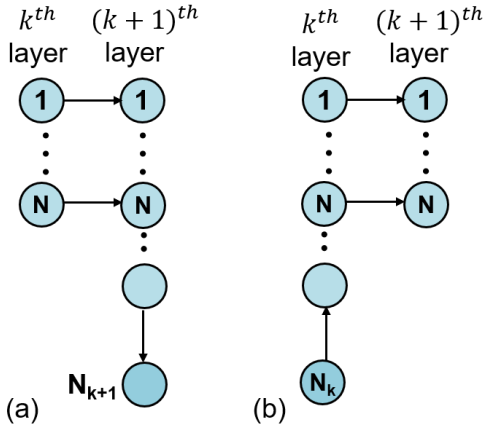



Figure 9.9: NoC architecture which achieves minimum possible latency when (a) $N_k < N_{k+1}$ and (b) $N_k > N_{k+1}$.

optimal as it has an equal number of routers in both layers. By adding the router $R_{k,(N+1)}$, we add the upward vertical link $L_{k,N} - L_{k,(N+1)}$. Specifically, the router $R_{k,(N+1)}$ will send one packet at cycle-1 to the router $R_{k,N}$. This packet will be sent to $R_{(k+1),N}$ through the link $L_{k,N} - L_{(k+1),N}$ at cycle-2. We note that this link is free at cycle-2 with the configuration shown in the dotted box. Subsequently, this packet will reach the router $R_{(k+1),1}$ at cycle-(N+1) which is the last transaction with this NoC configuration. Therefore, the NoC configuration shown in Figure 9.7(a) completes all transactions in minimum possible cycles and therefore it is optimal.

Next, we assume that the architecture with $N_k = N + j$ is optimal as shown in the dotted box in Figure 9.7(b). We will prove by induction that if the architecture with $N_k = N + j$ is optimal then the architecture with $N_k = N+j+1$ is also optimal which proves the general case. By introducing the router $R_{k+,(N+j+1)}$, the upward vertical link $L_{k,(N+j)} - L_{k,(N+j+1)}$ is introduced. Specifically, the router $R_{k,(N+j+1)}$ will send one packet at cycle-1 to the router $R_{k,(N+j)}$. This packet will reach the router $R_{k,N}$ at cycle-(j+1). In cycle-j it will be sent to the router $R_{(k+1),N}$ and in the

subsequent cycle the packet will traverse through the upward vertical link till it reaches the router $R_{(k+1),1}$. The last transaction that will happen in cycle-(N+j+1). Therefore, the NoC configuration shown in Figure 9.7(b) completes all transactions in minimum possible cycles and therefore it is optimal.

**Execution of the proposed network on dense neural network:** Figure 9.8 shows the operation of the proposed NoC on DenseNet [40]. We consider the $13^{th}$–$16^{th}$ layer of DenseNet which is a representative section of this DNN. In DenseNet, all neurons in a particular layer are connected with all other neurons in the subsequent layers as shown in Figure 9.8(a). The number of routers allocated with our proposed methodology for the $13^{th}$, $14^{th}$, $15^{th}$ and $16^{th}$ layers are 3,3,4,4 respectively which is shown in Figure 9.8(b). Different packets generated in different layers are shown in different markers in Figure 9.8(b)–9.8(e). We denote the number of packets to be communicated for each source to destination pair from $k^{th}$ layer to $(k+1)^{th}$ layer as $P_k$. We assume that at cycle-C the packets (shown in ●) in the $13^{th}$ layer is ready to be communicated to the $14^{th}$ layer. Specifically, at each round of communication, each router in $13^{th}$ layer will send one packet (marked with ●) to each router in $14^{th}$ layer. According to our proposed approach, each round of communication between $13^{th}$ and $14^{th}$ layer will take 3 cycles ($\max(N_{13}, N_{14}) = \max(3, 3) = 3$). Therefore, one round of communication will be finished at cycle-(C+3$P_{13}$). After that, the computations are performed in the $14^{th}$ layer. Without loss of generality, we assume that computations are performed in the same cycle the packets (activations) reach the layer. After computations, new packets are generated which are to be communicated to the next layer. The new packets are denoted by the marker ▲. Each type of the packets marked with ● and ▲ are to be communicated to $15^{th}$ layer. Each round of communication takes 4 cycles ($\max(N_{14}, N_{15}) = \max(3, 4) = 4$). Therefore, all the packets reach $15^{th}$ layer at cycle-(C+3$P_{13}$+4$P_{14}$). After that, the computations are

**Algorithm 4:** Proposed Algorithm to Reconfigure the NoC at Runtime

**Input:** Number of layers of the DNN ($K$), number of input activations for each layer ($A_k$), precision bit ($Q$), NoC bus width ($W$), Number of routers available on-chip for each layer ($N_k^{max}$)
**Output:** Number of routers required for each layer ($N_k$) and the optimal schedule ($sched_{out}$)
**Initialization:** $sched_{out} \leftarrow [\,]$
Obtain $N_k$ following Equation 9.4 and Equation 9.5
**for** *k = 1: K* **do**
    $N_k = \min(N_k, N_k^{max})$
**end**
**for** *k = 1: K-1* **do**
    `/* Number of packets */`
    $P_k = \left\lceil \frac{A_k}{N_k N_{k+1}} \frac{Q}{W} \right\rceil$
    `/* Constructing the schedules */`
    **for** $p = 1 : P_k$ **do**
        **if** $N_k == N_{k+1}$ **then**
            $sched_p$ = schedule constructed by following Table 9.2
            $sched_{out} \leftarrow [sched_{out}; sched_p]$
        **end**
        **if** $N_k < N_{k+1}$ **then**
            $sched_p$ = schedule constructed by following Table 9.3
            $sched_{out} \leftarrow [sched_{out}; sched_p]$
        **end**
        **if** $N_k > N_{k+1}$ **then**
            $sched_p$ = schedule constructed by following Table 9.4
            $sched_{out} \leftarrow [sched_{out}; sched_p]$
        **end**
    **end**
**end**

performed in the $15^{th}$ layer and the new packets (shown in ■) are generated. Similarly, the packets will reach the $16^{th}$ layer at cycle-$(C+3P_{13}+4P_{14}+4P_{15})$ and upon computation, new packets will be generated (shown in pentagon filled in black). Thus the proposed NoC architecture works seamlessly for densely connected networks.

## Constructing a Reconfigurable NoC

So far, we have discussed our proposed methodology to construct a latency-optimized NoC customized for a given DNN. However, an NoC architecture customized for a specific DNN is not practical due to the lack of reconfigurability. Since DNNs are ever-evolving, we can never guarantee that the set of DNNs considered at design time is exhaustive. Therefore, at run-time the NoC might need to execute a DNN which was not considered at design time. To overcome this challenge, we propose a technique to construct two reconfigurable NoCs for two categories of DNNs, namely, edge-based and cloud-based DNNs. There are two steps involved in constructing the reconfigurable architecture. *First*, we set the number of layers to be supported by the NoC architecture. *Second*, we set the number of routers per layer for the NoC architecture.

**Setting number of layers**: For each category, we set the number of layers to be the maximum number of layers among all DNNs available at design time in that particular category. Specifically, if $D^{(i)}$ is the number of layers of a DNN $i$ and the number of DNNs considered in that category is I, then the number of layers the NoC architecture can accommodate is $D = \max(D^{(1)}, D^{(2)}, \ldots, D^{(I)})$. For the DNNs we consider in the edge computing category, SqueezeNet has the maximum number of layers $(D = 26)$. Similarly, DNNs we consider in the cloud computing category, ResNet-152 has the maximum number of layers $(D = 152)$.

