Energy-efficient Communication Architectures for beyond von-Neumann AI Accelerators: Design and Analysis

by

Sumit K. Mandal

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Department of Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2022

Date of final oral examination: 04/22/2022

The dissertation is approved by the following members of the Final Oral 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

Dedicated to my parents Abha Mondal and Sasthi Pada Mandal.

ACKNOWLEDGMENTS

This dissertation is the result of efforts over the last five years of my graduate study and over 20 years of education. The last five years have been one of the most productive phase of my life and it would not have been possible without the support of faculty, colleagues, friends, and family. First and foremost, I would like to thank my advisor, Prof. Umit Y. Ogras, for the guidance, patience, and advice he has offered me over the years of my graduate study. His insightful recommendations on writing, time management, communication, and networking helped me grow as a researcher and person. I am also grateful for his help and support in my academic job search. His thoughtful comments and advice on the interview process helped me greatly in securing an academic position. Prof. Ogras did everything to ensure his students' success, which is something I will strive to achieve in my career.

I would like to thank Prof. Chaitali Chakrabarti, Prof. Mikko H. Lipasti and Prof. Karthikeyan Sankaralingam for being part of my Ph.D. defense committee. Their insightful comments helped me in improving this dissertation. I am also thankful to Prof. Partha Pratim Pande, Prof. Janardhan Rao Doppa, Prof. Prabhat Mishra, Prof. Yu Cao, Prof. Jae-sun Seo, and Prof. Sudip Pasricha for their inputs and guidance in my research over the past few years. I am also thankful to Dr. Michael Kishinevsky and Dr. Raid Ayoub for their continued mentorship.

I am grateful for the interesting discussions, words of encouragement, and lighter moments with friends and colleagues: Ujjwal, Ganapati, Samet, Manoj, Darshan, Ranadeep, Anish, Yigit, Sizhe, Alper, Toygun, Shruti, Jie, Harsh and Gokul. I also like to thank my other friends for their continuous encouragement and support: Priyabrata, Siddhartha, Gaurav, Subarna, Sudhir and Sanvi. I enjoyed the numerous trips that I took with them.

Finally, I would like to thank my parents, Abha Mondal and Sasthi

Pada Mandal, for their constant love, support, and encouragement over the years. It is their love and encouragement that motivated me to pursue graduate studies and contribute to research. I also thank my cousins and friends back in India, with whom I have numerous happy moments.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

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.

Graph convolutional networks (GCNs) have shown remarkable learning capabilities when processing data in the form of graph which is found inherently in many application areas. To take advantage of the relations captured by the underlying graphs, GCNs distribute the outputs of neural networks embedded in each vertex over multiple iterations. Consequently, they incur a significant amount of computation and irregular communication overheads, which call for GCN-specific hardware accelerators. We propose a communication-aware in-memory computing architecture (COIN) for GCN hardware acceleration. Besides accelerating the computation using custom compute elements (CE) and in-memory computing, COIN aims at minimizing the intra- and inter-CE communication in GCN operations to optimize the performance and energy efficiency. Experimental evaluations with various datasets show up to 174$\times$ improvement in energy-delay product with respect to Nvidia Quadro RTX 8000 and edge

GPUs for the same data precision.

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. Specifically, 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$\times$ speedup in full-system simulation.

1 INTRODUCTION

In recent years, deep neural networks (DNNs) and graph convolutional networks (GCNs) have shown tremendous success in recognition and detection tasks such as image processing, health monitoring, and language processing [116, 145]. 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) [212, 224, 225], 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 [147, 188]. Figure 3.2 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 [88] and DNN accelerators [44, 183]. Eyeriss-V2 [44] proposes to use three different NoCs for weights, activations, and partial sums. Such

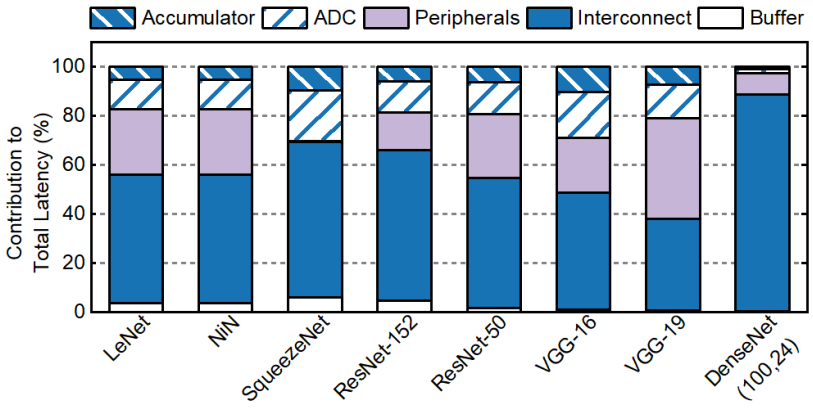


Figure 1.1: Percentage contribution of different components of the IMC hardware to total latency. 40%-90% of the total latency is spent on on-chip communication when bus-based H-Tree interconnect is used.

an architectural choice allows for higher performance at the cost of both area and energy consumption. ISAAC [183] 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 [183].

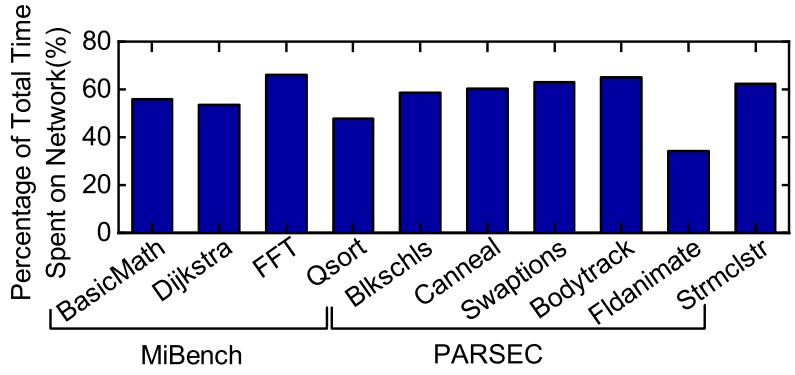


Figure 1.2: Experiments of different applications show that 40%-70% of the total simulation time is spent on the network.

Furthermore, we also proposed IMC-based communication-aware hardware accelerator for graph convolutional networks (GCNs). We perform rigorous experimental evaluation across popular graph datasets for GCN and comparison with respect to state-of-the-art GPUs and accelerator. The evaluation shows that our proposed GCN accelerator achieves up to 105× lower energy consumption with respect to state-of-the-art GCN accelerator.

Since the on-chip interconnect is a critical component of AI accelerators as well as multicore architectures, pre-silicon evaluation platforms contain cycle-accurate NoC simulators [6, 90]. 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 1.2 shows that 40%-70% of total simulation time is spent on the network itself when performing full-system simulation using gem5 [24]. 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 [158, 210, 174]. 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 [88, 178, 97]. First, routers give priority to the flits in the network to achieve predictable latency within the interconnect. flits to the local node in Figure 1.3 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

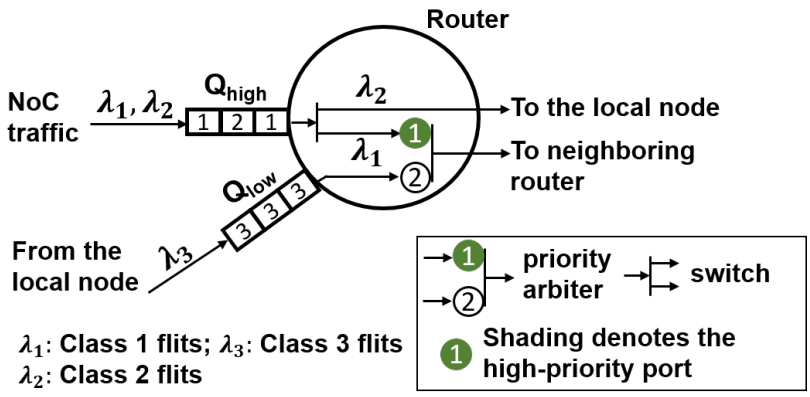


Figure 1.3: The high priority queue (Q_{high}) stores two different traffic classes which are already in the NoC, while the low priority queue (Q_{low}) stores the newly injected flits from the local node. As flits from class–2 are routed to the local node, low-priority flits compete with only class–1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

Figure 1.3 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 5.2. In contrast, analytical models that ignore priority would significantly underestimate the latency. Thus, prior approaches that do not model priority [158, 210, 174] and simple performance models for the priority queues [102, 91] are *inapplicable to priority-based industrial NoCs*.

We propose 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 1.3. 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.

In summary, this dissertation makes the following contributions:

- A comprehensive literature survey on energy-efficient communication architectures for general purpose processor as well as AI accelerator [137],
- Communication-aware hardware accelerator for DNN [111, 141],

- Fast and accurate performance analysis for NoCs with priority arbitration [136, 135, 139],
- Fast and accurate performance analysis technique for NoCs with weighted round robin arbitration [144].

The rest of the thesis is organized as follows. The thesis consists of communication-aware hardware accelerator for DNNs (Chapter 3) and GCNs (Chapter 4). It also spans performance analysis techniques for priority-aware NoCs (Chapter 5 – Chapter 7) as well as NoCs with weighted round robin arbitration under bursty traffic and deflection routing (Chapter 8). Finally, Chapter 9 concludes the thesis.

2.1 Energy-efficient Communication Architectures for General Purpose Processors

Diminishing instruction-level parallelism (ILP) and power wall led to the introduction of multicore NoC architectures more than a decade ago and fueled their growth to date [48, 87]. In turn, the growing number of cores has continuously increased the importance of efficient data movement between cores and memory. One can view the communication cost as a "necessary overhead" incurred to move data both within a chip or across multiple chips. Thus, we can evaluate the efficiency of communication architectures as a function of their performance, measured in terms of latency and throughput, versus their cost, measured by their contribution to power and energy consumption. Besides facilitating the design process through reuse and regularity, networks-on-chip architectures have proven their effectiveness with respect to this efficiency metric and become the mainstream communication fabric choice for multicore architectures [187, 168].

The transition from single core to multicore architectures played a pivotal role in satisfying the continuous demand for higher processing power. However, homogeneous multicore architectures have also reached their limits to exploit thread- and data-level parallelism. General-purpose cores facilitate programmability, but their flexibility comes at the expense of energy efficiency, which is orders of magnitude lower than special-purpose processors, such as video processors and hardware accelerators. Therefore, the heterogeneity of cores increases together with their number to drive the processing power and energy efficiency necessitating several

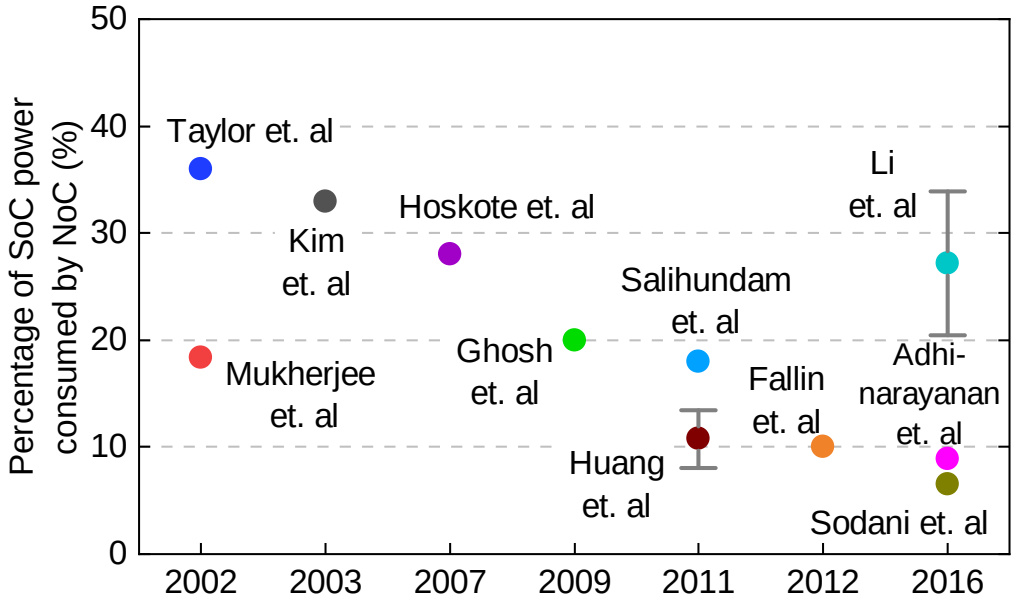


Figure 2.1: Percentage of the total power consumed by NoC over the past two decades.

power management techniques [55, 21, 73, 138, 109, 137, 143, 22]. In turn, the growing heterogeneity continues to increase the requirements on the communication architectures. On the one hand, communication latency should be in the same order as, or ideally lower than, the processing times of special-purpose processors, which could be in the order of nanoseconds. On the other hand, the power consumption overhead should scale down to match those of the special-purpose processors. Otherwise, the communication cost can undermine the benefits of heterogeneous architectures. Hence, the NoCs continue to play a pivotal role in developing processing systems with higher performance and energy efficiency.

NoC architectures are designed to meet the performance requirements, such as latency, throughput, and quality-of-service (QoS), while minimizing their area, power, and energy overhead. NoCs achieve a higher energy-efficiency than bus and point-to-point communication architec-

tures [121]. The exact contribution of the NoC architecture to the total power consumption is a function of many design parameters, such as the number and types of processing cores, communication bandwidth, and technology. While there is no public data for commercial designs, published academic work shows that NoC power consumption amounts to 18%–36% of the total chip power, as shown in Figure 2.1. Ghosh et. al [67], Huang et. al [83] and Li et. al [125] report absolute values, while Taylor et. al [194], Kim et. al [103], Mukherjee et. al [153], Hoskote et. al [79], Salihundam et. al [181], Fallin et. al [60], Sodani et. al [187] and Adhinaryanan et. al [5] report the percentage power consumption, which can be useful references for future work. While many other surveys discuss the energy-efficient design of and entire SoC [197, 166], in this chapter we discuss numerous prior work on NoC design.

NoC Router Design

Router design has a crucial impact on power consumption and performance since they are the fundamental building blocks of NoCs. On the one hand, fast and simple router architectures are preferred to achieve low latency (in terms of clock cycles) and small area overhead. On the other hand, deeper pipelines and more buffering space are needed for higher clock frequencies and throughput [49, 57]. Similarly, simpler crossbar and arbitration, such as round-robin, reduce the area-power overhead, but they can also severely limit the performance. As a result, the optimal router design must provide an intricate balance between power, performance, energy, and area.

Many techniques have been proposed to improve the router energy-efficiency since the introduction of NoCs [4, 148]. For example, Wang et al. explore the bottlenecks that contribute to power dissipation in interconnection networks [203]. The analysis prompts the authors to devise power-efficient microarchitecture techniques such as a cut-through cross-

bar, a segmented crossbar, and a write-through buffer. These techniques achieve an NoC power reduction of up to 44.9% and 37.9% with no performance overheads for both synthetic and real application traces, respectively. One of the early key design insights is exploiting application- or domain-specific information to customize the router architecture [81]. Hu et al. use this idea to allocate a given amount of buffer area non-uniformly across the NoC to maximize the performance [81]. Similarly, router architecture with shared buffers can improve both the area utilization and throughput in [177]. Decreasing the router pipeline delay has been an important design consideration. The basic idea is to bypass certain router pipeline stages or the whole router whenever possible, i.e., when there is no contention and packets already waiting in the queue [162, 108]. Towards achieving a single-cycle router latency, Kumar et al. [117] present a non-speculative single-cycle NoC router pipeline targeting 3.6 GHz frequency. A customized mesh NoC that exploits application-specific behavior to introduce single-cycle links between a source and destination is introduced in [39]. These clockless repeater links traverse upto 8mm in a single cycle at a clock frequency of 2 GHz in a 45nm process node. The application-specific link configuration introduces complexities in reconfiguring the routers when applications are frequently preempted during execution by the system scheduler. A 2-dimensional mesh NoC coprocessor is designed for data movement in an Epiphany 64-core superscalar processor architecture [200]. Each router uses three communication channels for read requests, on-chip writes, and off-chip write transactions to improve the NoC throughput. Recently, Arm has introduced a scalable low-latency, one cycle per-hop, high-bandwidth network-on-chip architecture that supports coherency [168]. The NoC can scale from 1×2 mesh to 8×8 mesh to support edge devices and high-performance computing systems.

NoC Architecture and Packet Routing

The interconnection of the routers defines the NoC architecture. Most of the early work and industrial solutions employ a 2D mesh topology due to its regularity [151, 168, 187]. A regular layout simplifies the floorplan and wiring, while providing predictable delays between the routers. Consequently, regular NoC topologies facilitate energy-efficient design as well as simplified testing and validation [87, 148]. However, both the average and worst-case latency scale poorly with the network size. Since the NoC plays a crucial role in determining the latency, energy and power, numerous studies evaluate other topologies with smaller diameter, such as hypercube, folded-torus, and fat tree [4, 57]. For example, an extensive power-performance evaluation of mesh and torus topologies on power and performance is presented in [151]. NoC topology can be optimized for a given application domain by either synthesizing a custom topology [161, 170] or altering a regular topology, such as a 2D mesh [162]. Kilo-NoC, a heterogeneous NoC architecture in which only a select number of routers support QoS requirements is presented in [72]. The heterogeneity in router architecture enables 45% and 29% savings in area and power when compared to a homogeneous NoC that supports QoS in all routers.

Switching, or flow control, techniques determine how the packets are transmitted from their sources to destinations [49, 57, 87]. Due to the large buffer requirements of the packet and virtual cut-through switching techniques, wormhole routing has been a popular choice for NoCs [148]. Virtual-channels have commonly been added to address head-of-line blocking and handling multiple traffic classes while avoiding deadlocks. Circuit switching have also been employed to take advantage of high-volume data transfer over persistent connections [130]. While the early work focused on buffered NoCs, bufferless solutions are later employed to minimize the buffer requirements, hence the NoC area and static power [52, 150]. Bufferless NoCs do not store the packets in the intermediate routers. Routers try

to forward the packets towards their destinations and then deflect them if their preferred direction is not available [87]. Studies show that bufferless NoC can perform better at low traffic loads, but they suffer from traffic congestion as the traffic load increases [150]. The advantages of buffered and bufferless techniques are combined in [60] to achieve as much as 16% higher energy-efficiency than the state-of-the-art buffered routers. Intel Xeon-Phi processors [187] use priority-aware bufferless NoCs, which can provide predictable latency within the network [135]. The NoC uses fewer buffers, which in turn helps to reduce energy consumption.

Routing strategies play a crucial role in determining the area, performance, power and energy of an NoC [49, 57]. For a given NoC topology and switching technique, routing algorithms determine the path taken by each packet while traveling to their destination. The choice of the routing algorithm is important since the length of the path and traffic congestion have a significant impact on both power and performance, while their complexity impacts the area [148]. Specifically, routing techniques influence: (1) end-to-end latency, (2) selection of routing paths, (3) livelock and deadlock avoidance, (4) fault-tolerance and (5) starvation avoidance. Routing techniques are broadly classified into deterministic and adaptive routing techniques. Deterministic algorithms require fewer resources and are simpler than their adaptive counterpart. Therefore, dimension-ordered routing, such as XY routing, and other deterministic algorithms are used more commonly in industrial and academic designs [17, 187]. At the same time, run-time adaptation to the workload can provide significant benefits [80]. Since the traditional routing techniques are covered in detail in the literature, we refer the reader to existing surveys and books on this topic for a complete taxonomy and review [4, 49, 57, 163, 89].

3D NoC Architectures

As the number of transistors on a single chip increases, three-dimensional integrated circuits (3D ICs) provide more floorplanning flexibility than traditional planar designs [62]. 3D ICs also provide more packaging density since they are not limited to two dimensions [51]. Moreover, 3D ICs can reduce power consumption since they use shorter wire lengths than planar ICs [176]. Since they have emerged as a new technology paradigm due to these advantages, 3D NoCs are employed to interconnect the cores in 3D IC.

Feero et al. [62] present a detailed performance evaluation of 3D NoCs and compare them to traditional 2D NoCs. This work considers 3D mesh, 3D stacked mesh, ciliated 3D mesh, 3D butterfly tree (BFT), and 3D fat-tree topologies. The experimental evaluations show that 3D mesh, ciliated 3D mesh, and stacked mesh consume 42%, 47%, 33% less energy per packet than 2D mesh, respectively, Authors also show that both 3D fat-tree and 3D BFT consume 49% less energy per packet than their 2D counterpart. Similar to the dimension ordered routing techniques in 2D NoCs, an X–Y–Z static routing can be applied to 3D NoC. For example, Ahmed et al. [7] present a 3D NoC (OASIS-NoC) with wormhole switching and 3-stage router pipeline: routing calculation, switch allocation, and switch traversal. Evaluations using synthetic traffic show that a $2\times2\times4$ 3D OASIS-NoC reduces 22% delay on average compared to a 4×4 2D OASIS-NOC. Since a fixed design may not be suitable for different applications, authors in [154] propose a synthesis approach to construct a power-efficient 3D NoC for a given application. Experiments on real applications show that the NoC topologies synthesized by the proposed methodology reduce power by 38% on average. Similarly, a floorplan-aware application-specific 3D NoC synthesis algorithm is proposed in [223]. The authors construct an irregular 3D NoC architecture that meets specific objectives for a given application in this work. The proposed algorithm's input is a directed

graph, where each node represents a core, and the edges represent the traffic flow between the cores. A multi-commodity flow (MCF) problem is formulated to optimize multiple objectives. The objectives include network power, average network latency and number of through silicon vias (TSV). Authors incorporate simulated allocation (SAL) to solve the MCF problem. The proposed methodology enables 22% power saving with respect to [154] for a set of synthetic benchmarks [214]. A summary of different 3D NoC technologies, their advantages, and drawbacks can be found in [176].

Several research teams have recently applied machine-learning techniques to design 3D NoCs [50, 95, 156]. For example, Das et al. [50] present a monolithic 3D-enabled energy-efficient NoC. Smaller dimensions of monolithic inter-tier vias provide the scope of high-density integration with reduced wire length compared to TSVs. Experimental evaluations show that the proposed methodology enables 32% lower energy-delay-product (EDP) than mesh-based interconnect and 28% lower EDP than TSV-based interconnect. TSV-based 3D NoCs are prone to failure. Therefore, a Near Field Inductive Coupling (NFIC)-based 3D NoC is proposed in [69]. In this work, convex optimization is used to co-optimize latency, power and area of the 3D NoC. Experimental evaluation on real benchmarks shows that the proposed NFIC-based 3D NoC is 34.5% more energy-efficient than TSV-based 3D NoC.

Joardar et al. [95] propose a 3D-NoC for heterogeneous many-core systems. The authors employ a machine learning-based multi-objective optimization (MOO) to construct 3D-NoC, which jointly optimizes latency, throughput, temperature, and energy. The proposed algorithm determines the optimal placement of the CPUs, GPUs, LLCs, and planar links. The authors show that the proposed technique enables 9.6% better EDP than a thermally-optimized 3D NoC design.

2.2 Communication-centric AI Accelerator Design

In-memory computing technique has a great potential to deliver energy-efficient AI accelerators. Most of the AI applications consist of layer-by-layer operation, i.e., the output to the k^{th} layer is the input of the $(k+1)^{th}$ layer. In IMC-based accelerators, the data movement (i.e., communication) between the DNN layers are enabled by on-chip interconnects. Since the number of parameters of the neural networks has grown over past ten years as shown in [140], the on-chip communication volume also increases. Increasing communication volume, in turn, increases energy consumption due to communication which can mask the energy benefit of IMC technology itself. Therefore, an energy-efficient communication strategy is required for AI accelerator to compliment the energy benefits of IMC technology. Indeed, a single technique may not be suitable for all kinds of AI algorithms as discussed in [112].

There exist several NoC architectures for DNN accelerators. A recent study aims to maximize local data reuse and reduce data access from DRAM [43]. To this end, a row-stationary data flow is proposed, where filter weights and input feature maps (ifmap) are reused to minimize movement of ifmaps and filter weights. The architecture has later been extended to incorporate compact and sparse neural networkss in [44]. In the extended version [44], a hierarchical mesh NoC is incorporated in the architecture. It consists of 16 PE clusters and 16 global buffer clusters distributed in an 8×2 array. Each PE cluster consists of 12 PEs arranged in a 3×4 array. However, both architectures consider a system with off-chip memory where frequent data transfer from off-chip to on-chip is required. Therefore, the NoC optimizations incorporated in these architectures are not applicable for IMC-based accelerators. ISAAC is the one of the first IMC-based DNN accelerators proposed in the literature [183]. It uses

an NoC with concentrated mesh (c-mesh) topology. However, only one NoC router is connected to each IMC tile and no special interconnect optimization is considered in this architecture. Since larger DNNs (e.g., DenseNet with 100 layers) may contain 100s of IMC tiles, this architecture may require 100s of NoC routers, which may not be practical at all.

The growing number of NoC routers increase area as well as on-chip interconnect power consumption. To that end, we propose an optimization technique to determine number of NoC routers for a given DNN [111]. We first construct an objective function for communication energy, which considers the number of activations between two consecutive layers for each layer as input. Then the objective function is minimized to obtain the number of routers needed for all layers of the DNN. A scheduling technique is also proposed in this work to minimize the congestion in the on-chip network. The optimized number of routers along with the scheduling technique provides up to 78% improvement in energy-delay product with respect to another DNN accelerator [183]. Although this work minimizes congestion in the on-chip network with a scheduling technique, it does not guarantee minimum latency for a given DNN. We also propose an NoC architecture which guarantees minimum possible communication latency for a given DNN [141]. However, the proposed NoC architecture is customized for a single DNN. Therefore, a reconfigurable NoC is also proposed in [111], where a certain number of routers (determined with handful DNNs known at the design time) are allocated for each DNN layer. At runtime, if a new (not considered in the design time) DNN appears, then first the number of routers required for each DNN layer is computed first. If the number of routers required for a particular layer is more than the number of available routers on-chip, then the DNN layer will occupy the maximum number of routers available for that layer. The reconfigruable NoC shows 60%–80% improvement in communication latency over state-of-the-art 2D-mesh NoC.

The area of monolithic hardware accelerators increases with increasing number of parameters of AI algorithms. Higher silicon area of a single monolithic system reduces the yield, which in turn increases fabrication cost [114]. Chiplet-based system solves the issue of higher fabrication cost by integrating multiple small chips (known as chiplets) on a single die. Since the area of each chiplet in the system is considerably lower than a monolithic chip (for the same AI algorithm), the yield of the chiplet-based system increases which reduces the fabrication cost. The communication between chiplets are performed through network-on-package (NoP). There are several works in the literature which propose NoP for chiplet-based system considering different performance objectives (e.g., latency, energy) [20, 201, 184, 114].

Kite is a family of NoP proposed in [20] which is mainly targeted for general purpose processors. In this work, three topologies are proposed – Kite-Small, Kite-Medium and Kite-Large. First, an objective function is constructed with combination of the average delay between source and destination, diameter and bisection bandwidth of the NoP. Experimental evaluations on synthetic traffic show that the proposed Kite topologies reduce latency by 7% and improve the peak throughput by 17% with respect to other well-known interconnect topologies. A chiplet-based system with 96-core processor, INTACT, is proposed in [201]. The chiplets are connected through a generic chiplet-interposer interfaces (called as 3D-plugs in the paper). 3D-plugs consist of micro-bump arrays. However, both Kite and INTACT are not specific to AI workloads.

Shao et al. designed and fabricated a 36-chiplet system called SIMBA for deep learning inference [184]. The chiplets in the system are connected through a mesh NoP. Ground-referenced signalling (GRS) is used for intra-package communication. The NoP follows a hybrid wormhole/cut-through flow control. The NoP bandwidth is 100 GBps/chiplet and the latency for one hop is 20 ns. Extensive evaluation on the fabricated chip

shows up tp 16% speed up compared to baseline layer mapping for ResNet-50. A simulator for chiplet-based system, SIAM, is proposed in [114], targeting AI workloads. In this simulator, a mesh topology is considered for NoP. It is shown that up to 85% of the total system area is contributed by NoP. In this work, multiple studies were performed by varying NoP parameters. For example, it is shown that increasing NoP channel width increases energy-delay product of the NoP for ResNet-110. This phenomenon is demonstrated for systems with 25 and 36 chiplets. However, none of the prior works considered any workload-aware optimization for the NoP. Therefore, there is ample opportunity of future research which considers NoP optimization for AI accelerators.

With increasing complexity of AI algorithms, the computing resource needed to execute the algorithms also increases. Therefore, complex AI algorithms require a large number of processing elements on-chip. For example, DenseNet-110 with 28.1M parameters requires 2,184 ReRAM tiles on a single system [114]. Increasing the number of on-chip tiles results in long-range inter-tile communication. Too many long-range communications hurt energy-efficiency of the system. Therefore, monolithic 3D (M3D)-based AI accelerators have emerged to facilitate energy-efficient communication between multiple processing elements. In M3D-based accelerators, multiple processing elements are placed in each plane. The processors across different planes are connected using through silicon vias (TSV).

REGENT is such an approach which integrates ReRAM-based IMC tiles as well as GPU cores on a M3D IC [96]. The processors in the IC are connected through a 3D-NoC. REGENT is optimized to perform energy-efficient CNN training. Specifically, a bin-package based framework is adopted to map CNN layers on processing cores as well as physically place the cores in such a way that the overall system meets certain performance objectives. However, REGENT does not consider hardware implementa-

tion of normalization layers. To address this drawback, authors in [94] propose a 3D-NoC enabled IMC-based system considering normalization layers. Apart from considering hardware implementation of normalization layers, a performance-thermal aware mapping of CNN layers is also proposed in this work. The mapping helps to reduce thermal noise which can degrades the quality of CNN training. As a result, the proposed architecture is able to perform CNN training which achieves accuracy similar to GPU. The accelerator proposed in the aforementioned work is further extended in [93] by considering fewer normalization layers for CNNs. In this work, the authors show that considering few normalization layers actually improves CNN classification accuracy, since normalization helps to reduce bias occurring from a weight with high absolute value. Then Bayesian optimization is utilized to construct a M3D system. The communication between multiple processing elements is facilitated by a mesh-NoC with XYZ routing. The accelerator proposed in this work reduces the latency by 15$\times$ compared to conventional GPU-based system. However, all these work only consider CNN training on IMC-based M3D system.

Recently, several other work proposed IMC-based M3D systems which are capable of training graph analytics [37] and graph neural networks (GNN) [142, 10, 12, 11]. ReGraphX is a 3D-NoC enabled heterogeneous IMC-based system which performs energy-efficient GNN training [10]. In this system, there are two types of processing elements: V-PEs, which perform vertex computations and E-PEs, which perform edge computation pertaining to a GNN. V-PEs consist of 128$\times$128 crossbar arrays whereas E-PEs consist of 8$\times$8 crossbar arrays. Experimental evaluations show that ReGraphX reduces energy consumption by 11$\times$ with respect to conventional GPUs. Authors in [12] show performance and accuracy trade-offs in 3D-NoC enabled IMC-based GNN accelerator. In this work, a stochastic rounding technique is proposed to reduce the precision of

ReRAM crossbar arrays. The reduced precision helps to improve energy-efficiency of the accelerator. A DropLayer-aware M3D-based manycore ReRAM architecture for training GNNs, DARe, is proposed in [11]. The droplayer-based technique reduces on-chip communication volume in the system, which in turn, prevents communication hotspot. Reduced communication hotspot improves the energy-efficiency of the overall system. The proposed architecture demonstrates 1.9× reduction in execution time with respect to ReGraphX [10]. Thus M3D-based systems with 3D NoC provide energy-efficient platform for CNN as well GNN training. Apart from DNN and graph applications, IMC-based accelerators are also proposed for recurrent neural networks (RNN) inference [36, 35], sorting application [127], video summarization [34].

2.3 Analytical Performance Modeling of Networks-on-Chip

Performance analysis techniques are useful for exploring design space [165] and speeding up simulations [158, 102, 210]. Indeed, there is continuous interest in applying novel techniques such as machine learning [174] and network calculus [173] to NoC performance analysis. However, these studies do not consider multiple traffic classes with different priorities. Since state-of-the-art industrial NoC designs [56, 88] 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 [99]. 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 [102]. This technique, however, assumes that each

queue in the network contains a single class of flits. 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 [136]. However, these models assume that the input traffic follows geometric distribution, which is not applicable for workloads with bursty traffic.

Deflection routing was first introduced in the domain of optical NoC as hot-potato routing [29]. Later, it was adapted for the NoCs used in high-performance SoCs to minimize buffer requirements and increase energy efficiency [152, 59, 60]. 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 [132] 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 [88]. 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.

An analytical bound on maximum delay in networks with deflection routing is presented in [31]. However, evaluating maximum delay is not useful since it leads to significant overestimation. Another analytical model for NoCs with deflection routing is proposed in [66]. 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.

Several techniques present performance analysis of priority-based

queuing networks outside the NoC domain [19, 28, 85]. Nevertheless, these techniques do not consider multiple traffic classes in the same queue. The work presented in [13] 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 [18] for a priority-based queuing system. This approach was later extended to multi-class systems [91]. 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 [202]. 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 8×8 NoC would have 64 sources and 64 destinations which will result in 4096 (64×64) variables with PGF. Furthermore, our approach outperforms this technique, as demonstrated in Chapter 5.

In contrast to prior approaches, we propose a scalable and accurate closed form solution for a priority-based queuing network with deflection routing executing multi-class bursty 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.

Multiple prior studies have proposed NoC performance analysis techniques with basic round-robin arbitration. [158, 63, 174]. Authors in [158] first construct a contention matrix between multiple flows in the NoC. Then, the average waiting time of the packets corresponding to each flow is computed. Support vector regression-based analytical model for NoCs is proposed in [174]. The analytical model proposed in [63] estimates the mean service time of the flows with RR arbitration. The estimated mean service time is used to find the average waiting time of the flows. However, none of these techniques are applicable in the presence of both bursty traffic and WRR arbitration.

Analytical modeling of round-robin arbitration has also been studied outside NoC domain [30, 71, 205]. The techniques presented in [30, 71] incorporate a polling model to approximate the effective service time of a queue in the presence of RR arbitration. However, none of these approaches are applicable when the input distribution to the queue is not geometric. A Markov chain-based analytical model is proposed in [205] to account for bursty input traffic. However, the technique is not scalable for a network of queues. Moreover, none of these techniques are applicable for discrete-time queuing systems. Since each transaction in NoC happens at discrete clock cycles, the analytical models need to incorporate discrete-time queuing systems. The major drawbacks of the prior approaches are summarized in Table 8.1.

The basic round-robin arbitration cannot provide fairness when requesters have widely varying data rate requirements and priorities. Therefore, weighted round-robin arbitration, i.e., WRR, has been used in on-chip communication architectures [172, 76]. Qian et al. compute delay bounds for different channels with different weights to assign appropriate weight to each input channel of the NoC [172]. They show that WRR delivers

Table 2.1: Comparison of prior research and our novel contribution.

Research	**Approach**	**WRR**	**Bursty Traffic**	**Scalable**	**Discrete Time**
Boxma et al. [30]	Polling model	**No**	No	Yes	No
Wim et al. [71]	Extended polling model	**No**	No	Yes	No
Wang et al. [205]	Markov chain	**No**	Yes	No	No
Fischer et al. [63]	Heuristic	**No**	No	Yes	No
Vanlerbergee et al. [199]	Moment generating function	**No**	No	No	Yes
This work	**Queue decomposition**	**Yes**	**Yes**	**Yes**	**Yes**

better quality of service than NoCs with strict priority-based arbitration. Authors in [76] propose a WRR-based scheduling policy. The proposed technique assigns larger bandwidth to input channels with higher weights. It achieves higher throughput compared to round-robin arbitration. *Although WRR has shown promise, no analytical modeling approach exists for NoCs with WRR to date.*

This work presents the first performance analysis technique for NoCs with WRR arbitration. It fills an essential gap since WRR can address the shortcomings of priority-based bufferless NoC architectures and the basic round-robin arbitration. Furthermore, the proposed technique supports bursty traffic observed in real applications, which is typically ignored due to its complexity. Hence, it is a vital step towards comprehending the theoretical underpinnings of NoCs with WRR arbitration and enabling their deployment in industrial designs.

3 COMMUNICATION-AWARE HARDWARE ACCELERATORS FOR DEEP NEURAL NETWORKS (DNNS)

3.1 Background and Motivation

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 [116, 145]. 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 [78]. 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 [183, 216, 188, 196]. Such advantages stem from the efficiency of matrix multiplication implementation in crossbar architectures. 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) [212, 224, 225, 126], further exacerbate the problem of on-chip communication due to larger

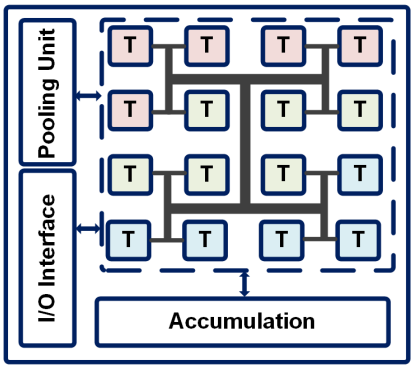


Figure 3.1: Multi-tiled IMC architecture with bus-based H-Tree interconnect [41].

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 [147, 188]. Figure 3.1 shows such a multi-tiled IMC architecture with bus-based H-Tree interconnect [41]. 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 [41] benchmarking tool. Figure 3.2 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 [88, 140] and DNN accelerators [44, 183]. Eyeriss-V2 [44] 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 area and energy consumption. ISAAC [183] 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

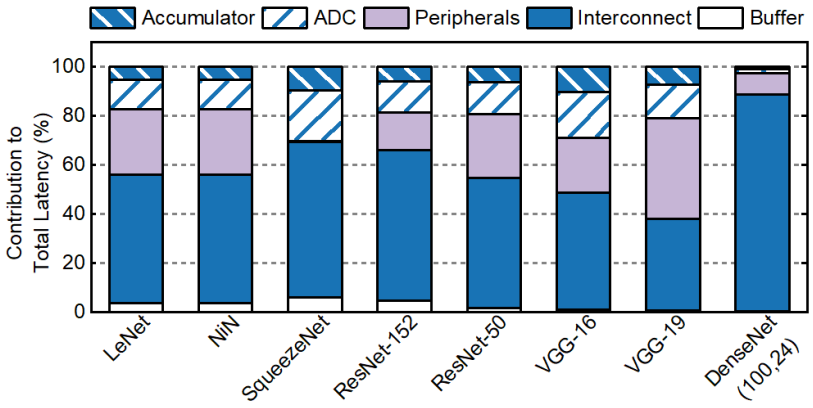


Figure 3.2: Percentage contribution of different components of the IMC hardware to total latency. 40%-90% of the total latency is spent on on-chip communication when bus-based H-Tree interconnect is used.

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 [183].

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 [219] and accuracy vs speed/power trade-off [215]. 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 [40] show the extensive usage of LeNet, SqueezeNet, VGG in various edge devices, and [61] 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 work 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 schedul-

ing 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.

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) [183, 188, 147, 100, 216]. Authors in [183] proposed a ReRAM-based IMC architecture for DNN inference. The architecture utilizes a crossbar of size 128×128 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, [188] 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 [175]. Authors in [147] proposed a systolic array-based ReRAM IMC design for DNN inference. Other works proposed in the past have explored SRAM-based IMC [216, 100].

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 [41, 169]. However, bus-based interconnect contributes up to 90% of the total inference latency, as shown in Figure 3.2. Hence, a bus-based interconnect 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 [183]. An IMC-based DNN accelerator for high-precision training is proposed in [86]. Ni *et al.* proposed a distributed in-memory computing architecture with a binary RRAM-based crossbar [157]. 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 [119] and Eyeriss-v2 [44] use a flexible interconnect for DNN accelerators. In MAERI [119], a fat-tree-based programmable interconnect is used to support various sparse and non-sparse DNN dataflows. The work in [44] 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 [133], DNN graph structure, and the computation-to-communication imbalance of the DNNs. A communication-centric IMC architecture is proposed in [111]. 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

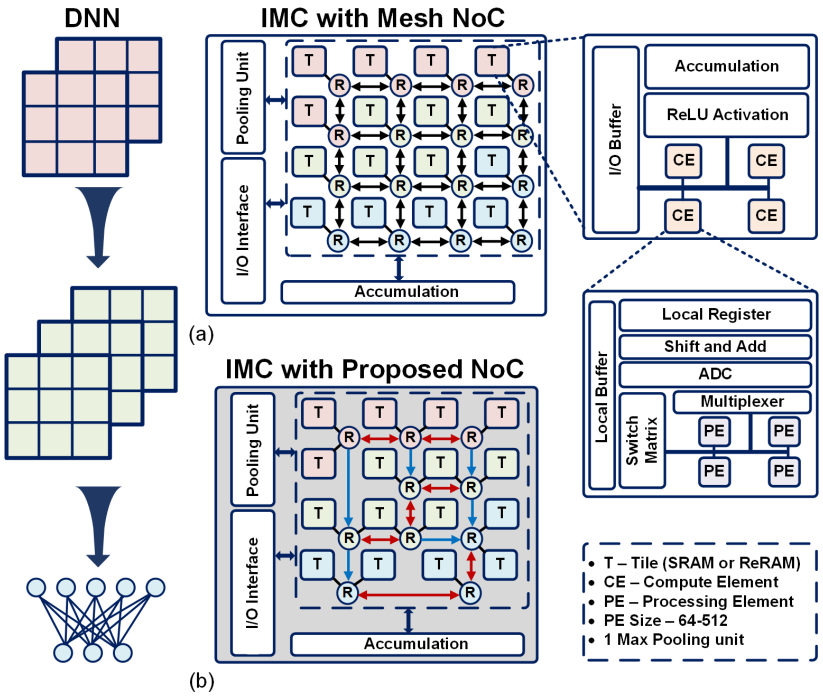


Figure 3.3: IMC architecture with an arbitrary DNN mapped onto different tiles with, (a) the mesh-NoC and (b) the proposed latency-optimized NoC.

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.

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 [183, 188, 196, 216]. 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 [183, 188].

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 3.3 shows a representative IMC hardware architecture for DNN inference acceleration [41]. 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 3.3(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 3.3(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 results in reduced energy and latency. The remaining details of the work is described in Appendix A of the thesis and in the reference [111, 141].

3.2 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 [41]. For such an architecture, one-to-one mapping of a router to tile [118] will require a large number of NoC routers and consume high power, as shown in Figure 3.4. 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 communicated 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:

$$E(\bar{\mathbf{n}}) = \Big(\sum_{k=1}^{K-1} \frac{I_k}{n_k n_{k+1}}\Big)\Big(\sum_{k=1}^{K} n_k\Big) \tag{3.1}$$

$E(\bar{\mathbf{n}})$ is proportional to the total communication energy of the DNN assuming that all transactions have a uniform size. We minimize this objective

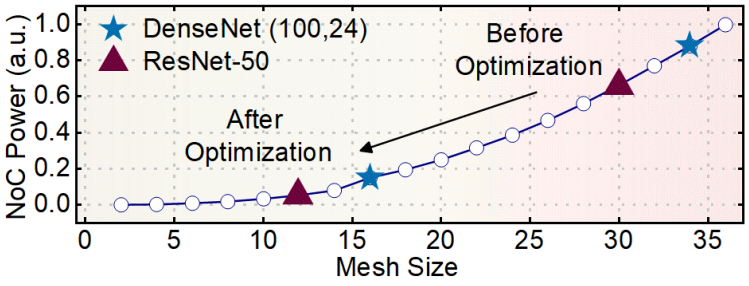


Figure 3.4: NoC optimization effectively reduces the power consumption because of its non-linear dependence on the mesh size. We obtain NoC power through BookSim [90] simulations.

function with an upper bound on the total number of routers, N as:

$$\begin{aligned} \underset{\bar{n}}{\text{minimize}} \quad & E(\bar{n}) \\ \text{subject to} \quad & n_k \geqslant 1;\ k = 1, \ldots, K, \\ & \sum_{k=1}^{K} n_k < N. \end{aligned} \tag{3.2}$$

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 $\mathcal{C}$. (3.3) shows the formulation of the non-overlap constraint, where the start time

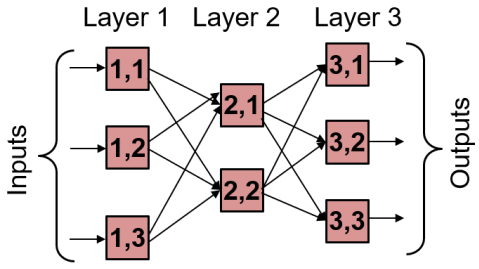
of two tasks is separated by the duration. Furthermore, the start time of all tasks are integers and greater than zero. We add one terminal task with the constraint that the start time of the terminal task ($t_{terminal}$) is greater than the start time of any of the source-destination pairs. We minimize $t_{terminal}$ to obtain the optimal schedule for all source-destination pairs.

$$\begin{aligned} \text{minimize} \quad & t_{terminal} \\ \text{subject to} \quad & t_{mn} > t_{pq} + n_{pq} \vee t_{pq} > t_{mn} + n_{pq}, \\ & \forall t_{mn}, t_{pq} \in \mathcal{C}, \\ & t_{xy} \geqslant 0, \forall\, t_{xy} \in \mathcal{T} \\ & t_{terminal} > t_{xy} + n_{xy}, \forall\, t_{xy} \in \mathcal{T}. \end{aligned} \tag{3.3}$$

3.3 Latency-aware NoC optimization

Determining Minimum Communication Latency

We aim to construct an NoC architecture, customized for different DNNs,



(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 3.5: (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.

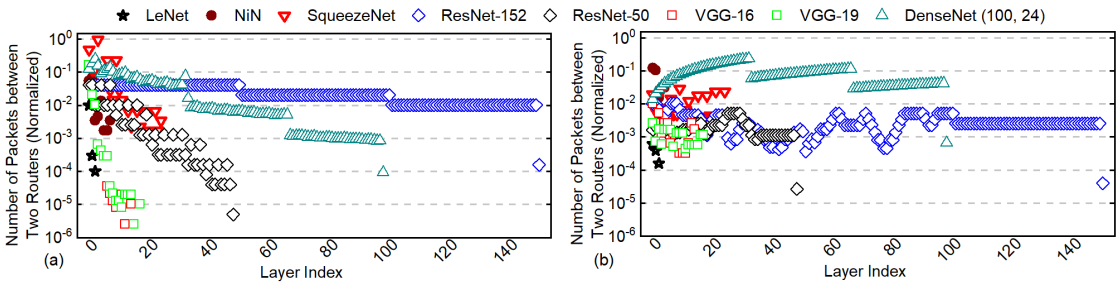


Figure 3.6: The number of packets between two routers with (a) one router per tile, and (b) the proposed technique. All the numbers are normalized with a factor of 4×10^5 (the highest number of packets per router between two layers with one router per tile, which occurs between 4^{th} and 5^{th} layer 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.

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 3.5(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 3.5(a)), then the minimum possible communication latency is achieved. We utilize below key insights to construct the schedules for obtaining minimum 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 3.5(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 with a large number of routers per layer. For example, with this NoC topology, more than 1.5×10^4 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.

Table 3.1: Summary of the notations used in this work.

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}

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 3.6(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 (4×10^5), 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 (3.4). 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.

$$L(\bar{\mathbf{N}}) = \sum_{k=1}^{K-1} \left(\left\lceil \frac{A_k}{N_k N_{k+1}} \frac{Q}{W} \right\rceil \right) \max(N_k, N_{k+1}) \tag{3.4}$$

$$\begin{aligned} \underset{\bar{\mathbf{N}}}{\text{minimize}} \quad & L(\bar{\mathbf{N}}) \\ \text{subject to} \quad & N_k > 0; k = 1, \ldots, K, \\ & \sum_{k=1}^{K} N_k < N. \end{aligned} \tag{3.5}$$

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 3.3, 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 (3.4)). 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 (4.4). 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 3.5(a) is achieved. For each round of commu-

nication, 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 3.4. For the configuration shown in Figure 3.5(a), $N_1 = 3, N_2 = 2, N_3 = 3$. Putting this in Equation 3.4 we obtain, $L(\tilde{\mathbf{N}}) = \max(3,2) + \max(2,3) = 3 + 3 = 6$. Therefore, minimum communication latency for this configuration is 6 cycles, which supports the schedules shown in Figure 3.5(b).

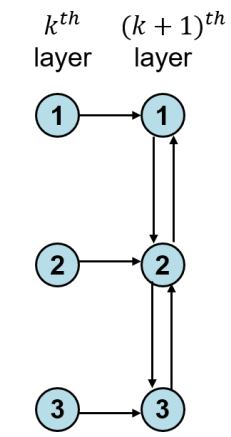
Figure 3.6(b) shows the number of normalized packets between two routers for different DNNs with our proposed tile-to-router mapping methodology. We observe that with the proposed mapping methodology, the number of packets between two routers for different layers of different DNNs is always less than 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 3.3. 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 3.7(a) shows our proposed NoC architecture and Figure 3.7(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 3.1. 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 3.7(a). First three rows in Figure 3.7(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 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



(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 3.7: (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.

that if any of the links shown in Figure 3.7(a) is removed, then some of the transactions will not be possible. Therefore, the NoC architecture in Figure 3.7(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 3.8 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 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

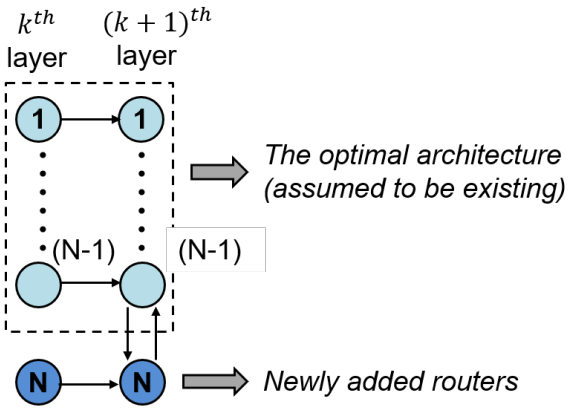


Figure 3.8: NoC architecture which achieves the minimum possible latency when two consecutive layers consist of N routers each.