**Setting number of routers per layer:** Next, we compute the number of routers to be allocated for each layer. To this end, for each DNN of that category, we evaluate the optimal number of routers required for each layer following the methodology described in Section 9.3. Then, for that particular layer, we allocate the maximum number of routers obtained across all DNNs of the category. If $N_k^{(i)}$ is the number of routers required for the $k^{th}$ layer of a DNN $i$, then the number of routers allocated in the NoC architecture for $k^{th}$ layer is $N_k^{max} = \max(N_k^{(1)}, N_k^{(2)}, \ldots N_k^{(I)})$, where

$k = 1, 2, \ldots D$ and $N_k^{(i)} = 0$ if $D^{(i)} < k$.

At runtime, we reconfigure the NoC to determine how many routers need to be used and determine the schedules of the packets between each pair of layers of the new DNN. Algorithm 4 shows the proposed algorithm which is executed on-chip to reconfigure the proposed NoC architecture. The algorithm takes different DNN parameters (number of layers, number of input activations, precision bit), NoC bus width, and number of routers available on-chip for each layer as input. The number of routers required for each layer and the optimal schedules are obtained as outputs.

The number of routers required for each layer ($N_k$) of DNN is determined by following the procedure described in Section 9.3 (shown in line 4 of Algorithm 4). If the required number of routers ($N_k$) are not available on-chip, then the maximum available routers ($N_k^{max}$) are utilized for that particular layer of the DNN (shown in line 5–7 of Algorithm 4). After that, we compute the schedules between two consecutive layers of the DNNs. First, we compute the number of packets between each source to each destination (line 9) and construct the schedules for the packets. In order to construct the schedules, we consider the three cases described in Section 26. Depending on the case, the schedules are constructed following Table 9.2 or Table 9.3 or Table 9.4. Second, the constructed schedule is then appended to the list of optimal layer-wise schedules ($sched_{opt}$) (shown in line 10–20 of Algorithm 4). The same procedure is repeated for all layers to obtain the complete schedule of the reconfigurable NoC.

## Router Architecture of the Proposed NoC

Figure 9.10 shows the router architecture of the proposed NoC. The router has three input ports: one input port ($I_P$) connects with a router in the previous layer and the other two input ports are connected with routers of the same layer. We assume that all the packets to be sent in the next layer are stored in a buffer inside the compute elements in the previous layer.

The input port $I_N$ gets the input from the router situated to the north and the input port $I_S$ gets the input from the router situated to the south. The router has two output ports: $O_N$ sends the output to the router situated to the north and $O_S$ sends the output to the router situated to the south. Inside the router, there are two multiplexers: $M_A$ and $M_B$. $M_A$ selects between the inputs coming from $I_P$ and $I_S$ and sends it to $O_N$. $M_B$ selects between the inputs coming from $I_P$ and $I_N$ and sends it to $O_S$. As discussed in Section 26, at cycle-2 of each round of the communication, the input from the previous layer is sent to both the routers situated to the north and south. In all other cycles, the input from the south is sent to the north and vice-versa. Therefore, $M_A$ and $M_B$ are controlled by the cycle-index (C) in each round of communication as shown in Figure 9.10. The router is interfaced with the tiles in the current layer. Upon receiving a packet, the router sends it to the corresponding computing tile. The tile computes on the packet and sends it back to the router, which then forwards it to the next router.

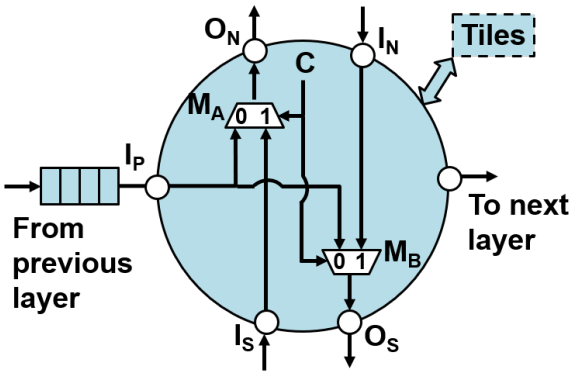



Figure 9.10: Router architecture of the proposed NoC

## 9.4 Experimental Evaluation

### Experimental setup

We evaluate our proposed latency-optimized NoC for IMC architecture for a wide range of DNNs. We consider LeNet [62] on MNIST dataset, NiN [67], ResNet-152 [36], VGG-16, VGG-19 [102], and DenseNet(100,24) [40] on CIFAR-100 dataset [58], and SqueezeNet [41] and ResNet-50 [36] on ImageNet dataset [25]. The DNNs we consider have parameters that range from 0.28M for LeNet to 45M for VGG-19 and the number of layers ranges from 5 for LeNet to 152 for ResNet-152. Moreover, the DNNs we chose have different connection patterns; linear (LeNet, NiN, SqueezeNet, VGG), residual (ResNet) and dense (DenseNet). These DNNs are a combination of fully connected (FC) and convolutional (Conv) layers. Therefore, our proposed methodology is applicable to fully connected layers as well as convolutional layers of a DNN.

**Benchmarking Simulator:** We developed an in-house simulator to evaluate the IMC architecture with the proposed latency-optimized NoC for different DNNs. The circuit part and interconnect part of the simulator are calibrated with NeuroSim [18] and BookSim [45], respectively. The inputs of the simulator include the DNN structure, technology node, NoC bus-width, type of IMC technology (ReRAM, SRAM, etc.), the number of bits per IMC cell, and frequency of operation. The circuit simulator performs the mapping of the entire DNN to a multi-tiled IMC architecture [100] and reports performance metrics, such as area, energy, and latency of the computing logic. The interconnect performance is evaluated using the interconnect simulator. The circuit simulator provides the number of tiles per layer, activations, and number of layers as output. These are used to construct the latency-optimized NoC, which are then fed to the interconnect simulator to compute the area, energy, and latency for the interconnect. The overall performance of the architecture is calculated

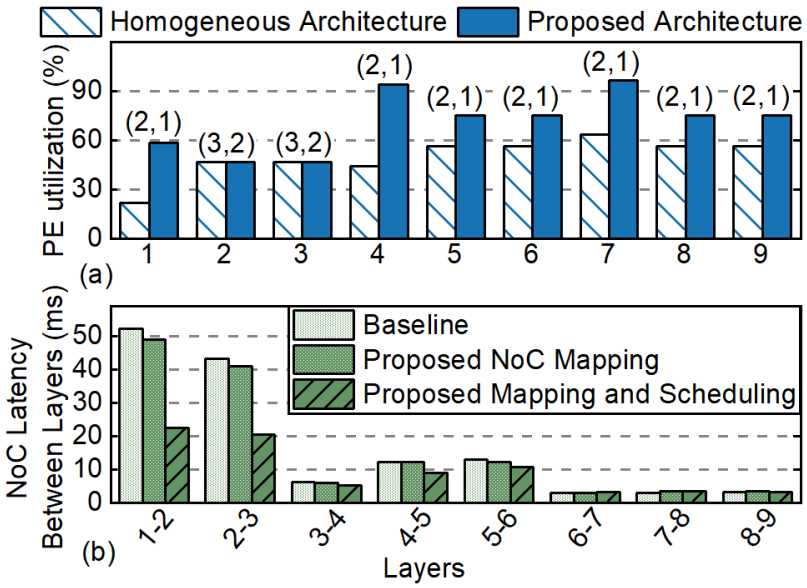



Figure 9.11: Layer-wise improvement for NiN in (a) PE utilization for each layer with SRAM-based heterogeneous tile architecture. The tile structure for each layer $(c_k, p_k)$ is shown on top of each bar and (b) communication latency for each layer with proposed NoC optimization.

by combining the circuit-level and interconnect-level performance. The details of the simulator is described in [56].