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 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 3.2 shows the schedules for this case.

Table 3.2: 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	$L_{(k+1),n}-$ $L_{(k+1),(n+1)}$	$R_{k,(n-m+2)}-$ $R_{(k+1),(n+1)}$	$n = 1 \dots (N-1)$

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 3.12(a).

Table 3.3 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 3.9(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 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 3.9(a) completes all transactions in minimum possible cycles and therefore it is optimal.

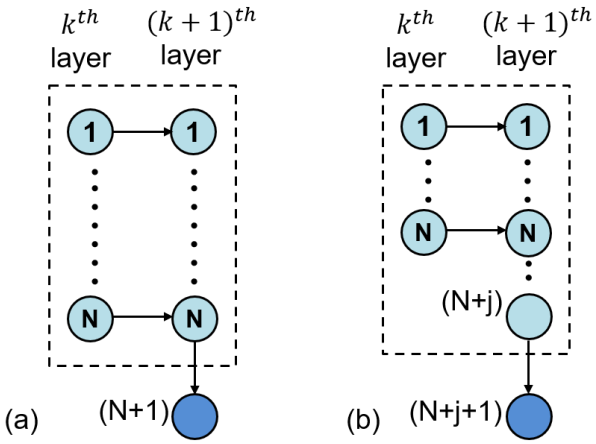


Figure 3.9: NoC architecture to achieve minimum latency for Case 2. (a) shows the case when there is one more router in $(k+1)^{th}$ layer than k^{th} layer. (b) shows the general case. The dotted box shows the optimal architecture (already proved) and the circles filled with dark color represent the newly added router.

Next, we assume that the architecture with $N_{k+1} = N + j$ is optimal as shown in the dotted box in Figure 3.9(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 3.9(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 3.12(b). Ta-

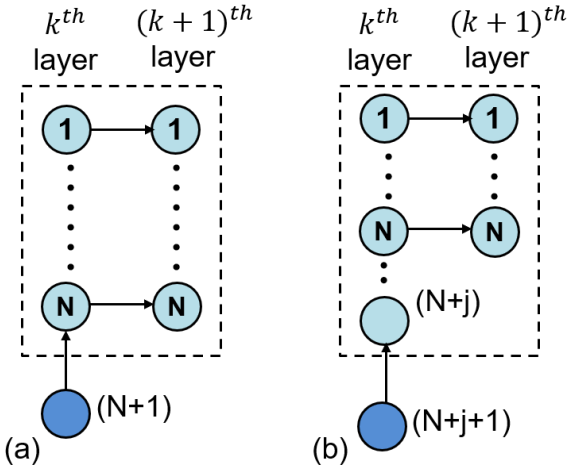


Figure 3.10: NoC architecture to achieve minimum latency for Case 3. (a) shows the case when there is one more router in $(k+1)^{th}$ layer than k^{th} layer, (b) shows the general case. The dotted box shows the optimal architecture (already proved) and the circles filled with dark color represent the newly added router.

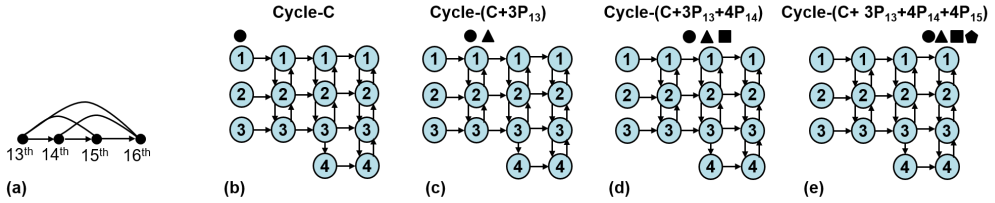


Figure 3.11: Operation of the proposed NoC for a section of DenseNet (100,24) [82]. (a) A representative section of DenseNet (100,24). (b)–(d) show the communication between the layers of DenseNet (100,24).

ble 3.4 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. 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-

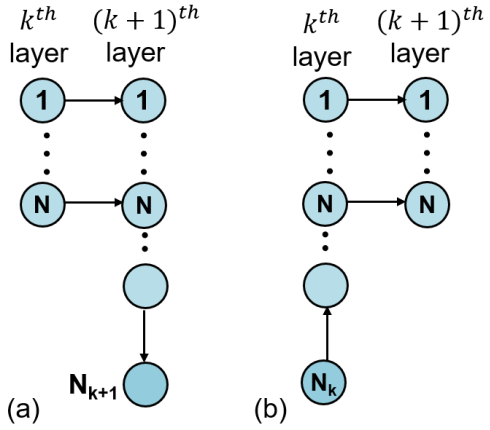


Figure 3.12: NoC architecture which achieves minimum possible latency when (a) $N_k < N_{k+1}$ and (b) $N_k > N_{k+1}$.

Table 3.3: 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	$L_{(k+1),n}-$ $L_{(k+1),(n+1)}$	$R_{k,(n-m+2)}-$ $R_{(k+1),(n+1)}$	$n = 1 \dots N_{k+1} - 1$

ture for densely connected DNNs such as DenseNet [82].

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 3.10(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 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 3.10(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 3.10(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 3.10(b) completes all transactions in minimum possible cycles and therefore it is optimal.

Execution of the proposed network on dense neural network: Figure 3.11

Table 3.4: Schedules for $N_k > N_{k+1}$

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

shows the operation of the proposed NoC on DenseNet [82]. 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 3.11(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 3.11(b). Different packets generated in different layers are shown in different markers in Figure 3.11(b)–3.11(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+3P_{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+3P_{13}+4P_{14}). After that, the computations are 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.

Algorithm 1: 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 3.4 and Equation 4.4
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 3.2
            $sched_{out} \leftarrow [sched_{out}; sched_p]$
        end
        if $N_k < N_{k+1}$ **then**
            $sched_p$ = schedule constructed by following Table 3.3
            $sched_{out} \leftarrow [sched_{out}; sched_p]$
        end
        if $N_k > N_{k+1}$ **then**
            $sched_p$ = schedule constructed by following Table 3.4
            $sched_{out} \leftarrow [sched_{out}; sched_p]$
        end
    end
end

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 3.3. 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 1 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 3.3 (shown in line 4 of Algorithm 1). 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 1). 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 earlier. Depending on the case, the schedules are constructed following Table 3.2 or Table 3.3 or Table 3.4. Second, the constructed schedule is then appended to the list of optimal layer-wise schedules ($sched_{opt}$) (shown in line 10–20 of Algorithm 1). The same procedure is repeated for all layers to obtain the complete schedule of the reconfigurable NoC.

Router Architecture of the Proposed NoC

Figure 3.13 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

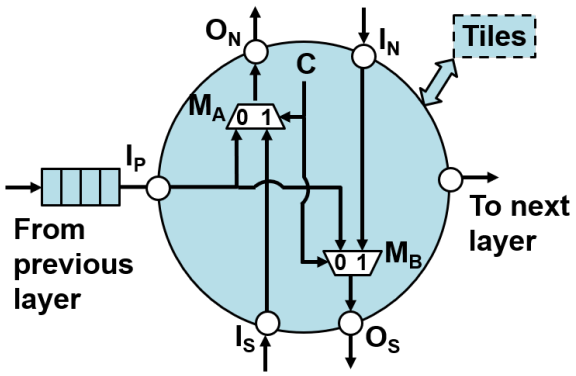


Figure 3.13: Router architecture of the proposed NoC

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 earlier in the section, 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 3.13. 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.

3.4 Experimental Evaluation

Experimental setup

We evaluate our proposed latency-optimized NoC for IMC architecture for a wide range of DNNs. We consider LeNet [120] on MNIST dataset, NiN [129], ResNet-152 [75], VGG-16, VGG-19 [186], and DenseNet(100,24) [82] on CIFAR-100 dataset [115], and SqueezeNet [84] and ResNet-50 [75] on ImageNet dataset [54]. 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 [41] and BookSim [90], 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 [183] 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 by combining the circuit-level and interconnect-level performance. The details of the simulator is described in [110]. Later, the simulator is extended for chiplet-based systems [114, 113].

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 3.14(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

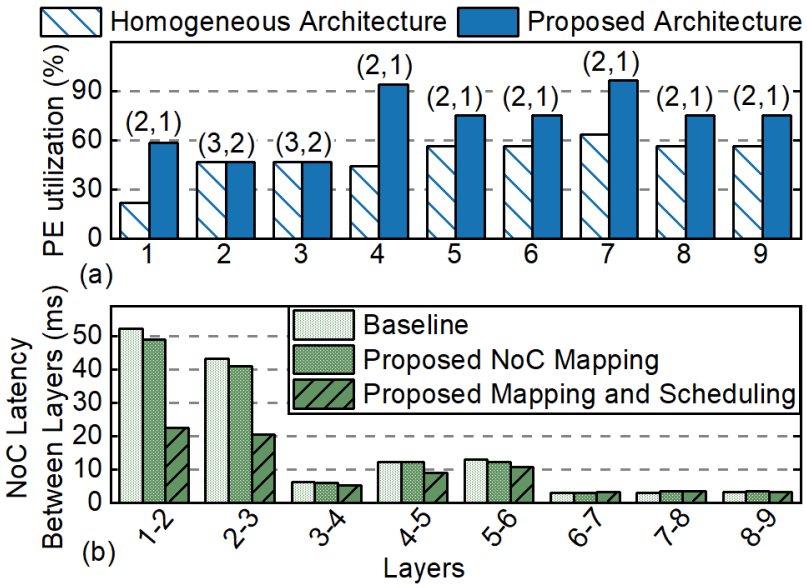


Figure 3.14: Layer-wise improvement for NiN in (a) PE utilization for each layer with SRAM-based heterogeneous tile architecture. The tile structure for each layer (c_k, p_k) is shown on top of each bar and (b) communication latency for each layer with proposed NoC optimization.

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 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 3.15(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.

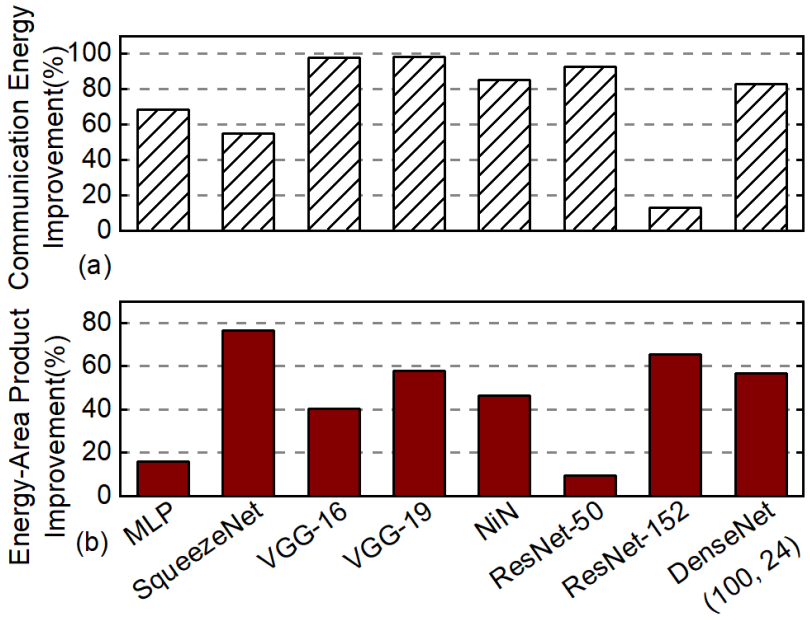


Figure 3.15: Improvement in (a) communication energy of the proposed energy-aware NoC optimization with respect to the baseline (SRAM) and (b) energy-area product of the generated SRAM-based architecture with respect to the baseline (SRAM).

Baseline Architecture: We utilize a crossbar-based multi-tiled IMC architecture to evaluate our proposed approach. Analog MAC computation is 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 [41]. We consider 32nm technology node [183], 1GHz frequency of operation, and a parallel read-out for the crossbar [188]. A mesh-based NoC with bus width of 32 bits and one router-per-tile is considered for the interconnect for the baseline architecture.

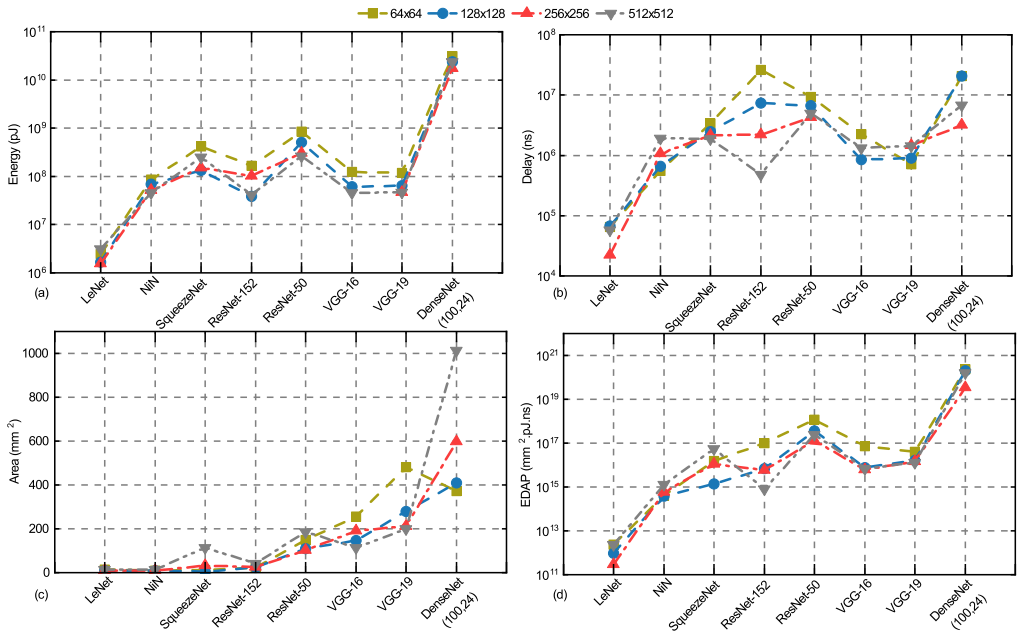


Figure 3.16: Performance of the baseline SRAM-based IMC architecture for different DNNs with different crossbar size. We observe that a crossbar size of 128×128 or 256×256 performs better (with mesh-NoC) than other crossbar sizes.

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 [41]. Figure 3.16 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

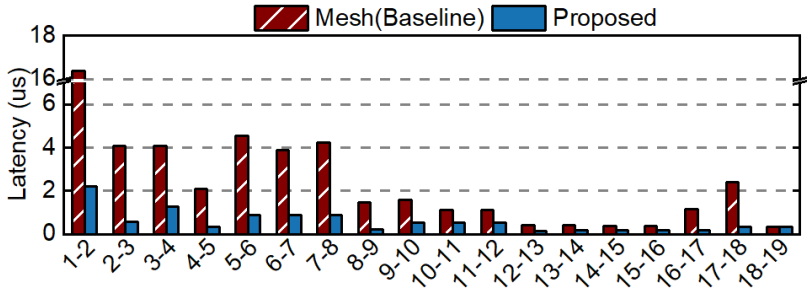


Figure 3.17: Improvement in communication latency for each layer of VGG-19.

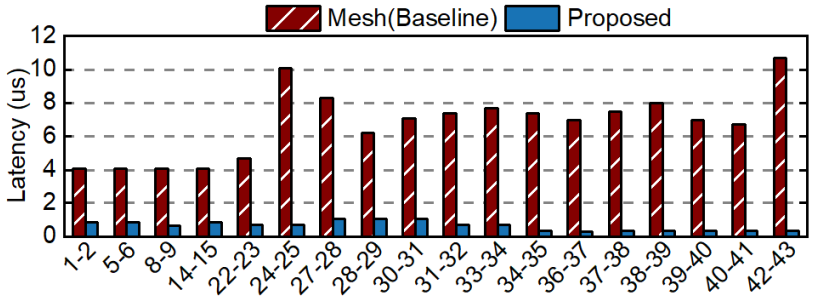


Figure 3.18: Improvement in communication latency for each layer of ResNet-50.

the right end). Figure 3.16(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 3.16(a), 3.16(b), and 3.16(c) respectively. Figure 3.16(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

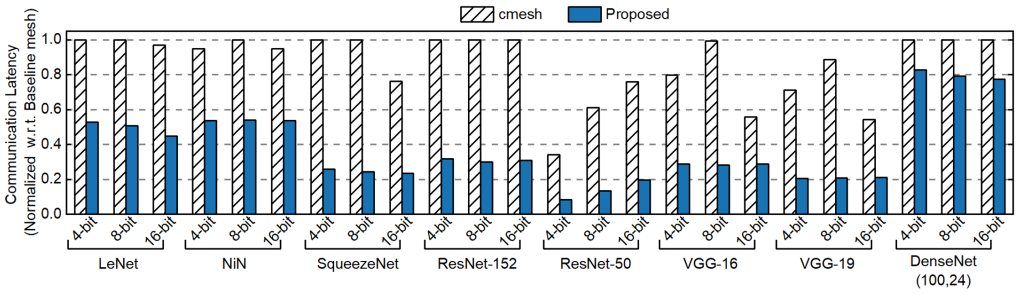


Figure 3.19: Improvement in communication latency for different DNNs (with crossbar size of 256×256) and weights and activation precision with respect to mesh-NoC for cmesh [183] and the proposed approach.

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 3.17 compares 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 3.18 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

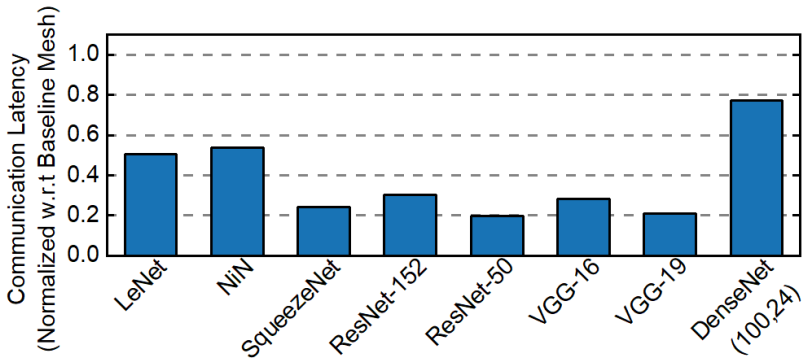


Figure 3.20: Improvement in communication latency with proposed NoC with respect to mesh for crossbar size of 128×128.

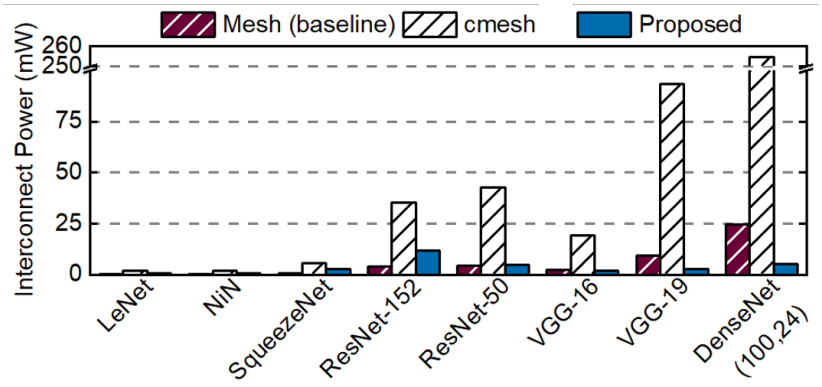


Figure 3.21: Comparison of interconnect power consumption with different techniques.

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 3.3.

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

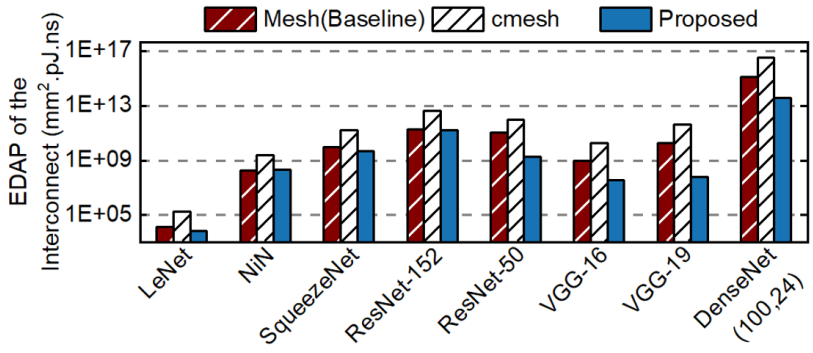


Figure 3.22: Interconnect EDAP comparison for different DNNs.

of the proposed NoC with the cmesh interconnect proposed in [183]. Figure 3.19 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 3.19. 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 3.20, 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 3.21 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 3.22 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

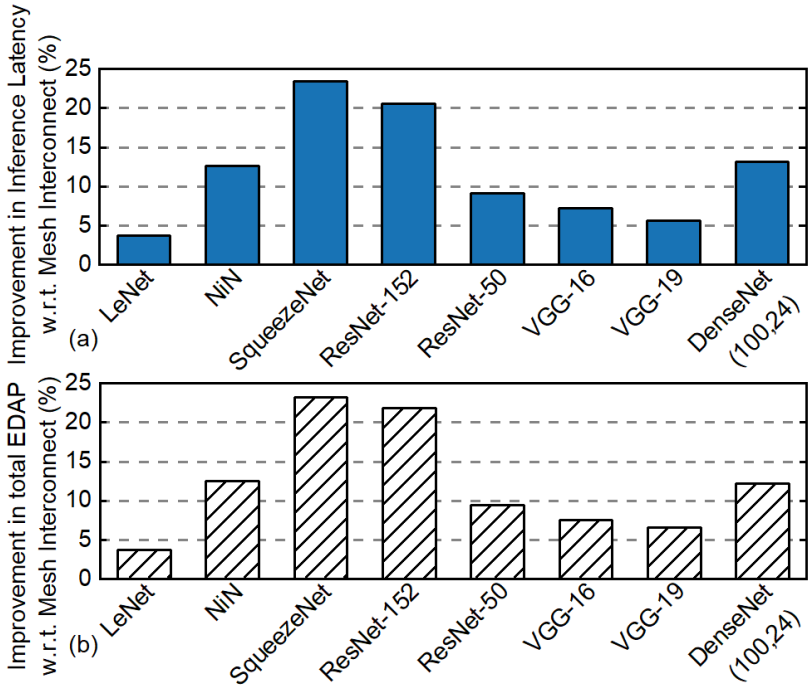


Figure 3.23: Overall improvement in (a) total inference latency and (b) total EDAP of a SRAM-based IMC architecture with the proposed latency-optimized interconnect with respect to the baseline.

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 3.23(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 mesh-NoC over [188, 175] is shown in [111]. 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 [188, 147].

Figure 3.23(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 interconnect 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

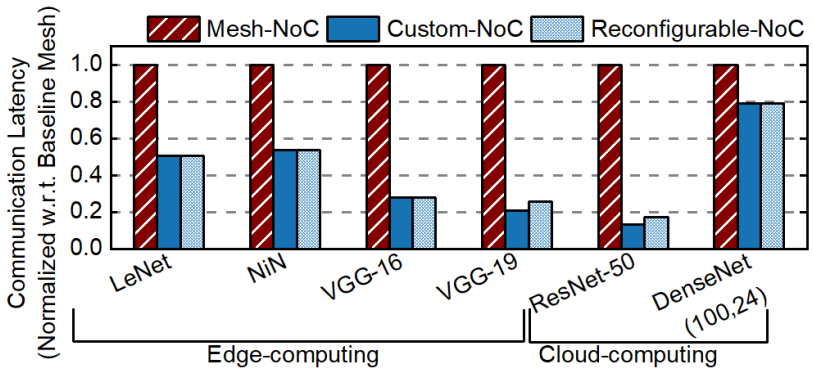


Figure 3.24: Results of leave-one-out experiments with reconfigurable NoC for edge computing- and cloud computing-based DNNs.

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 1 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 3.24. 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 3.24. 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.

4 COMMUNICATION-AWARE HARDWARE ACCELERATORS FOR GRAPH CONVOLUTIONAL NETWORKS (GCNS)

4.1 Background and Motivation

Graph convolutional networks (GCNs) have shown tremendous success for various applications, including node classification, social recommendations, and link predictions [58, 217, 47]. Their powerful learning capabilities on graphs have attracted attention to additional research areas like image processing and job scheduling [204, 146]. Consequently, leading technology companies, including Google and Facebook, have developed libraries and computing systems for GCNs [123, 1], stimulating further research on joint hardware and algorithm optimization.