## Energy-Aware NoC Optimization

The proposed methodology includes an energy-aware tile-to-router mapping and scheduling technique for the NoC. The upper bound on the number of routers is set as three times the number of DNN layers to balance energy and performance. Figure 9.11(b) shows the improvement in latency for each layer of NiN due to the proposed NoC optimization. The proposed NoC mapping reduces the communication latency between layers 1 and 2 from 51ms to 47ms. As we integrate the NoC mapping with the scheduling technique, latency reduces further to 22ms. The first three

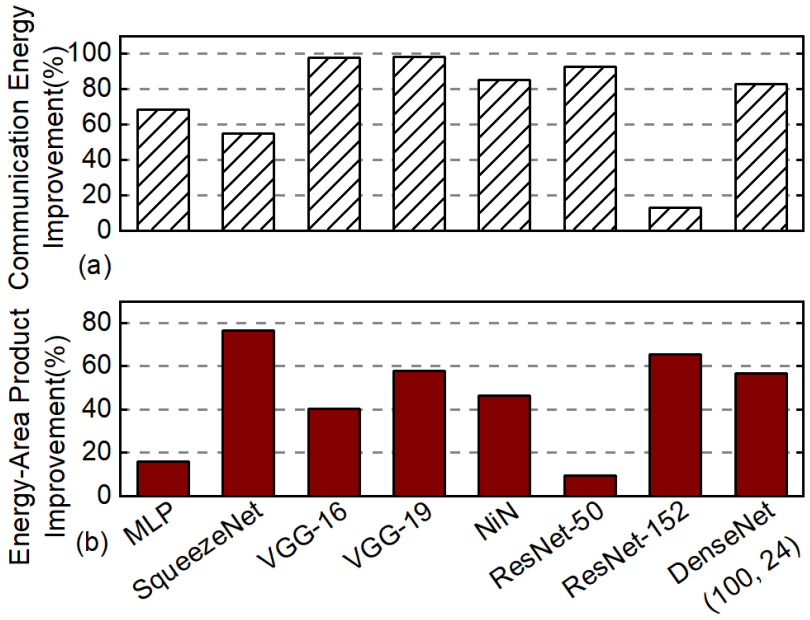



Figure 9.12: Improvement in (a) communication energy of the proposed energy-aware NoC optimization with respect to the baseline (SRAM) and (b) energy-area product of the generated SRAM-based architecture with respect to the baseline (SRAM).

layers of NiN contain more than 50% of the total number of activations. Therefore, the proposed NoC mapping reserves more routers for the first three layers, resulting in a significant reduction in latency for those layers. Additionally, the total number of routers is reduced which reduces the NoC area. A direct consequence of both latency and area reduction is lower communication energy, as shown in Figure 9.12(a) with an average reduction of 74%. The energy reduction is the highest for the case of VGG networks – 97%/98% for VGG-16/VGG-19. For ResNet-152, energy reduction is the lowest (15%), since the tiles are well distributed across layers for the baseline architecture, leaving less room for improvement.

**Baseline Architecture:** We utilize a crossbar-based multi-tiled IMC architecture to evaluate our proposed approach. Analog MAC computation is

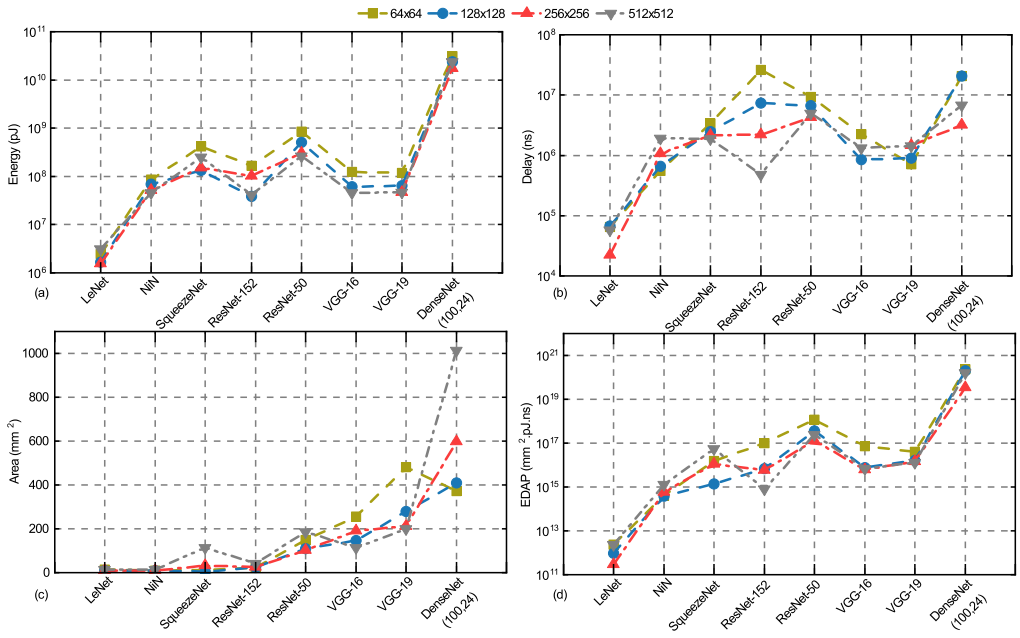



Figure 9.13: Performance of the baseline SRAM-based IMC architecture for different DNNs with different crossbar size. We observe that a crossbar size of 128×128 or 256×256 performs better (with mesh-NoC) than other crossbar sizes.

performed along the bitline, the analog voltage/current is digitized with a 4-bit flash ADC, a sample and hold circuit, and a shift and add circuit (read-out circuit) at the column periphery. 8 columns are multiplexed together to one read-out circuit to reduce chip area. Sequential input signalling is employed to do away with the DAC. Each tile consists of 4 compute elements (CEs) and each CE consists of 4 processing elements (PEs) or crossbar arrays [18]. We consider 32nm technology node [100], 1GHz frequency of operation, and a parallel read-out for the crossbar [105]. A mesh-based NoC with bus width of 32 bits and one router-per-tile is considered for the interconnect for the baseline architecture.

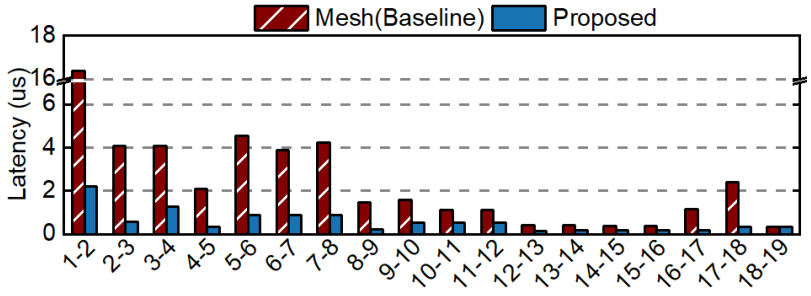



Figure 9.14: Improvement in communication latency for each layer of VGG-19.

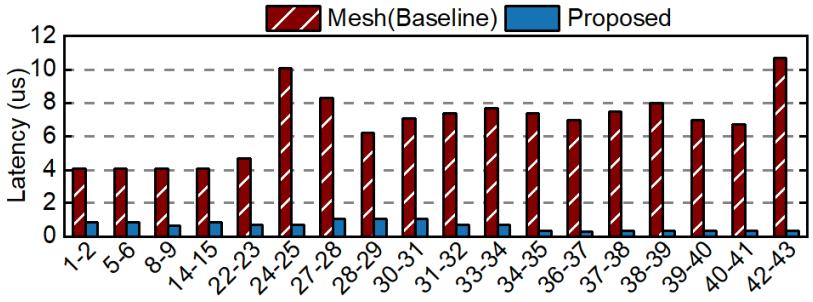



Figure 9.15: Improvement in communication latency for each layer of ResNet-50.

## Optimal Size of Crossbar Array

In this section, we show the area, energy, and latency comparison of different DNNs with the baseline IMC architecture having different sized crossbars. The performance of IMC architecture for different DNNs vary with number of layers, connection density (number of connections per neuron) and number of parameters of the DNN. We consider SRAM-based bitcell/array design [18]. Figure 9.13 shows the comparison of energy, delay, area, and energy-delay-area product (EDAP) of crossbar size varying from 64×64 to 512×512. In each of the subfigures, the DNNs are presented in increasing order of area (LeNet has the lowest area and is at the left end while DenseNet(100,24) has the highest area and is at

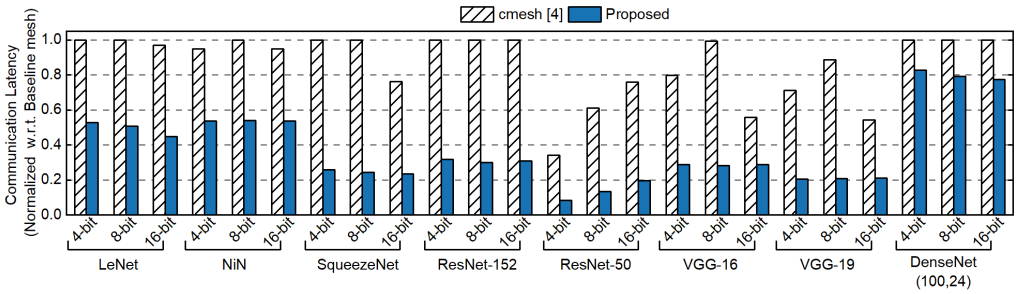



Figure 9.16: Improvement in communication latency for different DNNs (with crossbar size of 256×256) and weights and activation precision with respect to mesh-NoC for cmesh [100] and the proposed approach.

the right end). Figure 9.13(a) shows the energy consumption for different DNNs with different crossbar sizes. We observe that a crossbar size of 256×256 has the lowest energy consumption for 5 out of the 8 DNNs we evaluated. The IMC architecture with a crossbar size of 64×64 has poor performance in terms of energy consumption, inference latency, and area as shown in Figure 9.13(a), 9.13(b), and 9.13(c) respectively. Figure 9.13(d) shows the energy-delay-area product (EDAP) of different DNNs with different crossbar sizes. We observe that the architecture with a crossbar size of 128×128 and 256×256 has better performance in terms of EDAP for almost all DNNs. We observe a similar trend for IMC architecture with ReRAM-based crossbar arrays. Since larger crossbar size results in the chip being more compact, moving forward, we choose a crossbar size of 256×256 for all experiments. We also show some representative results with a crossbar size of 128×128.

## Layer-wise Comparison of Communication Latency

In this section, we show the improvement in communication latency with the proposed latency-optimized NoC (with an IMC architecture of 8-bit precision) for different layers of VGG-19 and ResNet-50. Figure 9.14 com-

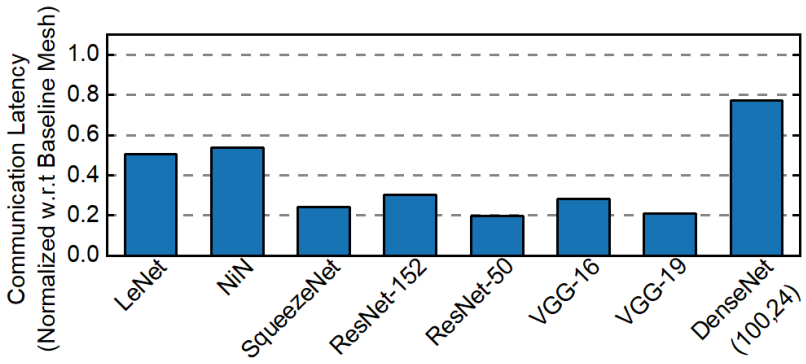



Figure 9.17: Improvement in communication latency with proposed NoC with respect to mesh for crossbar size of 128×128.

pares the communication latency of VGG-19 between the IMC architectures with the proposed NoC and the baseline mesh-NoC. We observe that the improvement in communication latency for the first 4 layers of VGG-19 is significant. Improvement in communication latency is highest between the first two layers of VGG-19, which is 86%.

We also observe significant improvement in communication latency for different layers of ResNet-50. Figure 9.15 shows the improvement for a few representative layers of ResNet-50 (we limit it for better visibility). The maximum improvement is seen between layer 42 and layer 43, which is 96%. The improvement in communication latency for each layer contributes to the improvement in total communication latency as shown in the next section. Such high improvement stems from the proposed latency-optimized NoC and the efficient distribution of routers among layers as discussed in Section 9.3.

## Overall Improvement in Communication Performance

Next, we evaluate the proposed latency-optimized NoC-based IMC architecture for different DNNs. We compare the total communication latency of the proposed NoC with the cmesh interconnect proposed in [100].

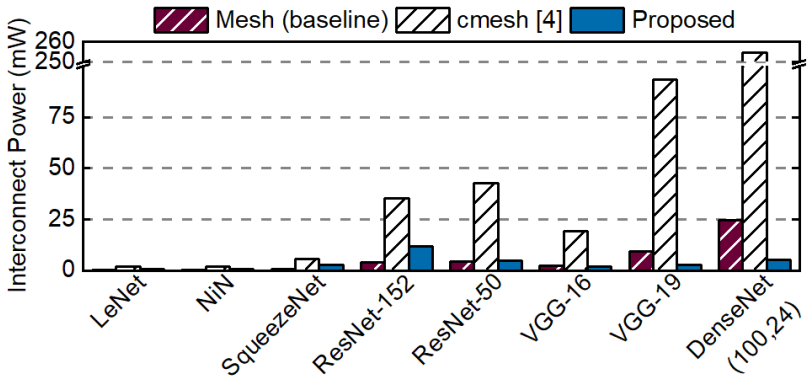



Figure 9.18: Comparison of interconnect power consumption with different techniques.

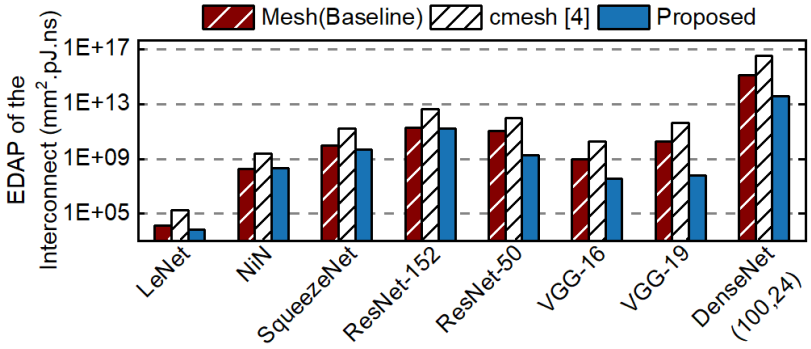



Figure 9.19: Interconnect EDAP comparison for different DNNs.