GCNs operate on graphs by preserving their interconnections. 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 patterns, 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 operation. 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 [46]. Hence, the state-of-the-art GCN models are large and complex [68, 74, 105]. Multiple software-based techniques have been proposed to reduce the computation by utilizing the sparsity of the graph [195, 131]. However, GCN execution still suffers from high latency

and energy consumption.

The prevalence and computational complexity of GCNs call for high-performance and energy-efficient hardware accelerators. In contrast to software implementations, hardware accelerators perform GCN computations with significantly lower latency and higher energy efficiency [213, 64, 10]. Due to this potential, a couple of recent studies proposed GCN accelerators [213, 128]. 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 a substantial number of 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 off-chip memory access consumes on average 1,000$\times$ more energy than computation [78]. Therefore, there is a critical 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 accesse [188]. 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 on-chip communication energy also increases exorbitantly. We implemented an IMC-based GCN accelerator baseline for popular benchmarks to quantify this effect. 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. GCNs consist of thousands of nodes. The connections between the nodes enable message passing. The message passing between the nodes result in high communication volume

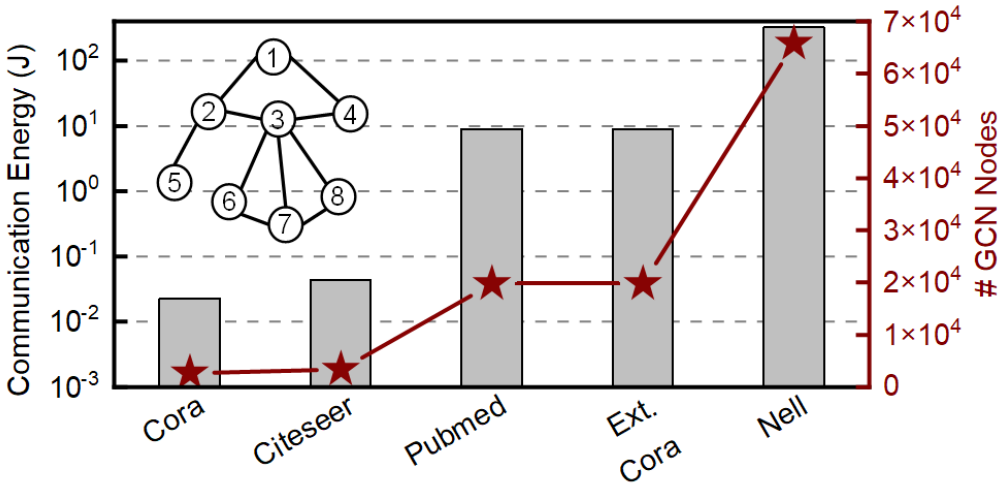


Figure 4.1: Communication energy with a baseline IMC-based GCN accelerator. In the baseline architecture, the number of compute elements is equal to the number of GCN nodes and compute elements are interconnected by a 2D mesh NoC through a dedicated router. The x-axis is sorted by increasing number of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

for GCN accelerators. For example, the Nell dataset with 65755 nodes results in up to 2.7 TB of data communicated between nodes. The high volume of communication data increases communication energy consumption. Figure 4.1 shows that the communication energy increases with the number of GCN nodes. Furthermore, larger GCNs require more compute elements and routers, leading to increased chip area. *Therefore, designing an efficient on-chip communication architecture is crucial for the in-memory acceleration of GCNs.*

This work proposes a communication-aware in-memory computing architecture (COIN) for GCN hardware acceleration. The COIN architecture distributes the GCN computations into multiple compute elements called CEs. Each CE utilizes RRAM-based crossbars for computation, 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 consumption of communication. We show that the objective function is convex. Then, we minimize the objective function to obtain the number of CEs. Note that the proposed methodology is also applicable to SRAM-based IMC.

The major contributions of this work are as follows:

- A novel RRAM-based IMC architecture, COIN, for GCN acceleration that utilizes a communication-aware IMC architecture and a novel dataflow,
- An methodology to determine the optimal number of compute elements (CEs) in COIN that ensures a balance between intra-CE and inter-CE data communication for GCN acceleration,
- Experimental evaluation across popular graph datasets for GCN and comparison with respect to state-of-the-art GPUs and accelerator. COIN achieves up to 105× lower energy consumption with respect to state-of-the-art GCN accelerator.

Graph processing accelerators have recently attracted attention due to their significant impact potential. GraphR [189] graph processing accelerator uses two components: memory and graph engine, both based on resistive random access memory. It performs the graph computations in matrix format without optimizing sparsity. The technique shows around 16× speedup compared to CPU baseline systems. A more recent graph processing accelerator, GraphS [8], uses spin-orbit torque magnetic random access memory (SOT-MRAM) for parallel computations to accelerate graph processing applications. It achieves around 5× speedup compared to processing depending on DRAM acceleration. GCNs involve convolution operations in addition to graph processing. *Therefore, accelerators that target only graph processing are not suitable for GCNs.*

A few recent studies propose GCN hardware accelerator architectures [213, 64]. For example, HyGCN [213] uses a hybrid system to incorporate convolution operation and tackle the irregularity of the GCN structures. It first divides the computations into two as aggregation and combination to exploit different levels of parallelism. Then, a task scheduler is used to exploit edge-level parallelism by sending edge processing loads onto single instruction multiple data cores. The combination phase performs the transformation process by utilizing a systolic-array structure. Main memory accesses take up a significant portion of the total execution time, although the HyGCN employs multiple optimizations to reduce DRAM accesses. Similarly, the GRIP [104] architecture also divides the GCN computations into aggregation and combination engines. It employs a parallel prefetch-and-reduce engine to handle irregular data for aggregation. Another recent technique, EnGN [128], uses a ring-edge-reduce-based approach for data transfer. This approach sends the output data to the subsequent processing unit in a physical ring system for the aggregation phase. However, most of the execution time comes from DRAM accesses as EnGN utilizes the main memory to load the weights to process. Similarly, Rubik [42] uses graph reordering, mapping-aware data reuse to achieve a better graph-level data locality. It uses a customized cache design for graph-level data reuse [3]. Another recent proposal, AWB-GCN, stores the adjacency matrix and the weights on off-chip memory [64]. The sparse matrix multiplication kernel periodically accesses the off-chip memory and performs the computation. It requires up to 503 GBps off-chip memory bandwidth to fully utilize the hardware. *A critical drawback of prior HW accelerators is large number of off-chip memory accesses, which increase the latency and energy consumption.*

IMC-based hardware accelerators reduce off-chip memory accesses by performing computation inside the memory element. Thus, RRAM and SRAM-based IMC accelerators have been proposed for DNNs in the

literature [175, 188]. However, IMC increases on-chip data volume, which increases latency and energy due to on-chip communication [141, 112, 114, 111]. The high density and complexity of GCNs make the on-chip communication for IMC-based accelerators even more critical. Authors in [207, 38] proposed an IMC-based accelerator for GCN. However, these technique does not address the issue of the on-chip communication performance of GCN accelerators. Recently, a RRAM-based 3D NoC-enabled accelerator for GNN training ReGraphX is proposed [10]. The authors show that the proposed architecture is more energy-efficient than conventional GPUs. More detailed survey of communication-aware IMC-based accelerators can be found in [140, 92].

To address the limitations of prior approaches, we propose a communication aware in-memory computing-based accelerator for GCNs. We first identify that the large data volume in GCN results in high latency and high energy consumption due to on-chip communication. Therefore, we co-optimize the communication energy and latency. We determine the optimal IMC architecture for GCN acceleration, COIN through this optimization. The communication-aware interconnect, and IMC-based computing elements significantly improve overall latency and energy for GCN acceleration. To the best of our knowledge, This is the first communication-aware in-memory computing-based GCN accelerator.

Background on Graph Convolutional Networks

An increasing amount of data is now represented in the form of graphs. Deep learning is effective at capturing the patterns in the Euclidean space, but the inherent irregularity of graphs makes them unsuitable for classical deep learning techniques. This limitation has led to advancements in GCNs, whose structure and operations are illustrated in Figure 4.2. GCNs maintain the graph information and can be considered as a generalized version of regular convolutional networks.

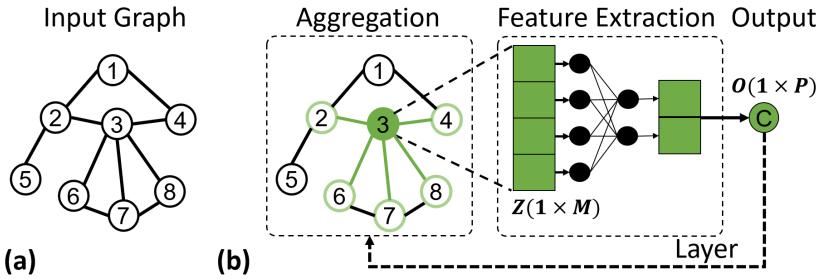


Figure 4.2: (a) An example input graph, (b) Graph Convolutional Network model.

GCN computations are divided into two stages. First, each node aggregates the feature information from all neighbors (node 2, 4, 6, 7, 8 in Figure 4.2(a)) with its own data (node 3) during the *aggregation stage*. For example, the Z matrix in Figure 4.2(b) represents the aggregated node features from node 3 and its neighbors shown in Figure 4.2(a). As the aggregation is done by summation or averaging, the output of the aggregation stage preserves the feature dimensions, M $\times$ 1, where M is the number of input features. The aggregation stage can also consider a weighted average of neighbors' features using their node degrees [105]. For example, the GCN can put more weights on the neighbor nodes with lower degrees to reduce the impact of high-degree nodes.

The second stage of GCN is the *feature extraction stage*. It is similar to regular convolutional neural network computations. The result of the aggregation stage (Z in Figure 4.2(b)) is fed into a multi-layer perceptron (MLP) based model. After that, an activation function like ReLU is applied. Finally, the O matrix (1 $\times$ P, where P represents the number of outputs in Figure 4.2(b)), is produced as the output of the feature extraction stage. These two stages repeat iteratively, where the number of layers determines the farthest distance a node feature can travel. For example, for a GCN with a single layer, each node gets information from only its neighbors. A typical GCN uses 2–3 layers [105].

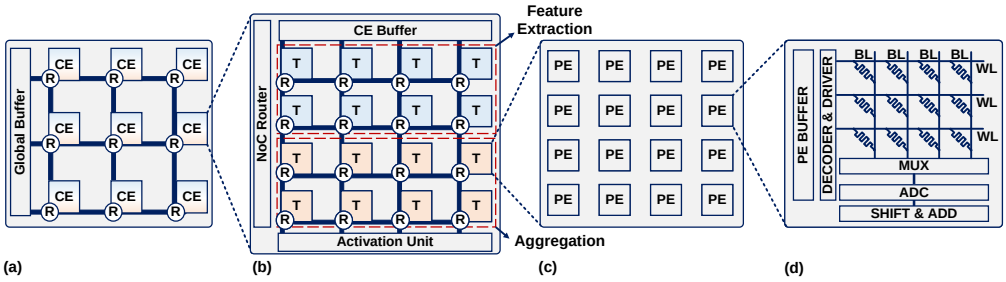


Figure 4.3: Overview of the COIN architecture for GCN acceleration. Each compute element (CE) consists of an array of processing elements (PEs) or RRAM-based IMC crossbar arrays connected by an NoC-mesh. A subset of the PEs performs the aggregation operation while the remaining performs the feature extraction. Aggregation PEs store the adjacency matrix while the feature extraction PEs store the layer weights.

IMC-based graph convolutional networks can suffer from significant on-chip communication overhead, especially when they have large graph sizes. For example, the NELL dataset [33] has 65,755 nodes, resulting in an adjacency matrix of size 65,755×65,755. The large adjacency matrix and corresponding computations incur a significant on-chip communication overhead. Hence, there is an urgent need for an optimized IMC architecture that exploits the parallelism within the GCN operations while considering the on-chip communication overhead.

4.2 The Proposed COIN Architecture

Finding the Number of Compute Elements (CEs)

IMC-based architecture reduces off-chip memory accesses at the expense of increased on-chip communication volume, leading to higher communication energy consumption and latency. As illustrated in Figure 4.1, on-chip communication itself consumes 320J of energy for the Nell dataset

with a baseline architecture. Both the baseline and proposed COIN architecture perform the GCN computations in dedicated IMC compute elements. Furthermore, in the baseline design each IMC element is connected with a dedicated router in a 2D mesh NoC. As a result, energy consumption and communication latency of the baseline design can be prohibitive for GCNs that process large datasets, such as the Nell dataset with 65755 nodes. Therefore, there is a need to optimize the number of CEs to minimize the on-chip communication volume, thus the energy consumption and latency.

This section presents an optimization technique to determine the optimal number of CEs by considering both intra- and inter-CE communication volume. First, we show a canonical example illustrating intra- and inter-CE communication volume. Then, we construct an objective function that captures total communication energy (both intra- and inter-CE) for a GCN. Finally, we employ the interior point algorithm to minimize the objective function obtaining the optimal number of CEs.

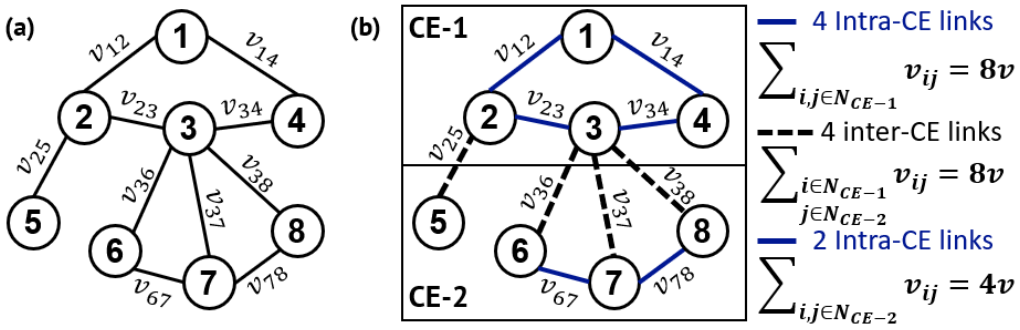


Figure 4.4: A canonical graph example for intra-CE and inter-CE communication.

An illustrative canonical example

We take a canonical example to illustrate the communication between two CEs (inter-CE) as well inside a CE (intra-CE). Figure 4.4(a) shows a graph with 8 nodes. Each node has a convolutional network with L hidden layers embedded to it. The output of the convolutional network after each layer at each node is communicated to the nodes located at the neighbor. Therefore, the communication between neighbors generates a communication volume. The bidirectional communication volume between node-i to node-j is represented as v_{ij}.

Let us assume, there are two CEs (CE-1 and CE-2) in the hardware. Let us also assume that node 1, 2, 3, and 4 are mapped to CE-1 and node 5, 6, 7, and 8 are mapped to CE-2 as shown in Figure 4.4(b). For the sake of simplicity, we consider v_{ij} are all equal ($v_{ij} = v,\ \forall i, j$). The connections inside each CE are shown in blue solid lines. Since there are total 4 connections inside CE-1, we obtain $8v$ as the intra-CE communication volume for CE-1. Similarly, 2 connections inside CE-2 makes communication volume as $4v$. The graph in Figure 4.4(b) has four connections between CE-1 and CE-2: 2–5, 3–6, 3–7, 3–8 shown in dashed black lines. Therefore, the total communication volume is $8v$ between CE-1 and CE-2 (intra-CE communication).

Constructing the objective function

Suppose the target GCN has N nodes each implementing a convolutional network with L layers, where $L, N \in \mathbb{Z}^+$. We denote the number of input activation bits of layer l as $a(l)$ for $1 \leqslant l \leqslant L$. We note that, the communication volume between nodes (v_{ij}) appears due to the activation bits. The number of output activation bits of layer l can be expressed as $a(l+1) \in \mathbb{Z}^+$ since it equals to the number of input activation bits of the succeeding layer. Finally, let the number of CEs be denoted by the variable

$k \in \mathbb{Z}^+$. Therefore, number of nodes mapped onto each CE is $\frac{N}{k}$. The objective function has two components which are described next.

Intra-CE Communication Energy: The first part of the objective function accounts for the intra-CE communication energy. The number of CEs dictates the communication volume inside each CE. Remember that the number of input activation bits to each node for l^{th} layer is $a(l)$. Since we consider sparse connections between nodes, we denote the probability of having a connection between two nodes in CE-m as $p_m^{(1)}$. Since there are $\frac{N}{k}$ number of nodes mapped to each CE, there are total $\sum_{m=1}^{k}(\frac{N}{k})(\frac{N}{k}-1)p_m^{(1)}$ transactions between all nodes inside a CE. Hence, the total number of output activation bits within the CE after the l^{th} layer of operation is $\sum_{m=1}^{k}(\frac{N}{k})(\frac{N}{k}-1)p_m^{(1)}a(l+1)$. We add the whole expression for $L-1$ layers to take account of the output activations for each layer. Finally, assuming energy per bit is proportional to square root of number of nodes per CE, we obtain the total intra-CE communication energy as follows:

$$E_{intra-CE}(k) = \sum_{m=1}^{k} \frac{N}{k}\Big(\frac{N}{k}-1\Big)p_m^{(1)} \sum_{l=1}^{L-1} a(l+1)k\Big(\frac{N}{k}\Big)^{\frac{1}{2}} \tag{4.1}$$

Inter-CE Communication Energy: The second part of the objective function accounts for the inter-CE communication energy. The number of CEs in the system is a key contributor to inter-CE communication volume. As a reminder, each CE implements the functionality of $\frac{N}{k}$ nodes of the target GCN. We denote the probability of having a connection between two nodes mapped in CE-i and CE-j as $p_{ij}^{(2)}$. Since each node generates $a(l+1)$ output activation bits after processing the l^{th} layer between CE-i and CE-j, the number of output activation bits generated is $\frac{N}{k}\frac{N}{k}a(l+1)p_{ij}^{(2)}$. We note that, all CEs generate output activation bits to $(k-1)$ other CEs. Therefore, the total inter-CE communication volume is obtained by adding the summations to take account of all CE pairs. Finally, assuming that

the energy per bit for inter-CE communication is proportional to square root of number of CEs ($k^{\frac{1}{2}}$) [90], we obtain the total inter-CE energy by multiplying the whole expression by $k^{\frac{1}{2}}$ as follows:

$$E_{inter-CE}(k) = \sum_{i=1}^{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} \left(\frac{N}{k}\right)\left(\frac{N}{k}\right) p_{ij}^{(2)} \left(\sum_{l=1}^{L-1} a(l+1) \right) k^{\frac{1}{2}} \tag{4.2}$$

Finally, we obtain the total communication energy by adding intra-CE (Equation 4.1) and inter-CE (Equation 4.2) communication energy as shown in Equation 4.3.

$$E(k) = E_{intra-CE}(k) + E_{inter-CE}(k) \tag{4.3}$$

Solving the objective function

Our goal is to minimize the objective function $E(k)$ with constraints in Equation 4.4. As a reminder, each CE is connected to a NoC router. Hence, the number of NoC routers is equal to the number of CEs. The constraint in Equation 4.4 states that the number of routers in the NoC (k) is positive and is linear on k. In Appendix A, we show that $E(k)$ is convex. Since $E(k)$ is a convex function with linear constraint, we can apply any standard algorithm to solve a convex optimization problem. In this work, we use the interior point algorithm [98] to solve Equation 4.3 with constraints in Equation 4.4. We use this algorithm since it provides a solution in polynomial time. Specifically, it takes only 10ms to obtain a global minimum. *Based on the result, we consider a 4×4 mesh NoC to connect CEs i.e. total number of CEs in COIN is 16.* With 16 CEs, COIN consists of 30 MB of memory on-chip.

$$\begin{aligned} \underset{k}{\text{minimize}} \quad & E(k) \\ \text{subject to} \quad & k > 0. \end{aligned} \tag{4.4}$$

Proposed Mapping and Dataflow

Mapping of GCN to the RRAM-based IMC crossbar arrays

This section describes the mapping of the adjacency matrix and layer weight matrix onto the RRAM-based IMC crossbar arrays. Since adjacency matrix remains the same for all the layers, we map the adjacency matrix onto the RRAM-based IMC crossbars inside a CE and reuse for all layers. Specifically, if a GCN consists of N nodes and the architecture has k CEs, then the size of the adjacency matrix to be mapped in each CE is $N \times \frac{N}{k}$. We note that the weight matrices are specific to each layer and are smaller in size. Therefore, the total number of IMC crossbar arrays required to map the weight matrices is smaller than the number of IMC crossbar arrays required to map the adjacency matrix. We further note that the mapping of both (adjacency and weight) matrices is performed as is without any matrix transformations.

Dataflow of the COIN architecture

We propose a layer-wise operation for the inference with COIN, as illustrated in Figure 4.5. In the beginning, the adjacency matrix and the weights are loaded into the corresponding IMC crossbars from the off-chip memory. Then, the proposed layer-wise computations within each layer are performed parallelly (simultaneously within each CE), while across layers are executed serially. We denote the input features to layer-i as X_i and weights of layer-i as W_i. The feature extraction unit (IMC crossbar arrays) performs the matrix multiplication between the feature matrix (X_i) and the weight matrix (W_i) to generate the intermediate output Z_i as shown

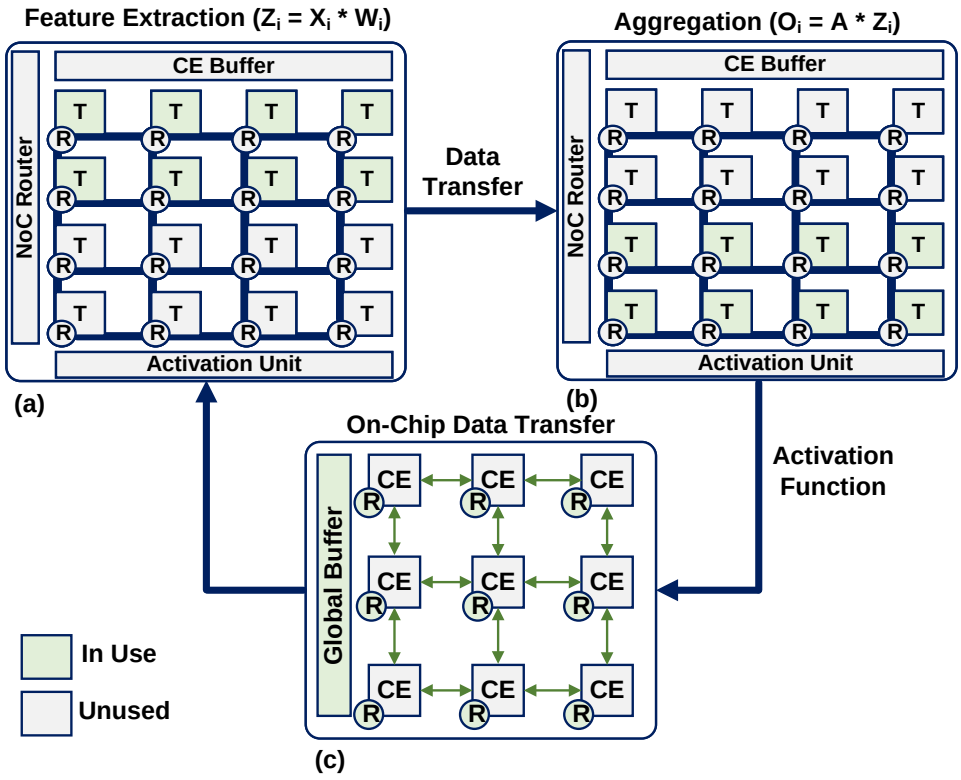


Figure 4.5: Layer-wise execution dataflow of the proposed COIN architecture. (a) shows feature extraction operation of layer-i in a CE, (b) shows aggregation operation of layer-i in a CE, (c) shows the communication between CEs.

in Figure 4.5(a). The weights are stored in the RRAM IMC crossbars in a column-wise manner. The transposed form of the input feature matrix X_i is then provided to the crossbar array as the input vector across the wordlines. Each input gets multiplied with the corresponding weight value stored in the RRAM to generate the output. The computed result then accumulates along the bitline (BL) in the current domain. Next, the analog current is then converted to digital using an analog-to-digital converter (ADC). The

proposed COIN architecture does not utilize a digital-to-analog converter, while it utilizes bit-serial computing for multi-bit inputs (shift and add circuit performs the accumulation based on the positional value of the input).

After that, the aggregation operation is performed. The adjacency matrix is stored in the RRAM IMC crossbar arrays to form the aggregation unit, similar to weights within the feature extraction operation. The input of the aggregation unit is the transposed form of the intermediate output (Z_i) from the feature extraction operation. The aggregation unit performs the matrix multiplication between the intermediate output Z_i and the adjacency matrix (A) to obtain $O_i = A.Z_i$ as shown in Figure 4.5(b). The computation within the aggregation unit (IMC crossbar arrays) is similar to that in the feature extraction step. The 'shift and add' unit inside PE performs the addition on the aggregated output. Next, the ReLU operation is performed on O_i to obtain the output from layer i across all CEs. Finally, the output from layer i is communicated to all CEs via the NoC to perform the computation for layer $i+1$ of the GCN, as shown in Figure 4.5(c).