Figure 9.16 shows the comparison of communication latency for three different data precisions (weights and activations): 4-bit, 8-bit, and 16-bit. In this case, we consider an IMC architecture with a crossbar size of 256×256. The communication latency values are normalized with respect to the communication latency values obtained with the baseline NoC. We observe that the communication latency for the cmesh interconnect is similar to that of the baseline NoC for LeNet, NiN, DenseNet (100,24), and ResNet-152. The cmesh-based IMC architecture performs significantly better for VGG-16, VGG-19, and ResNet-50. Specifically, for ResNet-50, on average,

the cmesh interconnect achieves 57% improvement in communication latency with respect to the baseline mesh-NoC.

Our proposed latency-optimized NoC reduces the communication latency significantly both with respect to baseline NoC and cmesh-based NoC as shown in Figure 9.16. With respect to cmesh-based NoC, there is a 20%-80% improvement for different DNNs with different bit precisions. Highest improvement with respect to the baseline NoC is observed for ResNet-50 with a 4-bit data precision. On average, the proposed latency optimized NoC improves the communication latency by 62% with respect to mesh-NoC, and 57% with respect to cmesh interconnect. Since our proposed NoC architecture achieves minimum latency, there is a significant improvement in communication latency with respect to state-of-the-art works.

In Figure 9.17, we show the improvement in communication latency with the proposed NoC with respect to mesh-NoC for an IMC architecture with a crossbar size of 128×128 and 8-bit precision. We observe that the improvement in communication latency follows a similar trend as the crossbar size of 256×256. Therefore, the improvement due to the proposed latency-optimized NoC is independent of crossbar size.

Figure 9.18 presents the comparison of interconnect power consumption for an IMC architecture with 8-bit precision for both weights and activations. Our proposed latency-optimized NoC achieves up to 4.6× improvement in interconnect power consumption with respect to baseline mesh-NoC. The power consumption with the proposed interconnect for ResNet-152 is higher than the baseline mesh-NoC due to the use of higher number of routers. However, this results in 3.32× improvement in interconnect latency with the proposed interconnect. We achieve 2.26×–47× improvement in power consumption as compared to the cmesh interconnect. The improvement is highest for DenseNet and least for SqueezeNet.

To further understand the efficacy of the proposed NoC, we compare

the energy-delay-area product (EDAP) with respect to the baseline mesh-NoC and cmesh interconnect for different DNNs. Figure 9.19 shows the comparison for an IMC architecture with 8-bit precision for both weights and activations. Our proposed latency-optimized NoC achieves up to 328× improvement in the EDAP of the interconnect with respect to baseline mesh-NoC. We achieve EDAP improvement for the interconnect in the range of 12×–6600× as compared to the cmesh interconnect. The improvement is highest for VGG-19 and least for NiN. Since cmesh interconnect uses additional number of routers and links to reduce latency, it results in higher area and energy. Therefore, the performance of cmesh interconnect is worse than mesh-NoC in terms of EDAP. The proposed latency-optimized NoC provides a large improvement in communication latency which results in reduced energy with reduced or comparable area. Therefore, our proposed latency-optimized NoC architecture performs significantly better in terms of area, energy and latency than both cmesh interconnect and the baseline NoC.

## Overall Improvement

In this section, we discuss the overall improvement in inference performance for an SRAM-based IMC architecture with our proposed latency-optimized NoC architecture. Figure 9.20(a) shows the improvement in total inference latency with respect to the baseline architecture with mesh-NoC. The improvement is in the range of 5%-25% for different DNNs. We observe higher improvement for SqueezeNet and ResNet-152. These two DNNs have a higher number of activations between layers compared to other DNNs. Higher number of activation leads to higher communication volume, which in turn results in more congestion for mesh-NoC. In contrast, our proposed latency-optimized NoC schedules the packets in such a way that there is no congestion in the NoC leading to significant improvement over mesh-NoC. The efficiency of IMC architecture with

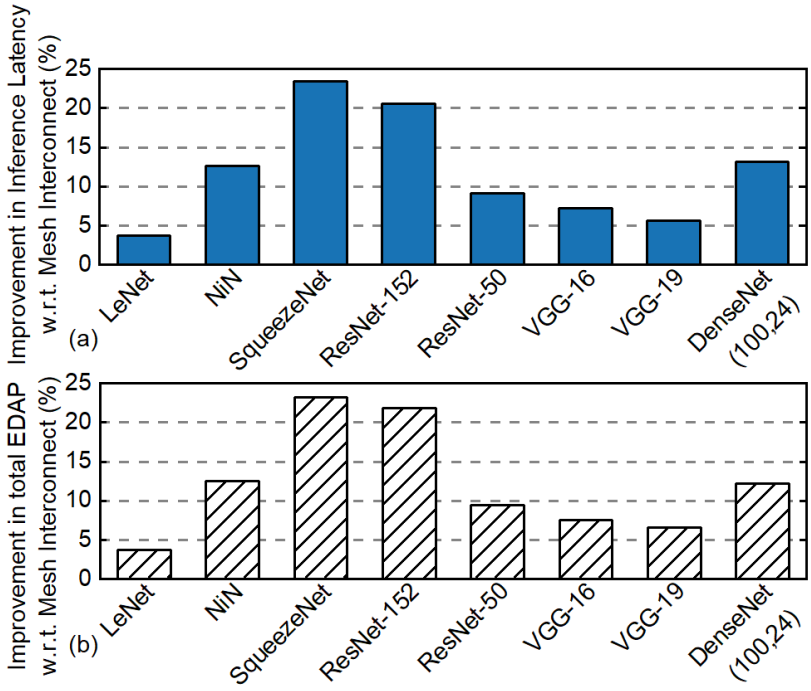



Figure 9.20: Overall improvement in (a) total inference latency and (b) total EDAP of a SRAM-based IMC architecture with the proposed latency-optimized interconnect with respect to the baseline.

mesh-NoC over [105, 97] is shown in [57]. Moreover, we observe that our proposed NoC architecture results in 13%-85% improvement in inference latency with respect to the IMC architecture with bus-based H-Tree interconnect [105, 79].

Figure 9.20(b) shows the improvement in total system EDAP for an SRAM-based IMC architecture with proposed latency-optimized NoC for all DNNs. Since improvement in inference latency is higher compared to the improvement in energy and area, we observe that the improvement in EDAP follows a similar trend as that of inference latency. On average, the proposed latency optimized NoC delivers 9.8% improvement in overall EDAP with respect to baseline architecture with mesh-NoC. Since intercon-

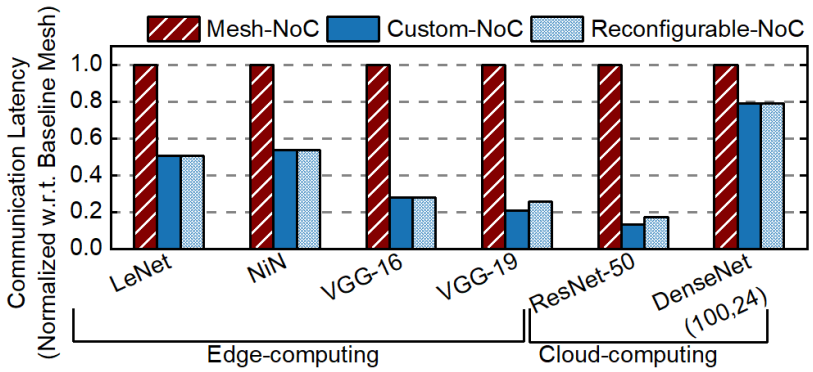



Figure 9.21: Results of leave-one-out experiments with reconfigurable NoC for edge computing- and cloud computing-based DNNs.

nect plays an important role in overall performance of an IMC architecture, the proposed NoC architecture contributes to a considerable improvement in overall inference performance for different DNNs.