Illustration on number of multiplication operations

The proposed dataflow helps to reduce the number of multiplication operations and hence the communication between feature extraction and aggregation unit within a CE. For example, let us consider the Nell dataset and the operation of its first layer. The size of the adjacency matrix A is 65755×65755, the size of the feature map is X_1 65755×5414, and the size of the weight matrix is W_1 5414×16. If the aggregation operation is performed first and then feature extraction, the total number multiplication operations are: 65755×65755×5414 (aggregation) + 65755×5414×16 (feature extraction) = $\mathbf{2.3}\times\mathbf{10}^{13}$. At the same time, if feature extraction is performed first and then aggregation (proposed approach), the total number of multiplications performed is given by 65755×5414×16 (feature

Table 4.1: Properties for different GCN datasets

	Cora	**Citeseer**	**Pubmed**	**Ext. Cora**	**Nell**
Dataset Type	Citation Network	Citation Network	Citation Network	Citation Network	Knowled. Graph
# Nodes	2708	3327	19717	19793	65755
# Edges	10556	9228	88651	130622	266144
# Features	1433	3703	500	8710	5414
# Output Labels	7	6	3	70	210
# Layers	2	2	2	2	2

extraction) + 65755×65755×16 (aggregation) = $\mathbf{7.4}\times\mathbf{10}^{10}$. Therefore, there is a 311× reduction in the number of multiplication operations for Nell with our proposed dataflow for COIN. The reduction comes from the fewer multiplication operations in the aggregation stage.

4.3 Experimental Evaluation

This section present the area, latency, and energy consumption evaluations of the proposed COIN architecture. This work assumes 32 nm process technology and 1 GHz operating frequency.

Table 4.2: Summary of circuit level and NoC parameters

Circuit		NoC	
PE array size	128 × 128	Bus width	32
Cell levels	2 bit/cell	Routing algorithm	X–Y
Flash ADC resolution	4 bits	Number of router ports	5
Technology used	RRAM	Topology	Mesh

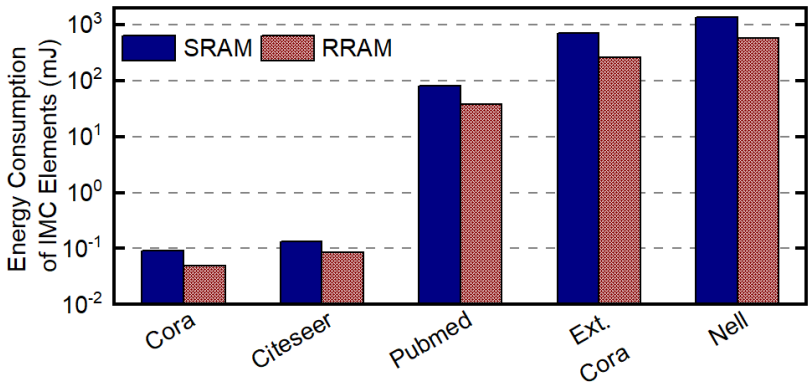


Figure 4.6: Comparison of Energy Consumption (in log scale) of IMC Elements between SRAM and RRAM-based designs.

Experimental Setup

Datasets: We evaluate the COIN architecture using widely used graph datasets: Cora [149], Citeseer [68], Pubmed [105], Nell [33] and Extended Cora [27]. Nell represents a knowledge graph dataset used to learn to read the web, and other datasets are scientific publication citation datasets for classification. Table 4.1 shows the details of these datasets.

Simulation framework: We customized an open-source simulator [110] to incorporate GCN attributes and support the COIN architecture. The inputs to the simulator include the GCN parameters, such as the number of nodes, the graph structure of each node, and input/output features for each layer. In addition, the simulator uses user inputs such as technology node, RRAM-based IMC crossbar size, frequency of operation, NoC size, NoC frequency, and the number of bits per RRAM cell, among others. The performance of the computing elements is measured by customizing NeuroSim [41]. The lower-level components (e.g., buffers, ADC, multiplexers) are simulated using the Predictive Technology Model (PTM) [221], and verified against circuit simulation (e.g., SPICE), reaching more than 90% accuracy. The communication performance is measured through a widely

used cycle-accurate NoC simulator, BookSim [90]. To this end, we developed a customized version of BookSim to evaluate the NoC performance that supports trace-based cycle-accurate simulation. Since different GCNs exhibit different graph structures, we first generate traces for a given GCN. The traces consist of the source router, destination router, and generation timestamps of each packet. Since each layer (also known as the iteration) of the GCN is executed sequentially, we generate a separate trace file for each layer. Then, we feed the traces to BookSim to evaluate the communication performance. Finally, the performance of computation and communication components are combined to obtain the total performance. Table 4.2 summarizes the parameters used in COIN. The simulation framework is publicly available in [180].

Comparison between SRAM and RRAM-based design: Figure 4.6 shows the comparison of energy consumption of IMC elements between SRAM and RRAM-based design across GCNs for different datasets. We note that the energy consumption by communication remains the same irrespective of the type of IMC elements used, since the volume of inter-CE and intra-CE communication do not change with different types of IMC elements. We observe that SRAM-based IMC elements consistently consume more energy than RRAM-based IMC elements. On-average SRAM-based IMC elements consume 2.1$\times$ more energy than RRAM-based IMC elements. Since RRAM-based devices use analog computation, they are more energy-efficient than SRAM-based devices. We note that the energy consumption by communication remains the same irrespective of the type of IMC elements used, since the volume of inter-CE and intra-CE communication do not change with different types of IMC element. Therefore, we consider RRAM-based IMC elements for our architecture.

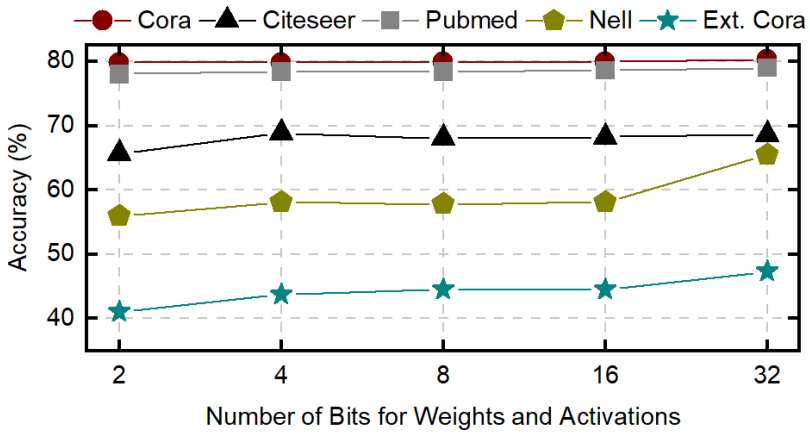


Figure 4.7: Accuracy with different quantization bits for weights and activations for different datasets.

Experiments with Different Quantization Bits

This section evaluates the accuracy of the GCN for different datasets with a varying number of quantization bits for weights and activations. We used the GCN structure described by the authors in [105]. Deep Graph Library (DGL) [206] with PyTorch backend and Nvidia Tesla V100 GPU are during experiments. The accuracy for the Nell dataset increases from 55.9% to 65.4% when the number of quantization bits is increased from 2 to 32, as shown in Figure 4.7. For Extended Cora, the accuracy varies from 41% to 47.3%. For all other datasets, the difference between the minimum and maximum accuracy is less than 3%. In the rest of the evaluations, we consider 4-bit quantization for weights and activations since it provides comparable accuracy with 32-bit precision.

Chip Area Evaluation

The total area of COIN is 17.43 mm^2 with 16 CEs with 30 tiles per CE. Figure 4.8 shows the area of different components of COIN responsible for computation and communication. The components for in-memory

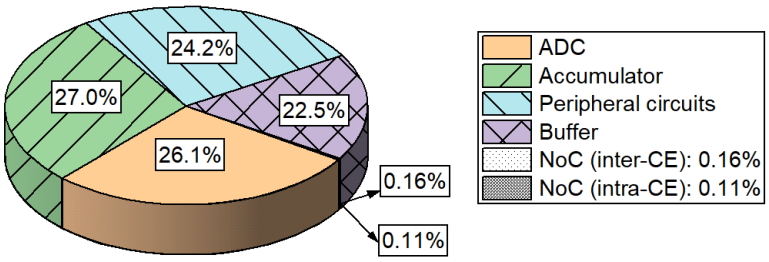


Figure 4.8: Different components of COIN and corresponding area.

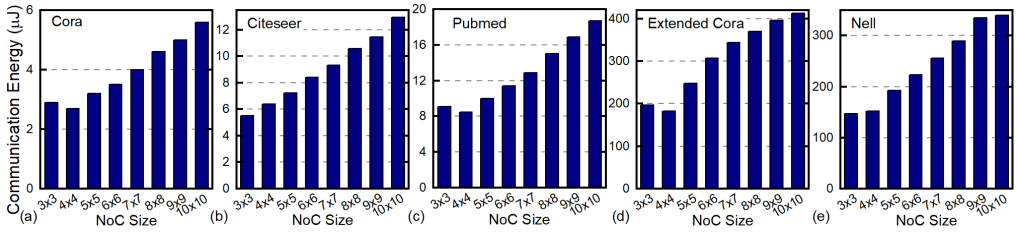


Figure 4.9: Comparison of communication energy consumption with different NoC sizes for (a) Cora, (b) Citeseer, (c) Pubmed, (d) Extended Cora and (e) Nell.

computing include ADC to convert analog multiplication results to digital values, accumulators to perform addition operations (in Tile level), buffers to store intermediate values, and peripheral circuits. We observe that the accumulator occupies 27% of the total area. The NoC for inter-CE and intra-CE communication occupy 0.16% and 0.11% of the total area respectively.

We also note that GCN for large datasets such as Nell or extended Cora require multiple instances of the COIN chip. More precisely, Cora and Citeseer require 1 chip, Pubmed requires 3 chips, extended Cora requires 20 chips, and Nell requires 45 chips. This design choice is widely adopted

for CNN accelerators (e.g. the work proposed in [183] uses up to 48 chips for a single CNN where area of each chip is 86 mm^2.)

Experiments with Different Mesh Sizes

In this section, we compare the communication energy consumption between different NoC sizes for GCNs with different dataset. Figure 4.9 shows the comparison. The NoC size is varied from 3×3 to 10×10. In each case, the number of CEs is equal to the number of NoC routers. We observe that 4×4 NoC (i.e. the design with 16 CEs) consumes least communication energy for most of the dataset. For example, the least communication energy consumption for Cora dataset is 2.7 μJ with 4×4 NoC. The communication energy consumption for the same dataset with 10×10 NoC is 5.6 μJ. Therefore, the results with different mesh sizes show that 4×4 results in the least communication energy consumption for most of the dataset which is aligned with our theoretical results.

Improvement with respect to Baseline

This section compares the performance of our proposed architecture against a baseline design. *We note that a baseline design is used to show the efficacy of the proposed architecture due to the lack of prior work using IMC architectures for GCN acceleration.* In the *baseline* design, computation of each GCN node is performed using an RRAM-based IMC crossbar array, and every node is connected through a router to an NoC. In the baseline architecture, for Cora dataset with 2708 nodes, the computations of all 2708 nodes are performed by individual IMC crossbar arrays, i.e., there are 2708 CEs. Each CE is connected through a router of the NoC. Figure 4.10 compares the total energy consumption of the baseline design and the proposed COIN architecture for different datasets. We observe significant improvement in energy consumption with COIN for all datasets. The

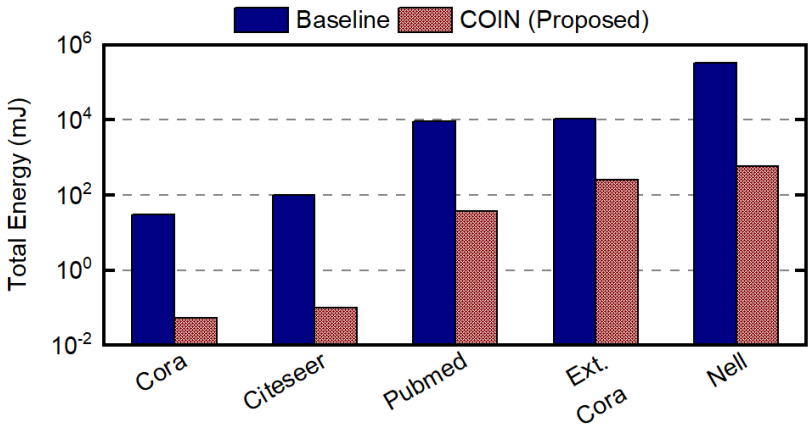


Figure 4.10: Comparison of total energy (in log scale) with respect to a baseline architecture. In the baseline architecture, the number of compute elements is equal to the number of GCN nodes and compute elements are interconnected by a 2D mesh NoC through a dedicated router.

largest improvement ($1100\times$) is obtained for the Citeseer dataset. The GCN for the Nell dataset has the largest energy consumption (with both architectures), since Nell dataset consists of the highest number of nodes. In this case, the baseline design consumes more than 300J energy. However, the proposed COIN architecture reduces the energy consumption to 577 mJ.

We also show the percentage of the total communication energy consumption for both baseline and the proposed architecture in Table 4.3. The communication energy contributes to a significant portion of the total energy with the baseline design. For example, the communication energy makes up 96% and 99% of the total energy for Pubmed and Nell datasets, respectively. With the COIN architecture, the communication energy is 7×10^{-3}% and 6×10^{-4}% of the total energy for Pubmed and Nell datasets. Since Pubmed, Extended Cora, and Nell dataset exhibit higher sparsity than Cora and Citeseer dataset [64], communication energy also contributes lesser (for Pubmed, Extended Cora, and Nell) to the total energy

Table 4.3: Comparison of Percentage Contribution of Communication Energy (%).

Datasets	Cora	Citeseer	Pubmed	Extended Cora	Nell
Baseline	43	44	96	58	99
COIN (Proposed)	4.7	5.3	0.007	0.003	0.0006

with our proposed architecture. The vast improvement in communication energy comes from the proposed optimization.

Improvement in Communication Performance

This section evaluates the improvement in communication performance with COIN for different datasets. We consider the same baseline architecture as Section 4.3. Figure 4.11 shows the comparison of communication energy between the baseline architecture and COIN. Since COIN optimizes communication, there is a substantial improvement in communication energy compared to the baseline architecture. For example, only communication itself consumes 9.2 J of energy to perform one inference in Pubmed with baseline architecture. In contrast, COIN architecture consumes only 0.02 mJ communication energy (5 orders of magnitude improvement). The improvement is the highest for the Nell dataset as expected (6 order of magnitude) since it has the highest number of nodes, hence the largest communication volume. We also show the comparison of communication energy against c-mesh NoC since it has been used for accelerators targeted to CNNs [183]. The comparison is shown in Figure 4.12. In this case, we assume that c-mesh has 16 routers, i.e., the same number of routers as COIN. C-mesh uses additional links and routers, which reduces latency compared to 2D mesh. However, c-mesh has higher energy consumption than COIN since it uses more resources. The largest communication energy saving is observed for the Nell dataset ($1.3\times$) as shown in Figure 4.12.

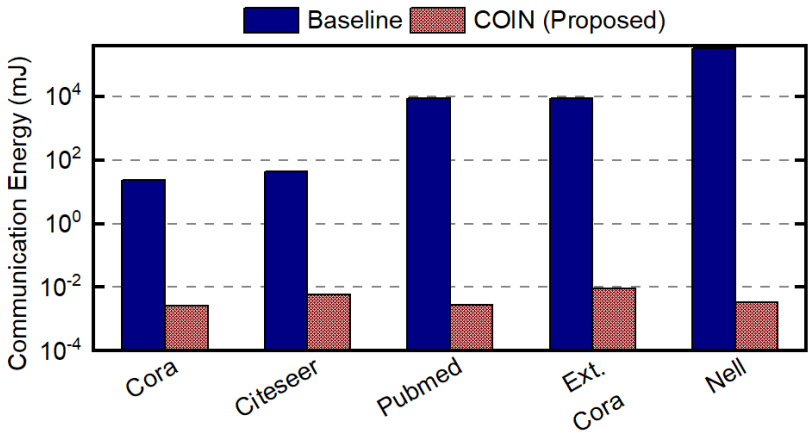


Figure 4.11: Comparison of communication energy (in log scale) between baseline and proposed COIN architecture.

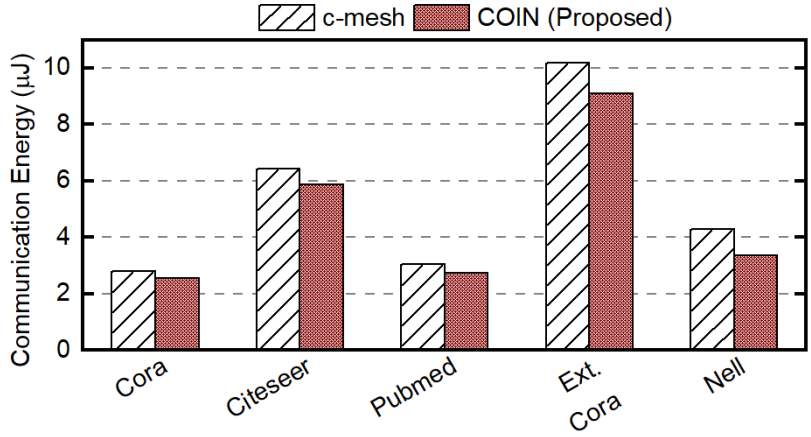


Figure 4.12: Comparison of inter-CE communication energy between the proposed architecture with c-mesh NoC and proposed COIN architecture with mesh NoC.

Overall, COIN significantly reduces energy consumption compared to both the baseline architecture and c-mesh NoC.

Figure 4.13 shows the comparison of energy-delay (EDP) product between the baseline and the proposed COIN architecture. Since COIN shows significant improvement in communication energy and latency

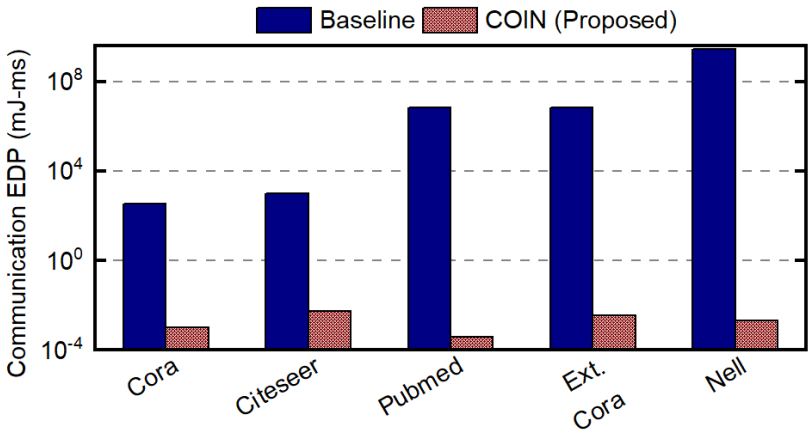


Figure 4.13: Comparison of EDP (in log scale) for on-chip communication between baseline and proposed COIN architecture.

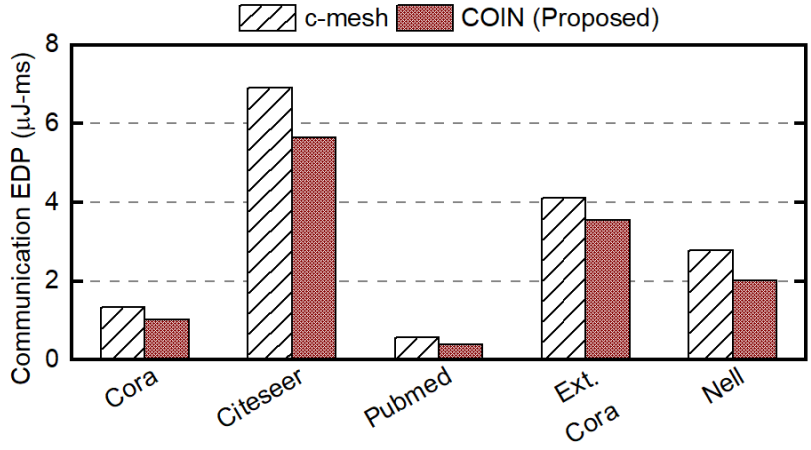


Figure 4.14: Comparison of EDP for inter-CE communication across baseline, proposed architecture with c-mesh NoC, and proposed COIN architecture. COIN achieves least EDP across all datasets.

compared to the baseline design, we also observe considerable EDP savings. For example, the Citeseer dataset shows seven orders of magnitude improvement in communication EDP compared to the baseline design. Similar to the results for energy, we also observe improvement in com-

Table 4.4: Comparison with Nvidia Quadro RTX-8000 GPU.

	Cora			Citeseer			Pubmed			Ext. Cora			Nell		
	RTX	COIN	Impr. (×)	RTX	COIN	Impr. (×)	RTX	COIN	Impr. (×)	RTX	COIN	Impr. (×)	RTX	COIN	Impr. (×)
Energy (mJ)	62.2	0.05	**1244**	90.50	0.10	**905**	89.1	38.13	**2.4**	1787.3	257.4	**6.9**	1504	577.1	**2.6**
Latency (ms)	1.22	0.6	**2**	1.22	1.10	**1.1**	1.22	0.57	**2.1**	7.45	9.96	**0.8**	14.94	1.04	**14.4**
EDP (mJ.ms)	75.78	0.03	**2526**	110.68	0.11	**1006**	108.65	21.56	**5.1**	13309	2564	**5.2**	22423	601.4	**37.3**

Table 4.5: Configuration of the edge devices considered

	# CPU Cores	Max CPU Freq. (GHz)	TOPs	# GPU Tensor Cores	Max GPU Freq. (GHz)
Xavier NX	6	1.4	21	48	1.1
AGX Xavier	8	2.26	32	64	1.37

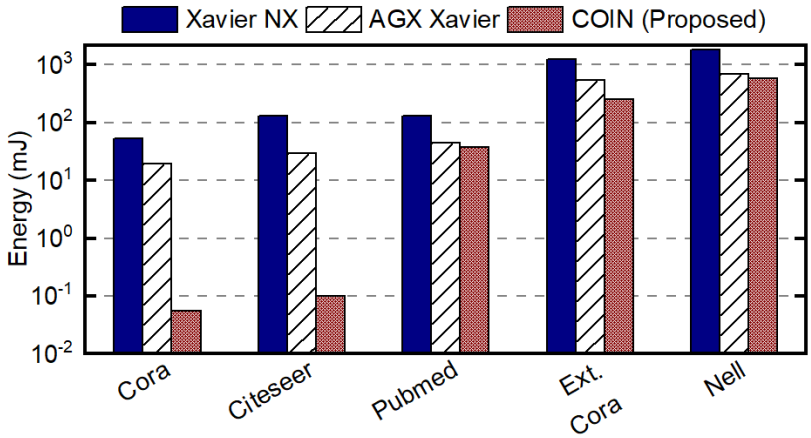


Figure 4.15: Comparison of energy (in log scale) between COIN and edge devices. COIN consumes less energy than both Nvidia Jetson edge devices.

munication EDP with respect to c-mesh as shown in Figure 4.14. The improvement compared to c-mesh is the highest for the Pubmed dataset (30%). In summary, our proposed optimization in NoC enables notable improvement in communication energy and EDP compared to the baseline and a design with c-mesh NoC.

Comparison with GPU and Edge Devices

We perform a detailed comparison of the proposed COIN architecture compared to state-of-the-art GPU – Nvidia Quadro RTX-8000 [2]. The trained GCN model for each dataset is considered, and inference is performed on the RTX-8000 GPU. We perform 2,000 inferences for the trained model and sample the GPU power using Nvidia's system management interface (SMI) program. The power measurements are performed in intervals of one second. Furthermore, we take the average of the measured power values to generate the average power for the 2,000 GCN inferences. The inference latency is then evaluated for the GCN using python's time function. Finally, the inference energy is evaluated by multiplying the average power and the inference latency. We note that the same data precision of 4-bits is used for the GPU performance evaluation.