## Results with the Reconfigurable NoC

In this section, we show the results of our proposed reconfigurable NoC. We identify two broad class of DNNs, namely, edge-based and cloud-based DNNs. We categorize the DNNs based on its application on edge-computing or cloud-computing based devices. We consider LeNet, Squeeze Net, NiN, VGG-16 and VGG-19 in the category of edge-based DNNs and ResNet-50, ResNet-152 and DenseNet (100, 24) in the category of cloud-based DNNs. We assume that the circuit part of the IMC architecture is reconfigurable and supports the specific class of DNNs under consideration. We perform leave-one-out experiment to evaluate our proposed reconfigurable NoC. For example, while performing experiment for VGG-19 (edge computing-based DNN), we assume that information of VGG-19 is not available at design time.

The number of layers for a reconfigurable NoC for edge-computing is

set at 26. For each layer, we set the number of routers as the maximum number of routers required for all DNNs for that particular layer. For example, for 1$^{st}$ layer, the optimal number of routers required for LeNet, NiN, SqueezeNet, and VGG-16 are 4, 2, 5, and 4, respectively. Therefore, we allocate 5 routers for 1$^{st}$ layer.

At runtime, on encountering the new DNN (VGG-19), we execute Algorithm 4 to generate the NoC schedules and execute VGG-19 with available resources on-chip. For fairness, we perform the same experiment with multiple DNNs as shown in Figure 9.21. We observe that there is $<$5% degradation in communication latency for VGG-19 and ResNet-50, while other DNNs have the same performance as that of the custom NoC. Since the optimal number of routers required for a few layers of the DNN may not be present on-chip, there might be a degradation in communication latency with respect to custom NoC. For example, for the experiment with VGG-19, 5 routers are allocated for the 1$^{st}$ layer. However, the custom NoC optimized for VGG-19 requires 6 routers. Since it can use up to 5 routers for the first layer, the communication latency of VGG-19 with reconfigurable NoC is more than the custom NoC. Still, for VGG-19 and ResNet-50, the proposed reconfigurable NoC performs significantly better than the baseline mesh-NoC as shown in Figure 9.21. We also observe that the runtime overhead of the proposed algorithm ranges from 0.049s (LeNet) to 49.93s (DenseNet). However, the overhead is negligible considering that the reconfiguration is a one-time effort for each DNN. Therefore, the proposed algorithm reconfigures the available NoC resources depending on the DNN being executed and provides significant benefit with respect to the baseline mesh-NoC.

## 9.5 Conclusion

In this work, we present a latency-optimized reconfigurable NoC for in-memory acceleration of DNNs. State-of-the-art interconnect methodologies include bus-based H-Tree interconnect and mesh-NoC. We show that bus-based H-Tree interconnect contributes significantly to the total inference latency of DNN hardware and are not a viable option. Mesh-NoC based IMC architectures are better than bus-based H-tree but they too do not consider the non-uniform weight distribution of different DNNs, DNN graph structure, and the computation-to-communication imbalance of the DNNs.None of the architectures holistically investigated minimization of communication latency. In contrast, our proposed latency-optimized NoC guarantees minimum possible communication latency between two consecutive layers of a given DNN. Furthermore, we proposed reconfigurable NoC for two representative categories of DNNs, namely, edge computing-based and cloud computing-based DNNs. Experimental evaluations on a wide range of DNNs confirm that the proposed NoC architecture enables 60%-80% reduction in communication latency with respect to state-of-the-art interconnect solutions.

# BIBLIOGRAPHY


[1] Graph nets library. `https://deepmind.com/research/open-source/graph-nets-library`.

[2] N. Agarwal et al. GARNET: A Detailed on-chip Network Model Inside a Full-system Simulator. In *2009 IEEE intl. symp. on performance analysis of systems and software*, pages 33–42.

[3] M. Arafa and thers. Cascade Lake: Next Generation Intel Xeon Scalable Processor. *IEEE Micro*, 39(2):29–36, 2019.

[4] I. Awan and R. Fretwell. Analysis of Discrete-Time Queues with Space and Service Priorities for Arbitrary Arrival Processes. In *Parallel and Distributed Systems. Proc. 11th Intl Conf. on*, volume 2, pages 115–119, 2005.

[5] B. A. P. C. (BAPCo). Benchmark, sysmark2014. `http://bapco.com/products/sysmark-2014`, accessed 27 May 2020.

[6] A. Bartolini et al. A Virtual Platform Environment For Exploring Power, Thermal And Reliability Management Control Strategies In High-Performance Multicores. In *Proc. of the Great lakes Symp. on VLSI*, pages 311–316, 2010.

[7] A. W. Berger and W. Whitt. Workload Bounds in Fluid Models with Priorities. *Performance evaluation*, 41(4):249–267, 2000.

[8] D. P. Bertsekas, R. G. Gallager, and P. Humblet. *Data Networks*, volume 2. Prentice-Hall International New Jersey, 1992.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of the Intl. Conf. on Parallel Arch. and Compilation Tech.*, pages 72–81, 2008.

[10] N. Binkert et al. The Gem5 Simulator. *SIGARCH Comp. Arch. News*, May. 2011.

[11] P. Bogdan and R. Marculescu. Workload Characterization and Its Impact on Multicore Platform Design. In *Proc. of the Intl. Conf. on Hardware/Software Codesign and System Synthesis*, pages 231–240, 2010.

[12] P. Bogdan and R. Marculescu. Non-stationary Traffic Analysis and its Implications on Multicore Platform Design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):508–519, 2011.

[13] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 2006.

[14] A. Borodin, Y. Rabani, and B. Schieber. Deterministic Many-to-Many Hot Potato Routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):587–596, 1997.

[15] J. T. Brassil and R. L. Cruz. Bounds on Maximum Delay in Networks with Deflection Routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(7):724–732, 1995.

[16] J. Bucek, K.-D. Lange, and J. v. Kistowski. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 41–42, 2018.

[17] J. Chen and X. Ran. Deep Learning with Edge Computing: A Review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[18] P.-Y. Chen, X. Peng, and S. Yu. NeuroSim: A Circuit-level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(12):3067–3080, 2018.

[19] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[20] M.-C. Chiang, T.-C. Yeh, and G.-F. Tseng. A QEMU and SystemC-based Cycle-accurate ISS for Performance Estimation on SoC Development. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):593–606, 2011.

[21] W. Choi et al. On-Chip Communication Network for Efficient Training of Deep Convolutional Networks on Heterogeneous Manycore Systems. *IEEE Trans. on Computers*, 67(5):672–686, 2017.

[22] I.-H. Chung, C. Kim, H.-F. Wen, and G. Cong. Application data prefetching on the ibm blue gene/q supercomputer. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–8. IEEE, 2012.

[23] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song. Learning steady-states of iterative algorithms over graphs. In *International conference on machine learning*, pages 1106–1114. PMLR, 2018.

[24] A. C. de Melo. The New Linux Perf Tools. In *Linux Kongress*, volume 18, 2010.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A Large-scale Hierarchical Image Database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

[26] J. Doweck et al. Inside 6th-generation Intel Core: New Microarchitecture Code-named Skylake. *IEEE Micro*, (2):52–62, 2017.

[27] D. Duvenaud et al. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*, 2015.

[28] C. Fallin, C. Craik, and O. Mutlu. CHIPPER: A Low-Complexity Bufferless Deflection Router. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 144–155, 2011.

[29] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu. MinBD: Minimally-buffered Deflection Routing for Energy-efficient Interconnect. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 1–10, 2012.

[30] Z. Fang, D. Hong, and R. K. Gupta. Serving Deep Neural Networks at the Cloud Edge for Vision Applications on Mobile Platforms. In *Proceedings of the 10th ACM Multimedia Systems Conference*, pages 36–47, 2019.

[31] A. Ghosh and T. Givargis. Analytical Design Space Exploration of Caches for Embedded Systems. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 650–655, 2003.

[32] P. Ghosh, A. Ravi, and A. Sen. An Analytical Framework with Bounded Deflection Adaptive Routing for Networks-on-Chip. In *2010 IEEE Computer Society Annual Symposium on VLSI*, pages 363–368, 2010.

[33] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98, 1998.

[34] A. Gotmanov et al. Verifying Deadlock-Freedom of Communication Fabrics. In *Intl. Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 214–231. Springer, 2011.

[35] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[37] J. Heißwolf, R. König, and J. Becker. A Scalable NoC Router Design Providing QoS Support using Weighted Round Robin Scheduling. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 625–632, 2012.

[38] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[39] M. Horowitz. Computing's Energy Problem (and What We Can Do About It). In *IEEE ISSCC*, pages 10–14, 2014.

[40] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[41] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and< 0.5 MB Model Size. *arXiv preprint arXiv:1602.07360*, 2016.

[42] S. Ikehara and M. Miyazaki. Approximate Analysis of Queueing Networks with Nonpreemptive Priority Scheduling. In *Proc. 11th Int. Teletraffic Congr*.

[43] M. Imani, S. Gupta, Y. Kim, and T. Rosing. Floatpim: In-memory Acceleration of Deep Neural Network Training with High Precision. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 802–815, 2019.

[44] J. Jeffers, J. Reinders, and A. Sodani. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann, 2016.

[45] N. Jiang et al. A Detailed and Flexible Cycle-accurate Network-on-chip Simulator. In *2013 IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 86–96.

[46] X. Jin and G. Min. Modelling and Analysis of Priority Queueing Systems with Multi-class Self-similar Network Traffic: a Novel and Efficient Queue-decomposition Approach. *IEEE Trans. on Communications*, 57(5), 2009.

[47] J. A. Kahle et al. Introduction to the Cell multiprocessor. *IBM journal of Research and Development*, 49(4.5):589–604, 2005.

[48] H. Kashif and H. Patel. Bounding Buffer Space Requirements for Real-time Priority-aware Networks. In *Asia and South Pacific Design Autom. Conf.*, pages 113–118, 2014.

[49] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron Processor for Multiprocessor Servers. *IEEE Micro*, 23(2):66–76, 2003.

[50] W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, S. Yu, et al. A 65nm 4Kb Algorithm-dependent Computing-In-Memory SRAM Unit-macro with 2.3 ns and 55.8 TOPS/W Fully Parallel Product-sum Operation for Binary DNN Edge Processors. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 496–498.

[51] A. E. Kiasari, Z. Lu, and A. Jantsch. An Analytical Latency Model for Networks-on-Chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):113–123, 2012.

[52] A. E. Kiasari, Z. Lu, and A. Jantsch. An Analytical Latency Model for Networks-on-Chip. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 21(1):113–123, 2013.

[53] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[54] D. Kouvatsos and P. Luker. On the analysis of queueing network models: Maximum entropy and simulation. In *UKSC 84*, pages 488–496. 1984.

[55] D. D. Kouvatsos. Entropy Maximisation and Queuing Network Models. *Annals of Operations Research*, 48(1):63–126, 1994.

[56] G. Krishnan, S. K. Mandai, C. Chakrabarti, J.-s. Seo, U. Y. Ogras, and Y. Cao. Interconnect-centric benchmarking of in-memory acceleration for dnns. In *2021 China Semiconductor Technology International Conference (CSTIC)*, pages 1–4. IEEE, 2021.

[57] G. Krishnan, S. K. Mandal, C. Chakrabarti, J.-s. Seo, U. Y. Ogras, and Y. Cao. Interconnect-Aware Area and Energy Optimization for In-Memory Acceleration of DNNs. *IEEE Design & Test*, 2020.

[58] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[60] H. Kwon, A. Samajdar, and T. Krishna. Rethinking Nocs for Spatial Neural Network Accelerators. In *2017 Eleventh IEEE/ACM Intl. Symp. on NOCS*, pages 1–8, 2017.

[61] H. Kwon, A. Samajdar, and T. Krishna. Maeri: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *ACM SIGPLAN Notices*, volume 53, pages 461–475, 2018.

[62] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[63] Y.-L. Lee, J. M. Jou, and Y.-Y. Chen. A High-speed and Decentralized Arbiter Design for NoC. In *2009 IEEE/ACS International Conference on Computer Systems and Applications*, pages 350–353, 2009.

[64] A. Lerer et al. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287*, 2019.

[65] R. Leupers et al. Virtual Manycore platforms: Moving towards 100+ processor cores. In *Proc. of DATE*, pages 1–6, 2011.

[66] S. Liang et al. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Transactions on Computers*, 2020.

[67] M. Lin, Q. Chen, and S. Yan. Network in Network. *arXiv preprint arXiv:1312.4400*, 2013.

[68] W. Liu and B. Vinter. A Framework for General Sparse Matrix–Matrix Multiplication on GPUs and Heterogeneous Processors. *Journal of Parallel and Distributed Computing*, 85:47–61, 2015.

[69] Z. Lu, M. Zhong, and A. Jantsch. Evaluation of On-chip Networks using Deflection Routing. In *Proceedings of the 16th ACM Great Lakes Symposium on VLSI*, pages 296–301, 2006.

[70] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo. Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1354–1367, 2018.

[71] P. S. Magnusson et al. Simics: A Full System Simulation Platform. *Computer*, 35(2):50–58.

[72] S. K. Mandal, R. Ayoub, M. Kishinevsky, M. M. Islam, and U. Y. Ogras. Analytical Performance Modeling of NoCs under Priority Arbitration and Bursty Traffic. *IEEE Embedded Systems Letters*, 2020.

[73] S. K. Mandal, R. Ayoub, M. Kishinevsky, and U. Y. Ogras. Analytical performance models for nocs with multiple priority traffic classes. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–21, 2019.

[74] S. K. Mandal, A. Krishnakumar, R. Ayoub, M. Kishinevsky, and U. Y. Ogras. Performance analysis of priority-aware nocs with deflection routing under traffic congestion. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[75] S. K. Mandal, A. Krishnakumar, and U. Y. Ogras. Energy-efficient networks-on-chip architectures: Design and run-time optimization. *Network-on-Chip Security and Privacy*, page 55, 2021.

[76] S. K. Mandal, G. Krishnan, C. Chakrabarti, J.-S. Seo, Y. Cao, and U. Y. Ogras. A latency-optimized reconfigurable noc for in-memory acceleration of dnns. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(3):362–375, 2020.

[77] C. D. Manning, C. D. Manning, and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT press, 1999.

[78] H. Mao et al. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 270–288. 2019.

[79] M. Mao, X. Peng, R. Liu, J. Li, S. Yu, and C. Chakrabarti. MAX2: An ReRAM-based Neural Network Accelerator that Maximizes Data Reuse and Area Utilization. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[80] T. Moscibroda and O. Mutlu. A Case for Bufferless Routing in On-chip Networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 196–207, 2009.