Table 4.4 compares the energy consumption, latency and EDP between COIN and RTX-8000. COIN shows significant improvement both in energy consumption and EDP for all datasets compared to the GPU implementation. For example, COIN shows 2.4× lower energy than RTX-8000 for the

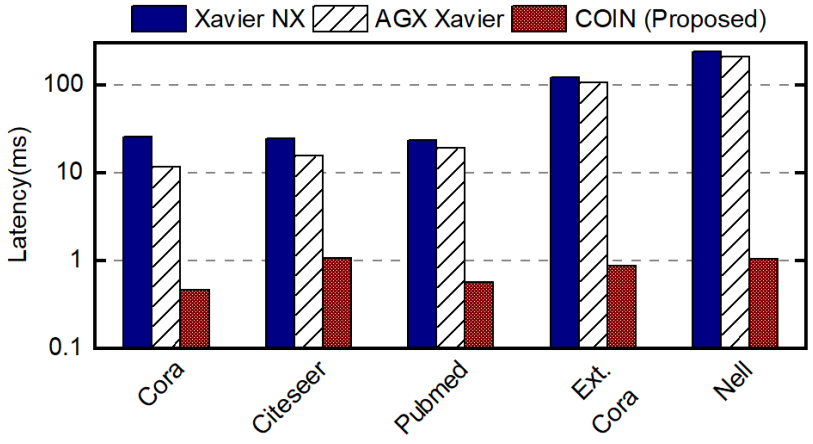


Figure 4.16: Comparison of latency (in log scale) between COIN and edge devices. COIN incurs less latency than both Nvidia Jetson edge devices.

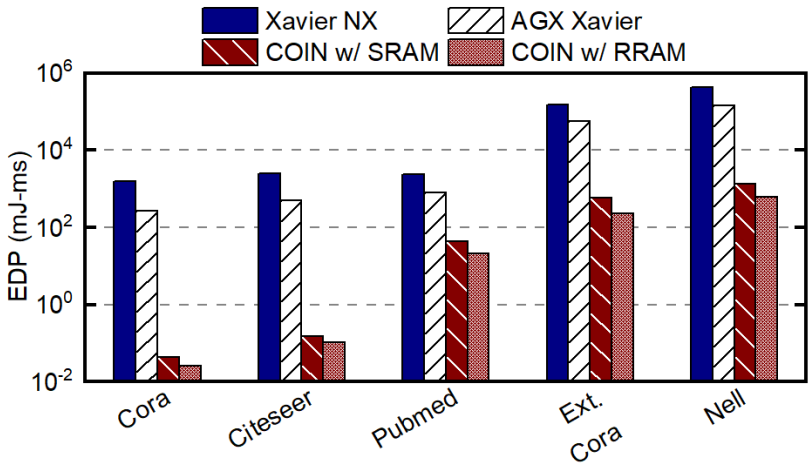


Figure 4.17: Comparison of EDP (in log scale) between COIN and edge devices. We present the performance of COIN with both SRAM and RRAM-based IMC elements. COIN with both kinds of devices outperforms both Xavier NX and AGX Xavier Nvidia Jetson devices across all datasets.

Pubmed dataset. The most significant improvement in energy is observed for Cora (1244×). Except the GCN for Extended Cora dataset, COIN shows improvement in latency over RTX-8000 GPU. We also observe notable improvement in EDP with COIN compared to RTX-8000, as shown in Table 4.4. For Cora, COIN achieves 2526× improvement compared to the GPU. Therefore, the proposed architecture COIN with an optimized NoC leads to significantly lower energy and EDP than state-of-the-art GPU.

We also compare the performance of our design against two edge devices - 1) NVIDIA Jetson Xavier NX and 2) NVIDIA Jetson AGX Xavier. Such a comparison justifies the use of the COIN architecture for edge GCN inference at edge. Table 4.5 shows the configurations of these two devices. We execute the GCN structures of corresponding datasets on the edge devices and record the power value at each epoch of the inference from the power sensor. The total execution time is also recorded while

executing the GCN. Figure 4.15 shows energy consumption of Xavier NX, AGX Xavier, and COIN. We observe significant improvement in energy consumption for all datasets. The improvement is highest for Citeseer dataset. Specifically, for this dataset, COIN's energy consumption is 1448× and 331× lower than Xavier NX and AGX Xavier, respectively. Figure 4.16 compares latency between Xavier NX, AGX Xavier and COIN across different datasets. COIN consistently incurs less latency than both edge devices. The highest improvement in latency is observed for Nell dataset. COIN incurs 232× and 200× less latency than Xavier NX and AGX Xavier respectively. We also compare EDP between COIN and two edge devices. A similar improvement in EDP is observed with COIN. Figure 4.17 shows the comparison for EDP between COIN and edge devices. The EDP of COIN is shown considering both SRAM and RRAM-based IMC elements. On average, COIN achieves 70.7× and 50× improvement in EDP with respect to Xavier NX and AGX Xavier respectively with SRAM-based IMC elements. COIN with RRAM-based IMC elements shows 73.6× and 52.1× improvement in EDP with respect to Xavier NX and AGX Xavier respectively. The largest EDP improvement is observed for the Cora dataset with RRAM-based IMC elements. In this case, COIN achieves 4 orders of magnitude lower EDP than Xavier NX and 3 orders of magnitude lower EDP than AGX Xavier. Therefore, irrespective of the type of IMC elements used, our proposed COIN architecture achieves significantly lower energy and EDP than state-of-the-art GPU and two edge devices for a wide range of popular GCN datasets.

Comparison with State-of-the-art GCN Accelerator

This section compares the performance of our proposed COIN architecture with a state-of-the-art GCN accelerator, ReGraphX [10] and AWB-GCN [64].

Comparison with ReGraphX [10]: The architecture proposed in Re-

Table 4.6: Comparison of energy (mJ) between COIN and state-of-the-art GCN accelerator [64].

	Cora	Citeseer	Pubmed	Nell
AWB-GCN [64]	2.28	3.69	31.5	439
AWB-GCN (scaled to 32nm)	5.27	8.54	73.0	1020
COIN (ours)	0.05	0.10	38.13	577.1
Improvement (×)	**105**	**85.4**	**1.91**	**1.77**

GraphX is composed of multiple processing elements (PEs; similar to computing elements in COIN). Some of the PEs (V-PEs) store the weights and are responsible for the feature extraction operation at GCN nodes (or vertices). The other PEs (E-PEs) store the adjacency matrix of the GCN and enable 'message passing' through the edges of the GCN. Similarly, in ReGraphX-2D, we allocated a set of CEs for feature extraction (V-CEs) and rest to store the adjacency matrix and enable message passing (E-CEs). To

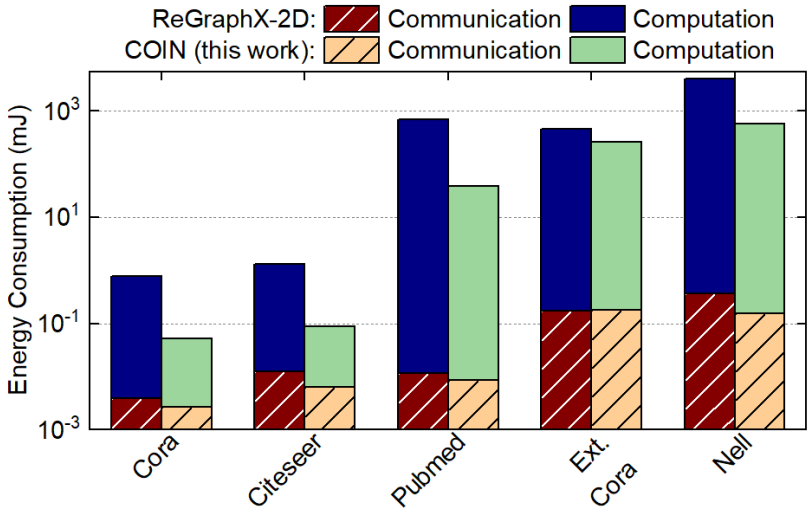


Figure 4.18: Comparison of energy consumption (in log scale) between 2D version of ReGraphX [10] and COIN. The breakdown between communication and computation energy is shown for both the architectures.

Table 4.7: Comparison of EDP (mJ-ms) between COIN and state-of-the-art GCN accelerator [64].

	Cora	Citeseer	Pubmed	Nell
AWB-GCN [64]	0.04	0.11	7.26	1425
AWB-GCN (scaled to 32nm)	0.12	0.33	22.2	4358
COIN (ours)	0.03	0.11	21.58	601
Improvement ($\times$)	**4.68**	**3.09**	**1.03**	**7.25**

have a fair comparison we consider a total of 16 CEs (same as COIN). 4 out of 16 CEs are V-CEs and 12 are E-CEs. The CEs are connected through network-on-chip routers. All CEs consist of 128$\times$128 RRAM crossbar arrays as discussed in [12]. We consider 128$\times$128 RRAM crossbar for COIN too. For a fair comparison we evaluate the performance of ReGraphX-2D through the same simulation environment as COIN.

Figure 4.18 shows the comparison of energy consumption between ReGraphX-2D and COIN. Specifically, we show both computation and communication components of both ReGraphX-2D and COIN. We observe that COIN consumes less energy than the ReGraphX-2D for GCN of all datasets we consider. On average, COIN consumes 8.7$\times$ less total energy than ReGraphX-2D. We also observe that communication energy consumed by COIN is consistently less than the ReGraphX-2D. For example, the GCN for CORA dataset consumes 2.7 μJ and 3.9 μJ of communication energy with COIN and ReGraphX-2D respectively. On average, COIN consumes 1.5$\times$ less communication energy than ReGraphX-2D across different datasets. In our proposed COIN architecture, the hierarchical communication network (inter-CE and intra-CE communication) enables more parallel communication than ReGraphX-2D. Similar to communication energy, COIN consumes less energy for computation as well compared to ReGraphX-2D. On average, the computation energy consumed by COIN is 9$\times$ less than ReGraphX-2D. In COIN, the adjacency matrix is distributed to more number of CEs compared to ReGraphx-2D. Therefore, the memory

utilization of COIN is higher than ReGraphx-2D. Hence, ReGraphX-2D requires more in-memory computing (IMC) tiles than COIN which results in higher computation energy consumption with ReGraphX-2D.

Comparison with AWB-GCN [64]: We note that we use 32 nm technology to evaluate COIN. However, AWB-GCN uses Intel D5005 equipped with Statix 10 SX FPGA. This FPGA incorporates 14 nm technology. Therefore, we estimate the performance of AWB-GCN with 32 nm technology using the technique described in [190, 182]. AWB-GCN stores the adjacency matrix and the weights in off-chip memory. The sparse matrix multiplication kernel periodically accesses the off-chip memory and performs the computation. Specifically, the AWB-GCN accelerator requires up to 503 Gbps off-chip memory bandwidth to fully utilize the hardware. Since AWB-GCN uses off-chip memory, it suffers from high energy consumption. In contrast, we use in-memory computing (IMC) to construct COIN without requiring frequent off-chip memory access. Moreover, we incorporate an optimization technique to reduce on-chip communication energy. The comparison in energy consumption between AWB-GCN and COIN for different GCN datasets is shown in Table 4.6. We present both the energy consumption reported in [64] and the energy consumption when the technology node is scaled to 32nm in the table. The energy with the Extended Cora dataset is not reported in AWB-GCN. Therefore, we cannot compare the results for Extended Cora. The improvement is shown with respect to the energy consumption of AWB-GCN when scaled to 32nm. We observe that COIN provides significant improvement in energy for all the datasets we consider. The most significant improvement is seen for the Cora dataset ($105\times$). On average, COIN shows a $13.2\times$ improvement in energy consumption over AWB-GCN.

Furthermore, the comparison in EDP between AWB-GCN and COIN for different GCN datasets is shown in Table 4.7. Similar to Table 4.6, we present both the EDP reported in [64] and the EDP when the technology

node is scaled to 32nm in the table. COIN shows improvement in EDP for all the datasets we consider. The most significant improvement is seen for the Nell dataset ($7.25\times$). The vast improvement in energy consumption as well as EDP comes from IMC-based hardware and our proposed communication-aware technique to construct the GCN accelerator.

5 PERFORMANCE ANALYSIS OF PRIORITY-AWARE NOCS

5.1 Background and Motivation

Industrial many-core processors incorporate priority arbitration for the routers in NoC [88]. Moreover, these designs execute bursty traffic since real applications exhibit burstiness [25]. Accurate NoC performance models are required to perform design space exploration and accelerate full-system simulations [102, 174]. 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 [193] and high performance computing (HPC) [88]. A recent technique targets priority-aware NoCs [136], 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* [25]. 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

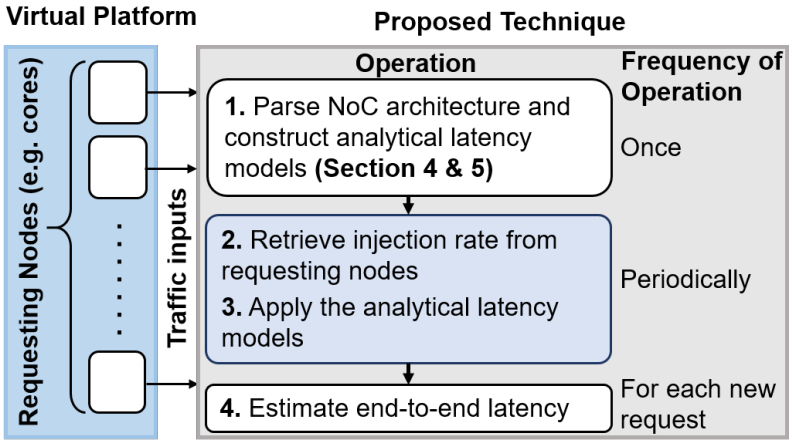


Figure 5.1: Overview of the proposed methodology.

queue nodes to obtain end-to-end latency.

The proposed performance analysis technique is evaluated with SYSmark® 2014 SE [15], applications from SPEC CPU® 2006 [77] and SPEC CPU® 2017 [32] 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.

Proposed Performance Analysis Flow

The primary target of the proposed model is to accelerate virtual platforms [16] and full-system simulations [24, 167, 134] 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 5.1. 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 5.1) 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.

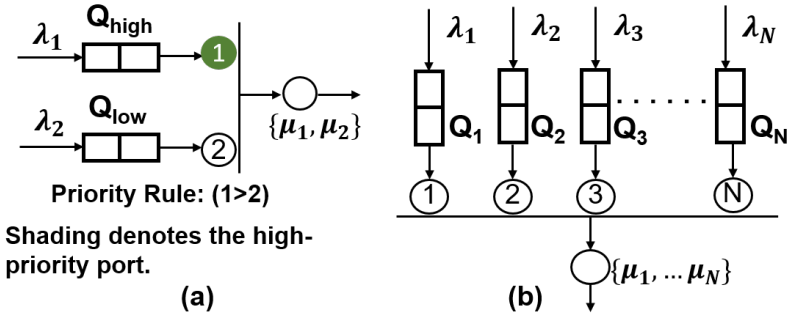


Figure 5.2: (a) A system with two queues. Flits in Q_{high} have higher priority than flits in Q_{low}. (b) A system with N queues, where Q_i has higher priority than Q_j for $i < j$

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 demonstrate in Section 5.4. 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 [158].

Performance analysis techniques in the literature [19, 91, 102] discuss basic priority-based networks in which each priority class has a dedicated queue, as illustrated in Figure 5.2(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 5.2(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 [19]. 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 5.2(b). Suppose that we are given the injection rates λ_i, service rates μ_i, residual time R_i, and server utilizations ρ_i for $1 \leqslant i \leqslant N$, where $N \geqslant 2$ Then, the waiting time of class-i flits W_i is given as:

$$W_i = \begin{cases} \frac{\sum_{k=1}^{N} R_k}{1-\rho_1}, & \text{for } i = 1 \\ \frac{\sum_{k=1}^{N} R_k + \sum_{k=1}^{i-1} \left(\rho_k + \rho_k W_k\right)}{1-\sum_{k=1}^{i} \rho_k}, & \text{for } i > 1 \end{cases} \tag{5.1}$$

The remaining details of the work is described in Appendix C of the thesis and in the reference [136].

5.2 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 5.3(a). As illustrated earlier, 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 5.3(b). In this figure, μ_i represents the service

rate of class-i for $i = 1, 2, 3$. If we use Equation 5.1 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 5.4. 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 5.3(b) to Figure 5.3(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, since μ_1 and μ_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 5.1. 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.

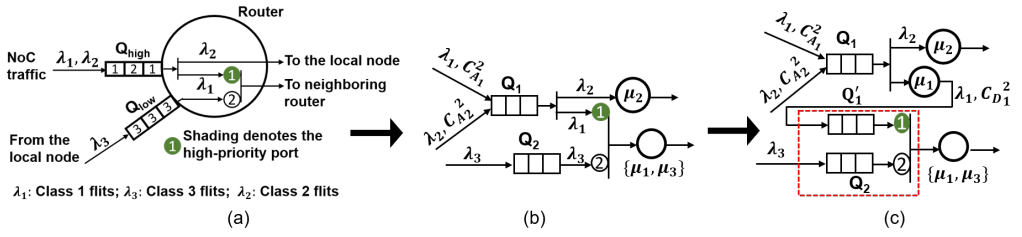


Figure 5.3: Split at high priority: Structural Transformation.

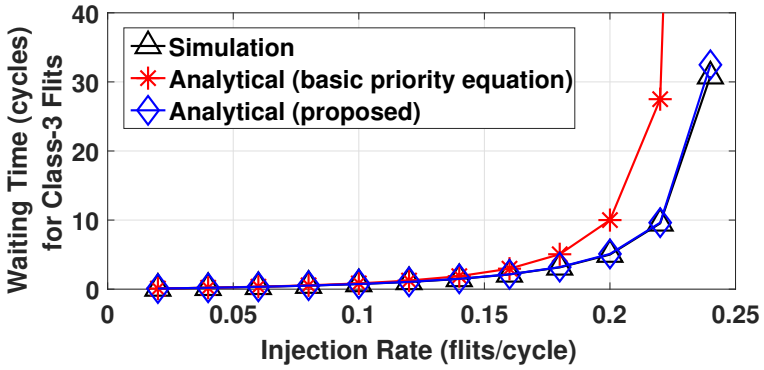


Figure 5.4: Comparison of simulation with the basic priority-based queuing model and proposed analytical model.

We derive the second order moment of inter-departure time of class-1 using the decomposition technique presented in [28]. 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 5.5. 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 5.5-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 5.5(Phase 3). By calculating the first two moments of the inter-arrival statistics of Q_1' as λ_1 and $C_{D_1}^2$, we ensure that the transformed structure in Figure 5.3(c) approximates the original system. This decomposition enables us to find the residual time for class-1 $R_1^{Q_1'}$ as:

$$R_1^{Q_1'} = \frac{1}{2}\frac{\rho_1}{\mu_1}\left(\frac{C_{D_1}^2 + C_B^2}{2}\right) - \frac{\rho_1\mu_1}{2} \tag{5.2}$$

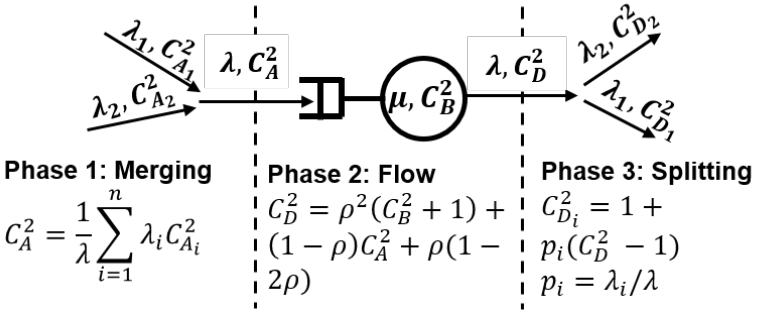


Figure 5.5: Decomposition technique: In phase 1, different traffic flows merge into a single flow with an inter-arrival time C_A; in phase 2, flits flow into the queue and leave the queue with an inter-departure time C_D; in phase 3, flits split into different flows with individual inter-departure time.

Proposed Analytical Model: The bottom part of the transformed system in Figure 5.3(c) is the basic priority queue (marked with the dotted red box). Therefore, the higher priority part of Equation 5.1 can be used to express the waiting time of class-1 flits as:

$$W_1^{Q_1'} = \frac{R_1^{Q_1'} + R_3}{1 - \rho_1} \tag{5.3}$$

where the residual time of class-1 flits $R_1^{Q_1'}$ is found using Equation 5.2. Subsequently, this result is substituted in the lower priority portion of Equation 5.1 to find the waiting time for class-3 flits:

$$W_3 = \frac{R_1^{Q_1'} + R_3 + \rho_1 + \rho_1 W_1^{Q_1'}}{1 - \rho_1 - \rho_3} \tag{5.4}$$

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 5.1 for the degenerate case of $N = 1$.

Figure 5.4 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 5.3(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 5.6(a). In this system, class-1 flits (λ_1) are waiting in Q_{high}, while class-2 flits (λ_2) and class-3 flits (λ_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 5.6(b) is used as an intermediate 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 5.7. 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.

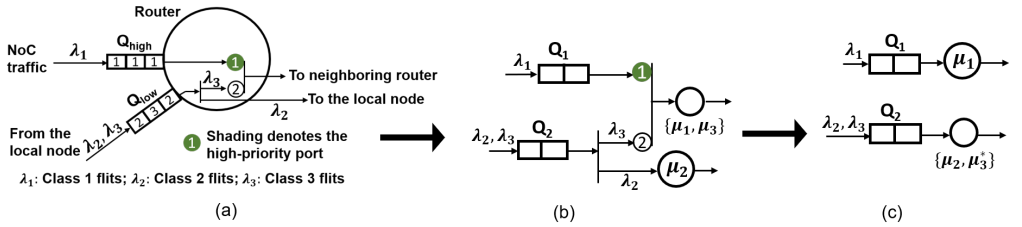


Figure 5.6: Split at low priority: Service Rate Transformation. μ^* denotes transformed service rate. The waiting time of class-1 flits depends on the residual time of the class-3 flits, as shown in Equation 5.5.

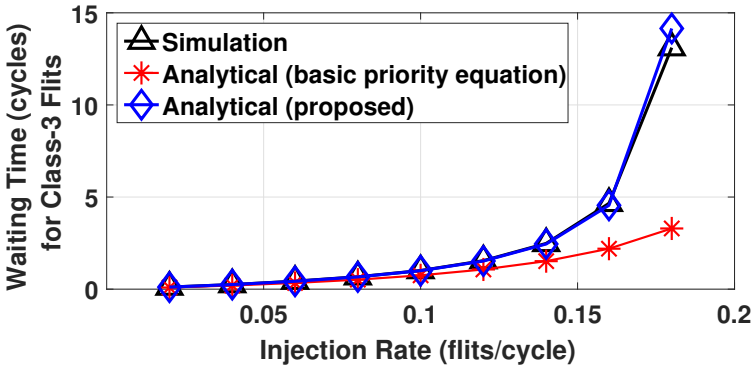


Figure 5.7: Comparison of simulation with the basic priority-based queuing model and proposed analytical model.

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 5.1 as:

$$W_1 = \frac{R_1 + R_3}{1 - \rho_1} \tag{5.5}$$

Hence, we represent Q_1 as a stand-alone queue, as shown in Figure 5.6(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 5.6(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 5.1 as:

$$W_3\Big|_{\lambda_2=0} = \frac{R_1 + R_3 + \rho_1 + \rho_1 W_1}{1 - \rho_1 - \rho_3} \tag{5.6}$$

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 ρ_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:

$$p = \rho_1 + \lambda_1 R_3 \tag{5.7}$$

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 (ΔT_3) is expressed as:

$$\begin{aligned} \Delta T_3 &= T_1 p(1-p) + 2T_1 p^2(1-p) + ... + nT_1 p^n(1-p) + \cdots \\ &= T_1 \frac{p}{1-p} \end{aligned} \tag{5.8}$$

Consequently, the modified service time (T_3^*) and utilization (ρ_3^*) of class-3 can be expressed as:

$$\begin{aligned} T_3^* &= T_3 + \Delta T_3 \\ \rho_3^* &= \lambda_3 T_3^* \end{aligned} \tag{5.9}$$

Suppose that the modified residual time of class-3 is denoted by R_3^*. We can plug R_3^*, the modified utilization ρ_3^* from Equation 5.9, and the additional busy period ΔT_3 from Equation 5.8 into Geo/G/1 model to express the waiting time W_3 as:

$$W_3 = \frac{R_3^*}{1-\rho_3^*} + \Delta T_3 \tag{5.10}$$

When λ_2 is set to zero, this expression should give the class-3 waiting time $W_3\Big|_{\lambda_2=0}$ found in Equation 5.6. Hence, we can find the following

expression for R_3^* by combining Equation 5.6 and Equation 5.10:

$$R_3^* = (1 - \rho_3^*)(W_3\Big|_{\lambda_2=0} - \Delta T_3) \tag{5.11}$$

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:

$$\begin{aligned} W_2 &= \frac{R_3^* + R_2}{1 - \rho_3^* - \rho_2} \\ W_3 &= \frac{R_3^* + R_2}{1 - \rho_3^* - \rho_2} + \Delta T_3 \end{aligned} \tag{5.12}$$

Figure 5.7 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.

5.3 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 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 5.2 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

Algorithm 2: End-to-end queuing time calculation for different traffic classes

1 **Input:** Injection rates for all traffic classes, NoC topology and Traffic routing pattern
2 **Output:** Queuing time for all traffic classes
3 **for** *n = 1: no. of queues* **do**
4 For queuing time expression of the current queue:
5 Initialize: $num_1 \leftarrow 0$, $den_1 \leftarrow 1$
6 Get all classes in current queue
7 **for** *i = 1: no. of classes* **do**
8 Get all higher priority classes than current class
9 For reference queuing time (W_{ref}) of current class:
10 Initialize: $num_2 \leftarrow R_i$, $den_2 \leftarrow (1 - \rho_i)$
11 **for** *j = 1:no. of higher priority classes* **do**
12 Calculate coefficient of variation (C_D) for current high priority class
13 Calculate queuing time expression (W_{ij}) and residual time expression (R_{ij}) using C_D using Eq. 5.2, Eq. 5.3, and Eq. 5.4
14 $num_2 \leftarrow num_2 + R_{ij} + \rho_{ij} + W_{ij}\rho_{ij}$
15 $den_2 \leftarrow den_2 - \rho_{ij}$
16 **end**
17 $W_{ref} = \frac{num_2}{den_2}$ (Eq. 5.6)
18 Modify service rate (T_i^*) of i^{th}class
19 Calculate residual time (R_i^*) using T_i^* and W_{ref} using Eq. 5.11
20 $num_1 \leftarrow num_1 + R_i^*$
21 $den_1 \leftarrow den_1 - \rho_i^*$
22 **end**
23 Queuing time of class-i in n^{th} queue = $\frac{num_1}{den_1} + \Delta T_i$
24 **end**

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

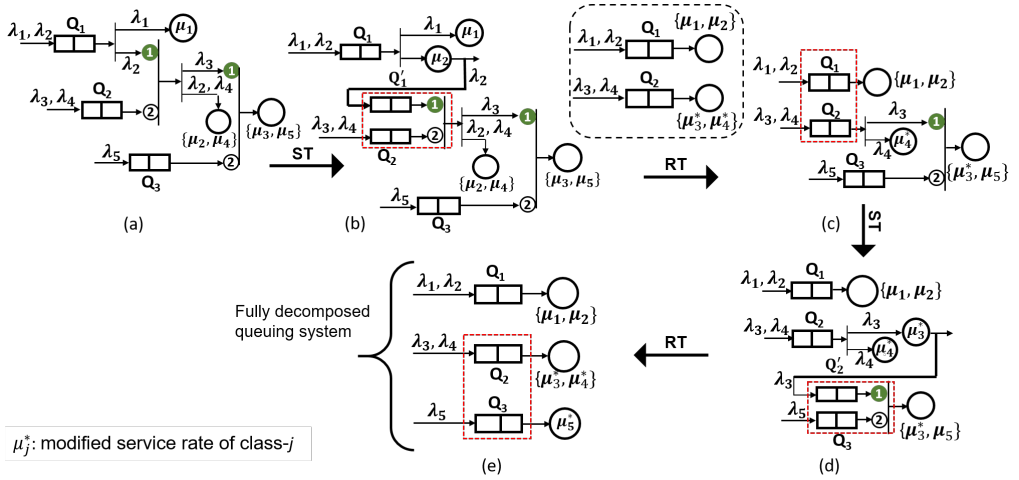


Figure 5.8: Applying the proposed methodology on a representative segment of a priority-based network. ST and RT denote Structural and Service Rate Transformation, respectively. Red-dotted squares show the transformed part from the previous step. Figure (a) shows the original queuing system. After applying 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).

structural transformation as described in Section 5.2 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 5.2. 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 5.8 illustrates the proposed approach on a representative example of a priority-based network to decompose the system. Figure 5.8(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 5.8(b). Next, the service rate transformation on class-3 and class-4 is applied to obtain modified service time (μ^*). This transformation allows us to form the network by decomposing Q_1 and Q_2, as depicted in Figure 5.8(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 5.8(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.

5.4 Experimental Evaluations

Experimental Setup

We applied the proposed analytical models to a widely used priority-based industrial NoC design [88]. 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 [160, 159]. 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 λ_1 to λ_{max}. Beyond λ_{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 [24] using Linux 'perf' tools [53]. 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 5.9 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 5.9 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.

Simulation Time: To evaluate the decrease in simulation time with the

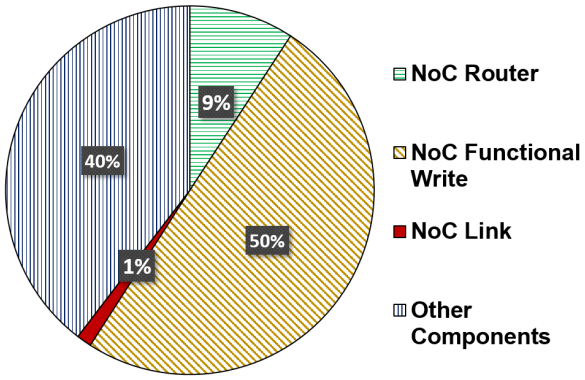


Figure 5.9: The fraction of simulation time spent by different functions while running Streamcluster in gem5. NoC-related functions take 60% of simulation time.

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× 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 5.10. 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%.

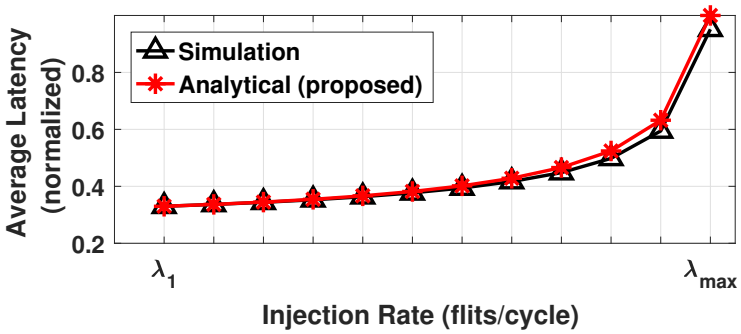


Figure 5.10: Evaluation of the proposed model on a ring with eight nodes.

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 [88], 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 5.4. 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 5.11 and Figure 5.12, 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 [202]. 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 5.11 and Figure 5.12 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.

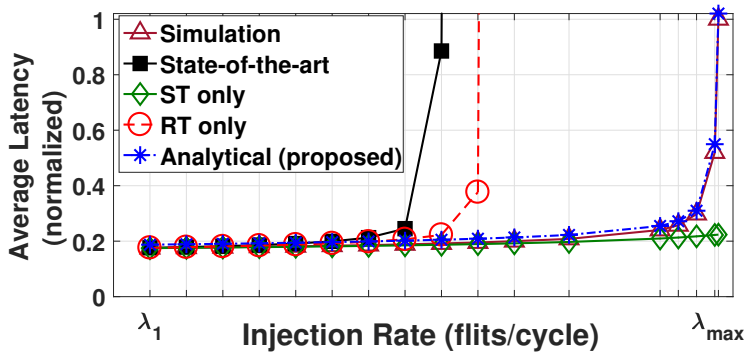


Figure 5.11: Evaluation of the proposed model on a 6×6 mesh.

Impact of coefficient of variation: One of the important parameters in 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 5.13. We observe that the proposed technique accurately estimates the average latency in comparison to cycle-accurate simulation. On average, the analytical models are

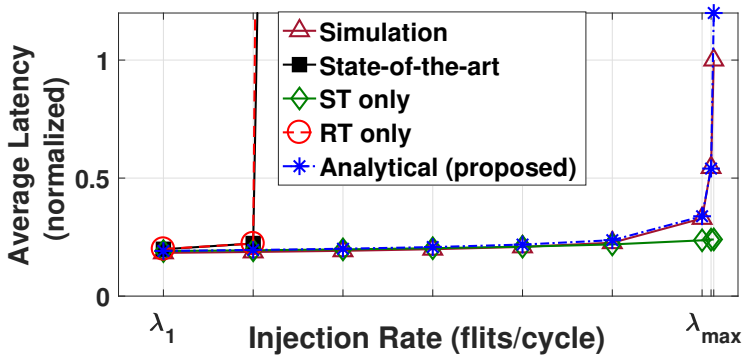


Figure 5.12: Evaluation of the proposed model on an 8×8 mesh.

97% accurate with respect to latency obtained from the simulation in this case.

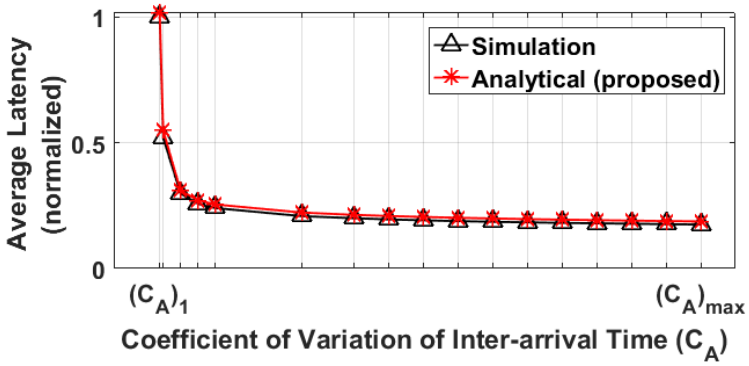


Figure 5.13: Effect of coefficient of variation of inter-arrival time on average latency for a 6×6 mesh.

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 [56] 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

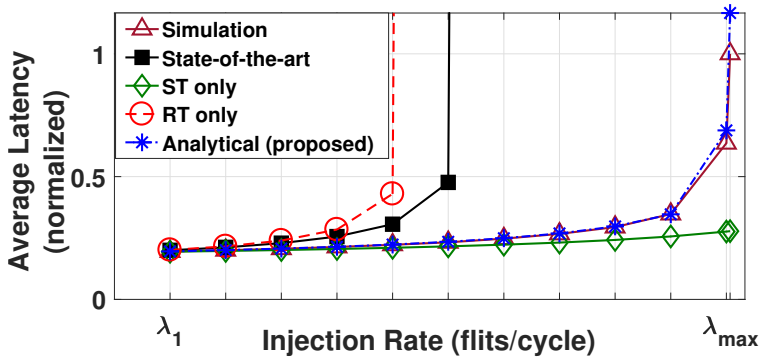


Figure 5.14: Evaluation of the proposed model on one variant of the Xeon server architecture.

proposed analytical models with this floorplan is shown in Figure 5.14. 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 8×8 mesh, the state-of-the-art NoC performance analysis technique [202] highly overestimates the average latency for this server architecture, as shown in Figure 5.14. 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 5.15 shows the latency of each class of the server architecture described above normalized 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.

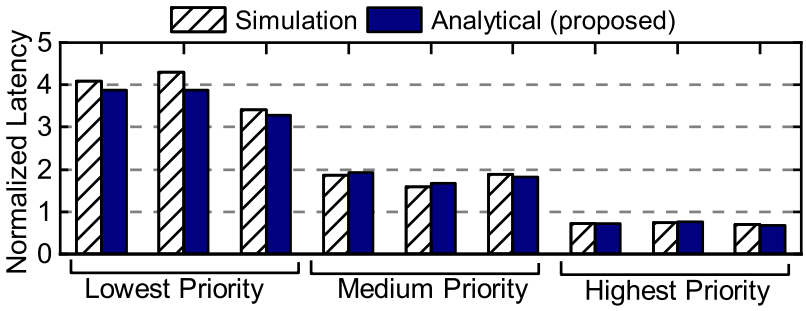


Figure 5.15: Per-class latency comparison for the server example.

Table 5.1: 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

Finally, we evaluate the proposed technique with different LLC hit rates. Table 5.1 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 [24] to extract traces of applications in Full-System (FS) mode. Garnet2.0 [6] is used as the network simulator in gem5 with the Ruby memory system. Table 5.2 shows the various configuration settings we used for FS simulation in gem5.

We collect traces of six 16-threaded applications from PARSEC [23]

benchmark suites: Blackscholes, Canneal, Swaptions, Bodytrack, Fluidanimate, and Streamcluster. We selected applications that show relatively higher network utilization as discussed in [209]. 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 [26], 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 rates are fed to our analytical models to obtain average latency.

Figure 5.16 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 ($L_{analytical}$).

Table 5.2: 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

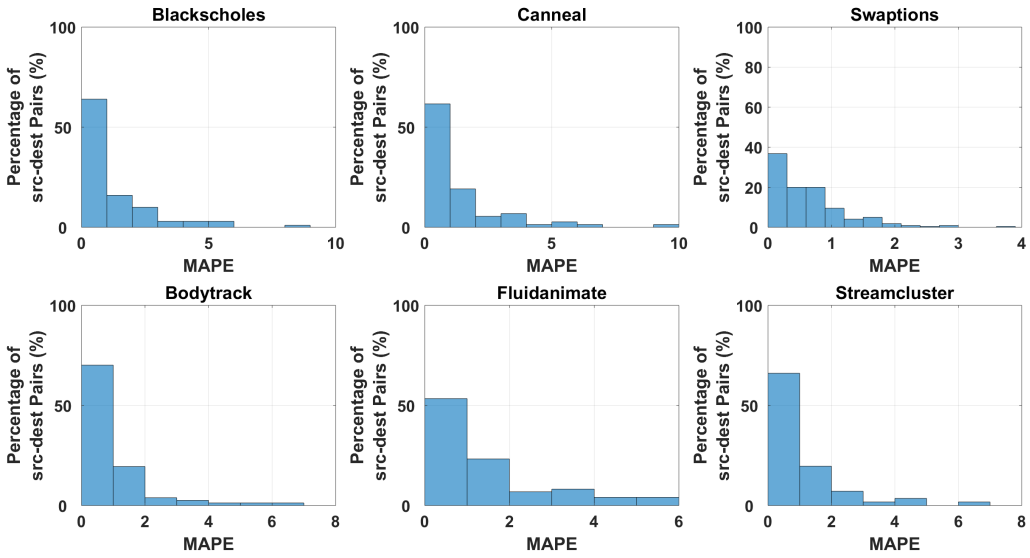


Figure 5.16: Model comparison for different applications from PARSEC suite.

MAPE is defined by the following equation:

$$\text{MAPE} = 100\left(\frac{|\text{L}_{sim} - \text{L}_{analytical}|}{\text{L}_{sim}}\right) \tag{5.13}$$

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 application. 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

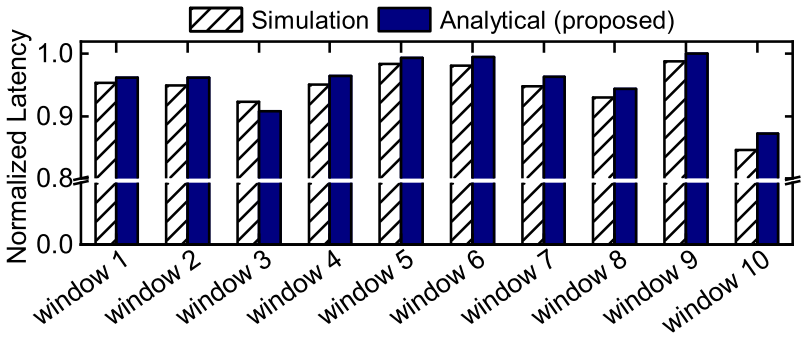


Figure 5.17: Evaluation of the proposed model under a finer level of time granularity (100K cycles) for Streamcluster application.

windows containing 100,000 cycles each. Average latency comparison for Streamcluster application in these smaller windows is shown in Figure 5.17. 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 earlier 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 [158]. 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 PERFORMANCE ANALYSIS OF NOCS WITH BURSTY TRAFFIC

6.1 Background and Motivation

Industrial many-core processors incorporate priority arbitration for the routers in NoC [88]. Moreover, these designs execute bursty traffic since real applications exhibit burstiness [25]. Accurate NoC performance models are required to perform design space exploration and accelerate full-system simulations [102, 174]. 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 [193] and high performance computing (HPC) [88]. A recent technique targets priority-aware NoCs [136], 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* [25]. 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 net-

work 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 [15], applications from SPEC CPU® 2006 [77] and SPEC CPU® 2017 [32] 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 [88, 193]. 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 [25, 174]. We approximate the input traffic using the GGeo discrete-time distribution model, which takes both burstiness and discrete-time feature of NoCs into account [107, 174]. GGeo model includes Geometric and null (no delay) branches, as shown in Figure 6.1. Selection between branches conforms to the Bernoulli trial,

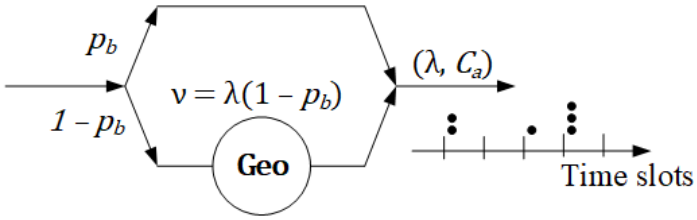


Figure 6.1: GGeo traffic model

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 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 [107]. Hence, we use p_b as a parameter of burstiness. GGeo distribution has two important properties [107]. *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 (λ, 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 thesis and in the reference [135].

6.2 Proposed Approach to Handle Bursty Traffic

In industrial NoCs, flits already in the network have higher priority than new injections to achieve predictable latency [88]. 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 [163].

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) [107]. 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 6.2(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

Table 6.1: Summary of the notations used in this work.

$\overline{\lambda}, \lambda_m$	Mean arrival rate of total traffic and class m
p_b	Probability of burstiness
$\overline{T}_m, \widehat{T}_m$	Original and modified mean service time of class m flits
$\overline{R}, R_{mk}$	Total residual time and residual time of class m while class k is served
ρ_m	Mean server utilization of class m flits ($=\lambda_m T_m$)
$\overline{C}_a, C_{a_m}$	Coeff. of variation of interarrival time of total traffic and class m flits
$\overline{C}_{s_m}, \widehat{C}_{s_m}$	Coeff. of variation of original and modified service time of class m flits
$\overline{C}_d, C_{d_m}$	Coeff. of variation of interdeparture time of total traffic and class m flits
$\overline{W}_m$	Mean waiting time of class m flits
$\overline{n}_m, n_m$	Mean and current occupancy of class m flits in a queue-node
$\overline{\beta}_m$	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

this system into individual queue-nodes with modified servers, as shown in Figure 6.2(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:

$$\widehat{T}_m = \frac{1 - p_m(0)}{\lambda_m} \tag{6.1}$$

where $p_m(0)$ is the marginal probability of having no flits of class m in the queue-node, as listed in Table 6.1.

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 (6.2) subject to the constraints listed in (6.3):

$$\underset{p}{\text{maximize}} \qquad H(p(\mathbf{n})) = -\sum_{\mathbf{n}} p(\mathbf{n}) \log(p(\mathbf{n})) \tag{6.2}$$

$$\begin{aligned} \text{subject to} \qquad & \sum_{\mathbf{n}=\mathbf{0}}^{\infty} p(\mathbf{n}) = 1, \\ & \sum_{\substack{\mathbf{n}=\mathbf{0} \\ \text{except} \\ n_m=1}}^{\infty} \alpha_m(\mathbf{n}) p(\mathbf{n}) = \rho_m, \; m = 1, \ldots, M \\ & \sum_{\substack{\mathbf{n}=\mathbf{0} \\ \text{except} \\ n_m=n_k=1}}^{\infty} n_m \alpha_k(\mathbf{n}) p(\mathbf{n}) = \bar{n}_{mk}, m, k = 1, .., M \end{aligned} \tag{6.3}$$

The notation ∞ means a state vector $\mathbf{n}$ with all elements set to ∞, and ($\mathbf{n} = \mathbf{0}$ 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 (6.3) comprise three types: normalization, mean server utilization and mean occupancy. We introduced an extended set of mean occupancy constraints compared to [107]

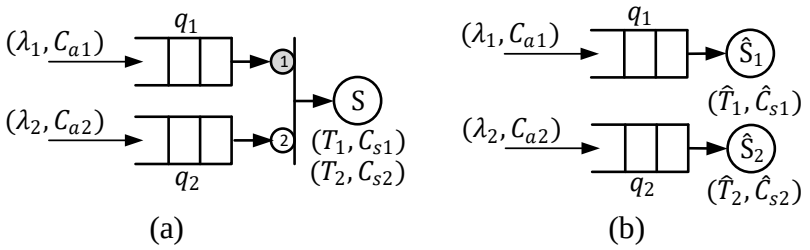


Figure 6.2: Decomposition of a basic priority queuing

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 6.2(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, β_m, R_{mk} and W_m (defined in Table 6.1) derived in [107, 136].

We solve the nonlinear programming problem in (6.2, 6.3) 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 queues in a priority structure with a single class per queue as:

$$p_m(0) = 1 - \rho_m - \sum_{k=1, k\neq m}^{M} \rho_k \frac{\overline{n}_{mk}}{\rho_k + \overline{n}_{mk}} \tag{6.4}$$

Plugging the expression of $p_m(0)$ from (6.4) into (6.1), 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 [107] and the modified server utilization $\widehat{\rho}_m = \lambda_m \widehat{T}_m$ to obtain the following expression for $\widehat{C}^2_{s_m}$:

$$\widehat{C}^2_{s_m} = \frac{(1-\widehat{\rho}_m)(2\overline{n}_m - \widehat{\rho}_m) - \widehat{\rho}_m C^2_{a_m}}{\widehat{\rho}^2_m} \tag{6.5}$$

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 6.3(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 [70]. 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 6.3(b). First, we set λ_3 to zero which leads to a basic priority structure. Then, we apply the decomposition method discussed in Section 6.2 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:

$$W_m = \frac{R + \sum_{k=1}^{M} \widehat{\rho}_k \widehat{T}_k \beta_k}{1 - \sum_{k=1}^{M} \widehat{\rho}_k} + \widehat{T}_m(\beta_m + 1) - T_m \tag{6.6}$$

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

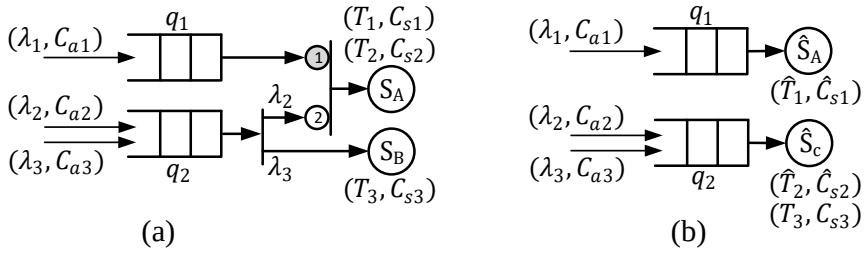


Figure 6.3: Decomposition of flow contention at low priority

The other basic structure, *contention at high priority*, is shown in Figure 6.4(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, $C^2_{d_2}$, of class 2 using the split process formulation of GGeo streams given in [107]. 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 6.4(b). Subsequently, we apply the decomposition method described in Section 6.2 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 6.4(c).

Iterative decomposition algorithm

Algorithm 3 shows a step-by-step procedure to obtain the analytical model using our approach described in Section 6.2. 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

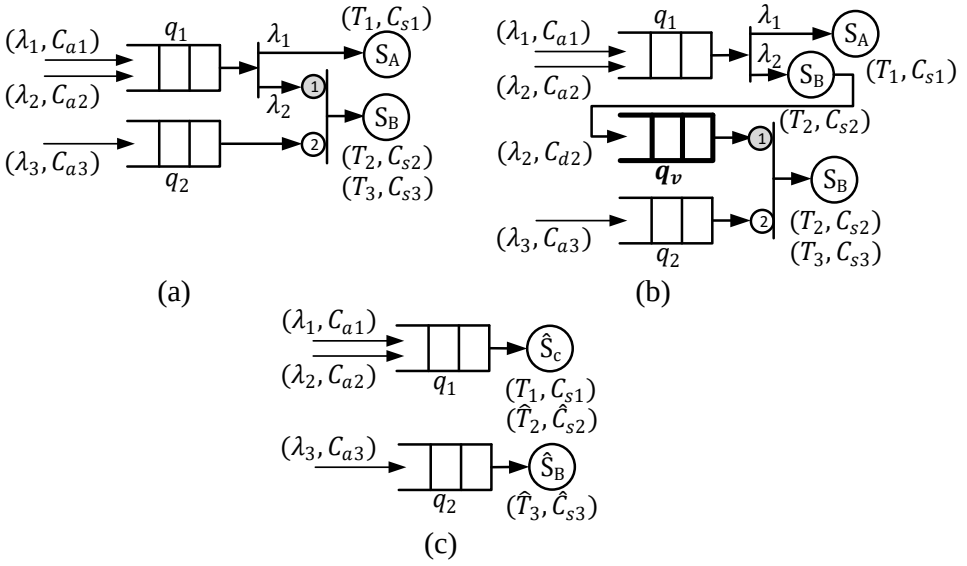


Figure 6.4: Decomposition of flow contention at high priority

contention at high and low priority, as shown in line 7 – 8 of Algorithm 3. Subsequently, we calculate the modified service process $(\widehat{T}, \widehat{C}_s^2)$ using (6.1, 6.4) and (6.5). Then, we compute the waiting time per class following (6.6). Finally, we obtain the average waiting time in each queue (W_q), as shown in line 12.