[81] L. Ni, H. Huang, Z. Liu, R. V. Joshi, and H. Yu. Distributed In-Memory Computing on Binary RRAM Crossbar. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

[82] L. M. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *Computer*, 26(2):62–76, 1993.

[83] U. Y. Ogras, P. Bogdan, and R. Marculescu. An Analytical Approach for Network-on-Chip Performance Analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):2001–2013, 2010.

[84] U. Y. Ogras, Y. Emre, J. Xu, T. Kam, and M. Kishinevsky. Energy-Guided Exploration of On-Chip Network Design for Exa-Scale Computing. In *Proc. of Intl. Workshop on System Level Interconnect Prediction*, pages 24–31, 2012.

[85] U. Y. Ogras, M. Kishinevsky, and S. Chatterjee. xPLORE: Communication Fabric Design and Optimization Framework. Developed at Strategic CAD Labs, Intel Corp.

[86] U. Y. Ogras and R. Marculescu. *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, volume 184. Springer Science & Business Media, 2013.

[87] M. Palesi and T. Givargis. Multi-objective Design Space Exploration Using Genetic Algorithms. In *Proc. of the Intl. Symp. on Hardware/-Software Codesign*, pages 67–72, 2002.

[88] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance Evaluation and Design Trade-offs for Network-on-Chip Interconnect Architectures. *IEEE transactions on Computers*, 54(8):1025–1040, 2005.

[89] A. Patel et al. MARSS: a Full System Simulator for Multicore x86 CPUs. In *Design Autom. Conf.*, pages 1050–1055, 2011.

[90] A. Pellegrini et al. The Arm Neoverse N1 Platform: Building Blocks for the Next-gen Cloud-to-Edge Infrastructure SoC. *IEEE Micro*, 40(2):53–62, 2020.

[91] X. Peng, R. Liu, and S. Yu. Optimizing Weight Mapping and Data Flow for Convolutional Neural Networks on RRAM based Processing-In-Memory Architecture. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2019.

[92] M. Petracca, B. G. Lee, K. Bergman, and L. P. Carloni. Photonic NoCs: System-Level Design Exploration. *IEEE Micro*, 29(4):74–85, 2009.

[93] G. Pujolle and W. Ai. A Solution for Multiserver and Multiclass Open Queueing Networks. *INFOR: Information Systems and Operational Research*, 24(3):221–230, 1986.

[94] Y. Qian, Z. Lu, and Q. Dou. Qos Scheduling for NoCs: Strict Priority Queueing versus Weighted Round Robin. In *2010 IEEE International Conference on Computer Design*, pages 52–59, 2010.

[95] Y. Qian, Z. Lu, and W. Dou. Analysis of Worst-case Delay Bounds for Best-effort Communication in Wormhole Networks on Chip. In *2009 3rd ACM/IEEE Interl. Symp. on Networks-on-Chip*, pages 44–53.

[96] Z.-L. Qian et al. A Support Vector Regression (SVR)-based Latency Model for Network-on-Chip (NoC) Architectures. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 35(3):471–484, 2015.

[97] X. Qiao, X. Cao, H. Yang, L. Song, and H. Li. Atomlayer: A Universal ReRAM-based CNN Accelerator with Atomic Layer Computation. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6.

[98] A. Rico et al. ARM HPC Ecosystem and the Reemergence of Vectors. In *Proc. of the Computing Frontiers Conf.*, pages 329–334. ACM, 2017.

[99] E. Rotem and S. P. Engineer. Intel Architecture, Code Name Skylake Deep Dive: A New Architecture to Manage Power Performance and Energy Efficiency. In *Intel Developer Forum*, 2015.

[100] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. ISAAC: A Convolutional Neural Network Accelerator with in-situ Analog Arithmetic in Crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 14–26, 2016.

[101] E. S. Shin, V. J. Mooney III, and G. F. Riley. Round-robin Arbiter Design and Generation. In *Proceedings of the 15th international symposium on System Synthesis*, pages 243–248, 2002.

[102] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[103] M. P. Singh and M. K. Jain. Evolution of Processor Architecture in Mobile Phones. *Intl. Journ. of Computer Applications*, 90(4), 2014.

[104] A. Sodani et al. Knights Landing: Second-generation Intel Xeon Phi Product. *Ieee micro*, 36(2):34–46, 2016.

[105] L. Song, X. Qian, H. Li, and Y. Chen. Pipelayer: A Pipelined ReRAM-based Accelerator for Deep Learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552. IEEE, 2017.

[106] S. M. Tam et al. SkyLake-SP: A 14nm 28-Core Xeon® Processor. In *2018 IEEE ISSCC*, pages 34–36, 2018.

[107] C. Tian, L. Ma, Z. Yang, and Y. Dai. PCGCN: Partition-Centric Processing for Accelerating Graph Convolutional Network. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 936–945, 2020.

[108] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma. A 64-tile 2.4-Mb In-Memory-Computing CNN Accelerator Employing Charge-domain Compute. *IEEE Journal of Solid-State Circuits*, 54(6):1789–1799, 2019.

[109] S. R. Vangal et al. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, 2008.

[110] J. Walraevens. *Discrete-time Queueing Models with Priorities*. PhD thesis, Ghent University, 2004.

[111] L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan. Graph attention convolution for point cloud semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10296–10305, 2019.

[112] L. Wang, G. Min, D. D. Kouvatsos, and X. Jin. Analytical Modeling of an Integrated Priority and WFQ Scheduling Scheme in Multi-service Networks. *Computer Communications*, 33:S93–S101, 2010.

[113] D. Wentzlaff et al. On-chip Interconnection Architecture of the Tile Processor. *IEEE micro*, 27(5):15–31, 2007.

[114] P. Wettin et al. Performance Evaluation of Wireless NoCs in Presence of Irregular Network Routing Strategies. In *Proc. of the conf. on DATE*, page 272, 2014.

[115] Y. Wu et al. Analytical Modelling of Networks in Multicomputer Systems under Bursty and Batch Arrival Traffic. *The Journ. of Supercomputing*, 51(2):115–130, 2010.

[116] G. Xiaopeng, Z. Zhe, and L. Xiang. Round Robin Arbiters for Virtual Channel Router. In *The Proceedings of the Multiconference on" Computational Engineering in Systems Applications"*, volume 2, pages 1610–1614, 2006.

[117] S. Xie, A. Kirillov, R. Girshick, and K. He. Exploring Randomly Wired Neural Networks for Image Recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1284–1293, 2019.

[118] M. Yan et al. Hygcn: A gcn accelerator with hybrid architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–29. IEEE, 2020.

[119] L. Yang, Z. He, Y. Cao, and D. Fan. Non-uniform DNN Structured Subnets Sampling for Dynamic Inference. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[120] S. Yin, Z. Jiang, M. Kim, T. Gupta, M. Seok, and J.-s. Seo. Vesti: Energy-Efficient In-Memory Computing Accelerator for Deep Neural Networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(1):48–61, 2019.

[121] R. Ying et al. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018.

[122] S. Yoo, G. Nicolescu, L. Gauthier, and A. A. Jerraya. Automatic Generation of Fast Timed Simulation Models for Operating Systems in SoC Design. In *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, pages 620–627, 2002.

[123] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang. Slimmable Neural Networks. In *International Conference on Learning Representations*, 2018.

[124] C. A. Zeferino and A. A. Susin. SoCIN: A Parametric and Scalable Network-on-Chip. In *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.*, pages 169–174, 2003.

[125] B. Zoph and Q. V. Le. Neural Architecture Search with Reinforcement Learning. *arXiv preprint arXiv:1611.01578*, 2016.

[126] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.