6.3 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 the models is evaluated against an industrial cycle-accurate simulator [159] 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 6.2 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 6.2 also shows that priority-based analytical models *which do not* consider burstiness [136] significantly underestimate the latency by 33% on average (highlighted with the shaded row). In contrast, the work without the proposed decomposition technique [102] leads to over 100% overestimation even at low traffic loads (highlighted with text in italics). In this case, GGeo models can not handle partial

Algorithm 3: Iterative Decomposition Algorithm

Input: NoC topology, routing algorithm, server process, (λ) and (p_b) for each class as parameters
Output: Average waiting time for each queue (W_q)
N = number of queues in the network
S_q = set of classes in queue q
for $q = 1$:N **do**
    for $m = 1$:$|S_q|$ **do**
        Apply decomp. for contention at high priority (if found)
        Apply decomp. for contention at low priority (if found)
        Compute $\widehat{T}$, $\widehat{C}_s^2$ using (6.1, 6.4) and (6.5)
        Compute queuing time ($W_{q,m}$) using (6.6)
    end
    $W_q = \frac{\sum_{i=1}^{|S_q|} \lambda_{q,m} W_{q,m}}{\sum_{i=1}^{|S_q|} \lambda_{q,m}}$
end

Table 6.2: Comparisons against existing alternatives (Reference [102] and Reference [136]). *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[102]	*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[136]	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

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.

Evaluation on Architectures with Mesh NoCs

Table 6.2 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 [88]. 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 [136] 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 analytical models on 6×6 and 8×8 NoC for burst probability of 0.2 and 0.6 in Figure 6.5(a) and Figure 6.5(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 [15], and applications from SPEC CPU® 2006 [77] and SPEC CPU® 2017 [32] benchmark suites. These applications are chosen since they show different levels of burstiness. First, we run these applications

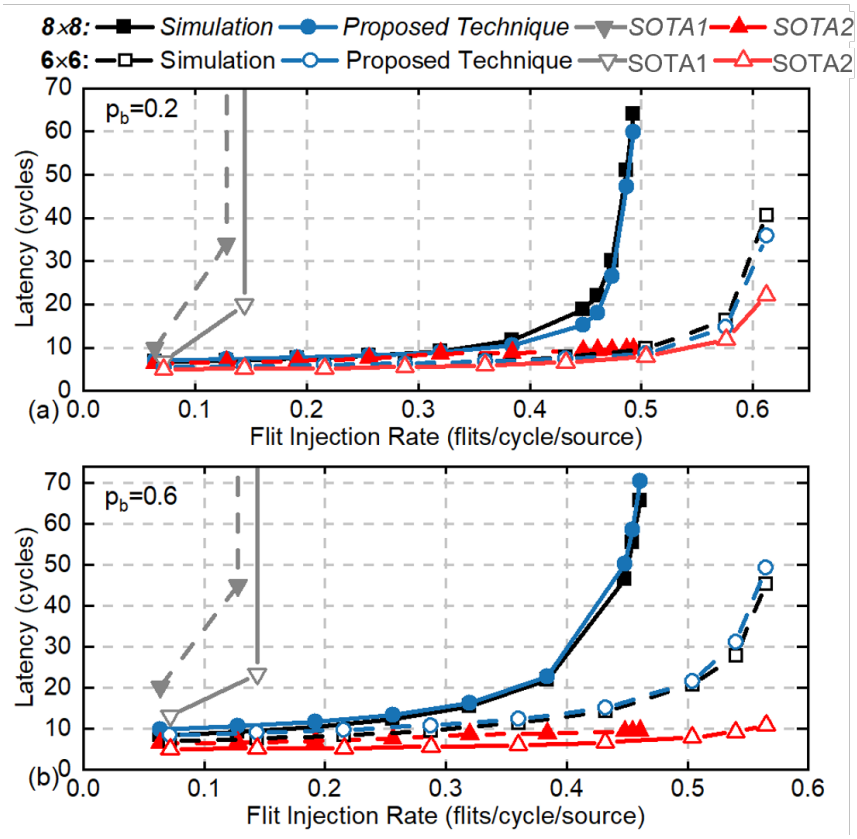


Figure 6.5: Comparison of a proposed analytical model with cycle-accurate simulation for 8×8 and 6×6 mesh for (a) $p_b = 0.2$ and (b) $p_b = 0.6$. SOTA1 and SOTA2 refer to the analytical modeling techniques proposed in [102] and [136] respectively.

on gem5 [24] and collect traces with timestamps for each packet injection. Then, we use the traces to compute the injection rate (λ) 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 [107] to find the occupancy and then p_b of each class.

The proposed analytical models are used to estimate the latency using

Table 6.3: 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 [102]	14.62	11.99	7.69	12.29	5.18	13.64	11.46	7.25
	Ref [136]	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 [102]	10.33	12.73	12.07	22.90	19.17	9.93	5.99	19.04
	Ref [136]	12.15	29.99	10.00	19.65	5.44	10.78	14.74	7.94

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 6.3, 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 [102] and [136] incur significant modeling error.

7 PERFORMANCE ANALYSIS OF NOCS WITH DEFLECTION ROUTING

7.1 Background and Motivation

Pre-silicon design-space exploration and system-level simulations constitute a crucial component of the industrial design cycle [164, 65]. They are used to confirm that new generation designs meet power-performance targets before labor- and time-intensive RTL implementation starts [24]. Furthermore, virtual platforms combine power-performance simulators and functional models to enable firmware and software development while hardware design is in progress [124]. These pre-silicon evaluation environments incorporate cycle-accurate NoC simulators due to the criticality of shared communication and memory resources in overall performance [6, 90]. 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 [218].

Analytical NoC performance models are used primarily for fast design space exploration since they provide significant speed-up compared to detailed simulators [158, 102, 99, 174]. However, most existing analytical models fail to capture two important aspects of industrial NoCs [88]. *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

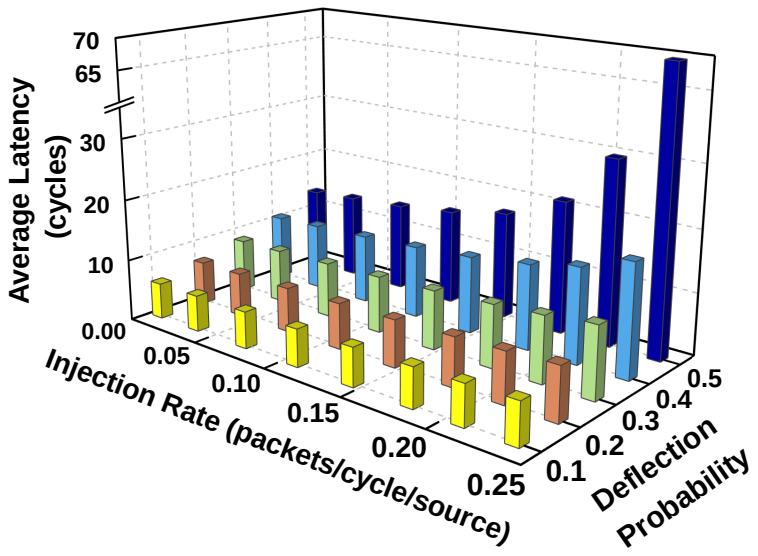


Figure 7.1: Cycle-accurate simulations on a 6×6 NoC show that the average latency increases significantly with larger deflection probability (p_d) at the sink.

NoCs [136, 135]. 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 7.1 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 modeling 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 [102, 136]. 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.

Background on Deflection Routing

Assumptions and Notations

Architecture: This work considers priority-aware NoCs used in high-end servers and many core architectures [88]. Each column of the NoC architecture, shown in Figure 7.2, is also used in client systems such as Intel i7 processors [179]. 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 7.2 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 7.2). Then, it competes with the packets already in the corresponding row. That is, the path from the source (Node

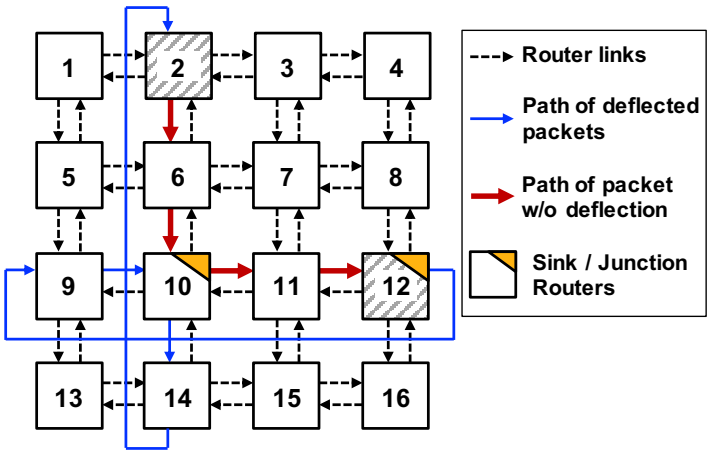


Figure 7.2: A representative 4×4 mesh with deflection routing.

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 7.2. 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 [25, 174]. This work considers generalized geometric (GGeo) distribution for the input traffic, which takes burstiness into account [107]. GGeo traffic is characterized by an average injection rate (λ) 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 λ_i and as C_i^A respectively, as shown in Table 7.1. 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 (λ, 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 (λ, 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 D of the thesis and in the reference [139].

7.2 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

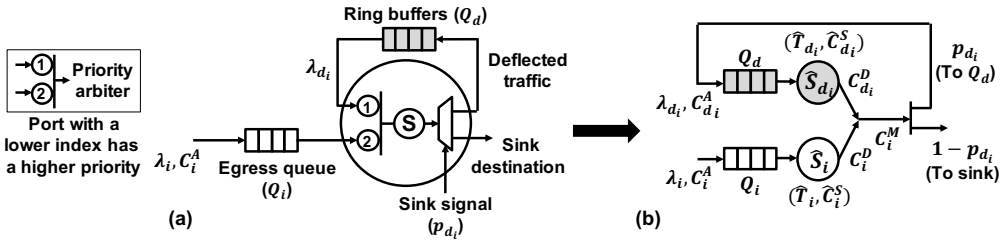


Figure 7.3: (a) Queuing system of a single class with deflection routing (b) Approximate queuing system to compute $C_{d_i}^A$.

Table 7.1: Summary of the notations used in this work.

λ_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
ρ_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

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. 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. 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.

An Illustration with a Single Traffic Class

Figure 7.3(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 7.2). We denote the traffic of Q_i as class-i, which is modeled using GGeo distribution with two parameters (λ_i, C_i^A). The packets in Q_i are dispatched to a priority arbiter

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 7.2) and reaches the destination. Since the number of hops is constant for a particular traffic class, *we omit these details in Figure 7.3(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 7.2) 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 7.3(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.. To obtain W_i, we first need to derive the analytical expression for the rate of deflected packets of class-i (λ_{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 (λ_{d_i}): λ_{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:

$$\begin{aligned} N_{d_i} &= p_{d_i}(1-p_{d_i}) + 2p_{d_i}^2(1-p_{d_i}) + \ldots + np_{d_i}^n(1-p_{d_i}) + \ldots \\ &= \sum_{n=1}^{\infty} np_{d_i}^n(1-p_{d_i}) = \frac{p_{d_i}}{1-p_{d_i}} \end{aligned} \tag{7.1}$$

Therefore, λ_{d_i} can be expressed as:

$$\lambda_{d_i} = \lambda_i N_{d_i} = \lambda_i \frac{p_{d_i}}{1 - p_{d_i}} \tag{7.2}$$

Coefficient of variation of inter-arrival time of deflected packets ($C^A_{d_i}$): To compute $C^A_{d_i}$, 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 7.3(a) into an approximate representation shown in Figure 7.3(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 [107]. 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,

$$p_{Q_i}(0) = 1 - \rho_i - \rho_{d_i} \frac{\overline{n}_i}{\overline{n}_i + \rho_i + \rho_{d_i}} \tag{7.3}$$

where ρ_i and ρ_{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:

$$\widehat{T}_i = \frac{1 - p_{Q_i}(0)}{\lambda_i} \tag{7.4}$$

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

$$(\widehat{C}_i^S)^2 = \frac{(1 - \widehat{\rho}_i)(2\overline{n}_i + \widehat{\rho}_i) - \widehat{\rho}_i (C_i^A)^2}{\widehat{\rho}_i^2} \tag{7.5}$$

where $\widehat{\rho}_i = \lambda_i \widehat{T}_i$. We follow similar steps (Equation 7.3 – Equation 7.5) 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 [171]. Then, we find the coefficient of variation (C_i^M) of the merged traffic from queues Q_d and Q_i as:

$$(C_i^M)^2 = \frac{1}{\lambda_{d_i} + \lambda_i} (\lambda_{d_i} (C_{d_i}^D)^2 + \lambda_i (C_i^D)^2) \tag{7.6}$$

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 consumed at the sink, we apply the traffic splitting method from [171] to approximate $C_{d_i}^A$ as:

$$(C_{d_i}^A)^2 = 1 + p_{d_i}((C_i^M)^2 - 1) \tag{7.7}$$

Finally, we extend the priority-aware formulations in continuous time domain [28] to discrete time domain to obtain the average waiting time of

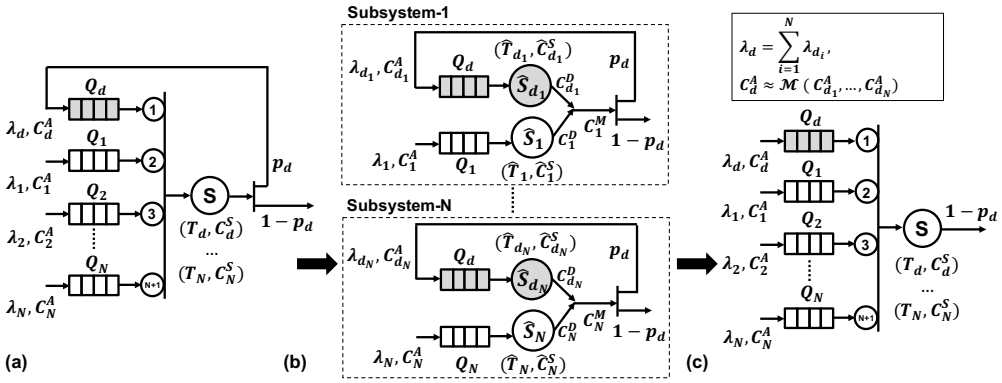


Figure 7.4: (a) Queuing system with N classes with deflection routing, (b) Decomposition into N subsystems to calculate GGeo parameters of deflected traffic per class, (c) Applying superposition to obtain the GGeo parameters of overall deflected traffic. $\mathcal{M}$ denotes the merging process.

the packets in Q_{d_i} and Q_i:

$$W_{d_i} = \frac{\rho_{d_i}(T_{d_i} - 1) + \rho_i(T_i - 1) + T_{d_i}((C^A_{d_i})^2 + \lambda_{d_i} - 1)}{2(1 - \rho_{d_i})} \tag{7.8}$$

$$W_i = \frac{\rho_{d_i}(T_{d_i} + 1) + 2\rho_{d_i}W_{d_i} + \rho_i(T_i - 1) + T_i((C^A_i)^2 + \lambda_i - 1)}{2(1 - \rho_i - \rho_{d_i})} \tag{7.9}$$

Queuing System with Multiple Traffic Classes

The analytical model for the system with a single class presented in Section 7.2 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 7.2.

Figure 7.4(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 (i) have higher priority than those routed to port (j) 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 7.4(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 7.2*. Hence, we first compute λ_{d_i} and $C^A_{d_i}$ of each subsystem-i following the procedure described in Section 7.2. Subsequently, we apply the superposition principle to λ_{d_i} and $C^A_{d_i}$ for $i = 1 \dots N$ to obtain the GGeo distribution parameters of the deflected traffic (λ_d, C^A_d).

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, $(\lambda_j = 0, j = 1 \dots N, j \neq i)$. The values of λ_{d_i} and $C^A_{d_i}$ can be expressed as:

$$\lambda_{d_i} = \lambda_d \Big|_{\lambda_j = 0, j \neq i; \lambda_i > 0} \quad \text{and} \quad C^A_{d_i} = C^A_d \Big|_{\lambda_j = 0, j \neq i; \lambda_i > 0} \tag{7.10}$$

Subsequently, we apply the principle of superposition to obtain the distribution parameters of Q_d as shown in Figure 7.4(c). First, we compute λ_d

by adding all λ_{d_i} as:

$$\lambda_d = \sum_{i=1}^{N} \lambda_{d_i} \tag{7.11}$$

The value of C_d^A is approximated by applying the superposition-based traffic merging process [171] for each $C_{d_i}^A$, as shown below:

$$(C_d^A)^2 = \sum_{i=1}^{N} \frac{\lambda_{d_i}}{\lambda_d} (C_{d_i}^A)^2 \tag{7.12}$$

Next, we use these distribution parameters (λ_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) [19]:

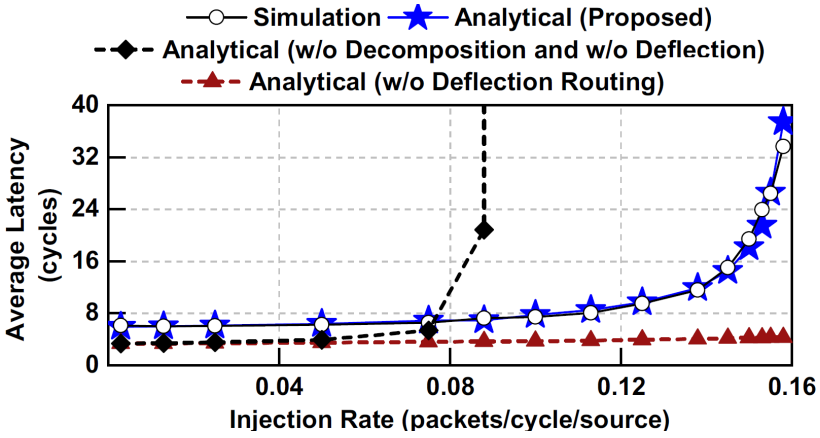


Figure 7.5: Comparison of average latency between simulation and analytical model for the canonical example shown in Figure 7.4 with $p_d = 0.3$ and $N = 5$.

$$W_i = \frac{\rho_d(T_d+1)+2\rho_i W_d}{2(1-\rho_d-\sum_{n=1}^{i}\rho_n)} + \frac{\sum_{n=1}^{i-1}(\rho_n(T_n+1)+2\rho_i W_n)}{2(1-\rho_d-\sum_{n=1}^{i}\rho_n)} + \frac{\rho_i(T_i-1)+T_i((C_i^A)^2+\lambda_i-1)}{2(1-\rho_d-\sum_{n=1}^{i}\rho_n)} \quad (7.13)$$

The first term in Equation 7.13 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 [135] to obtain the queuing time of different classes.

Figure 7.5 shows the average latency comparison between the proposed analytical model and simulation for the system in Figure 7.4. 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 [102] highly overestimates the latency as it does not consider multiple traffic classes. The performance model of the priority-aware NoC in [136] accounts for multiple traffic classes, but it does not model deflection. Hence, it severely underestimates the average latency.

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.

Algorithm 4: End-to-end latency computation

1 **Input:** NoC topology, routing algorithm, service process, input distribution for each class, (λ, C_A), deflection probability (p_d) for each sink
2 **Output:** Average end-to-end latency (L_{avg})
3 $\mathcal{S}$ = set of all classes in the network
4 N = number of queues in the network
5 $\mathcal{S}_n$ = set of classes in queue n
`/* Distribution of deflected traffic */`
6 **for** $i = 1$: $|\mathcal{S}|$ **do**
7 Compute λ_{di} and $C^A_{d_i}$ using Equation 7.10
8 Compute λ_d and C^A_d using Equation 7.11 and Equation 7.12
9 **end**
10 Compute W_d using λ_d and C^A_d
`/* Average waiting time of each class */`
11 **for** $n = 1$:N **do**
12 **for** $s = 1$:$|\mathcal{S}_n|$ **do**
13 Compute W_{ns} using Equation 7.13 (if $|\mathcal{S}_n| = 1$)
14 Compute W_{ns} following the decomposition method in [135] (if $|\mathcal{S}_n| > 1$)
15 **end**
16 **end**
17 $L_{avg} = \frac{\sum_{n=1}^{N}\sum_{s=1}^{\mathcal{S}_n}(W_{ns}+L_{ns})\lambda_{ns}}{\sum_{n=1}^{N}\sum_{s=1}^{\mathcal{S}_n}\lambda_{ns}}$ (*For mesh this term includes the latency both on the rows and the columns.*)

End-to-End latency computation: Algorithm 4 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 7.4(b) and λ_{d_i} and $C^A_{d_i}$ for each subsystem-i are computed. Subsequently, the proposed superposition methodology is applied to compute λ_d and C^A_d, shown in lines 6–9 of the algorithm. Then, λ_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.

7.3 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 [88] and high-end client CPUs [179], 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 [198] and special-purpose [208] 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 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 7.6 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.

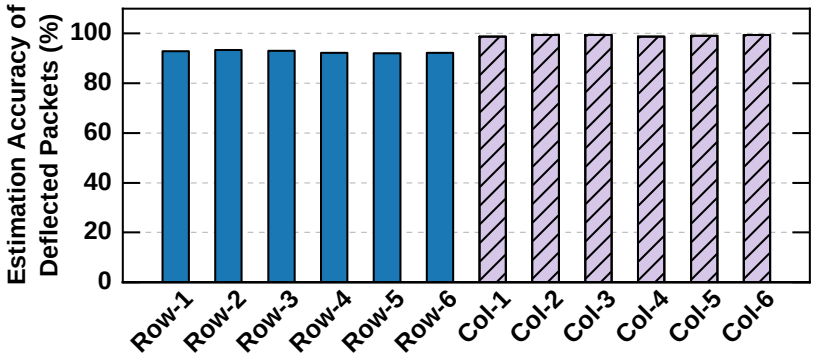


Figure 7.6: Estimation accuracy of average number of packets deflected for each row and column in a 6×6 mesh with p_d=0.3.

Table 7.2: Validation of the proposed analytical model for 6×6 mesh and 6×1 ring with bursty traffic arrival, and comparisons against prior work [102, 136]. '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[102]	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[136]	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

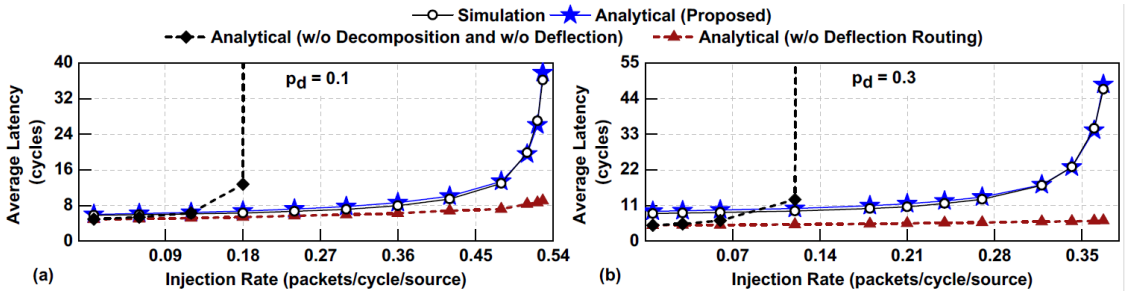


Figure 7.7: Comparison of average latency between simulation, the analytical model proposed in this work, and analytical models proposed in [102, 136] for a 6×6 mesh with deflection probability (a) 0.1 and (b) 0.3.

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 [102, 136]. The model presented in [102] does not incorporate multiple traffic classes and deflection routing. On the other hand, the model presented in [136] 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 7.7(a) and Figure 7.7(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 [102] significantly overestimates the latency starting with moderate injection rates, since it does not consider multiple traffic classes. Its performance degrades even

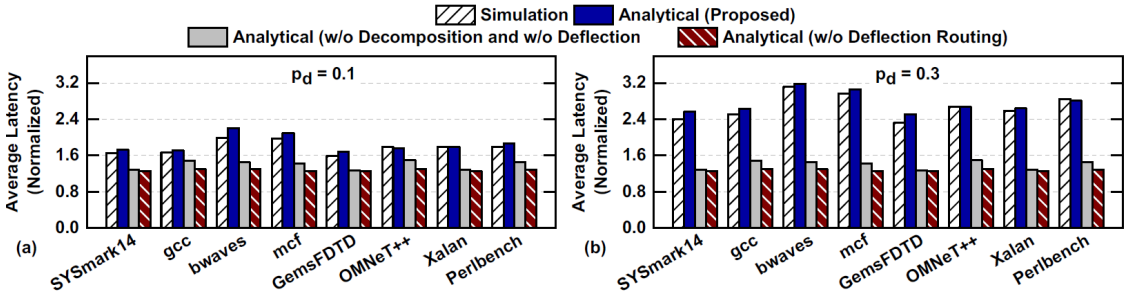


Figure 7.8: Average latency comparison between simulation, the analytical model proposed in this work, and analytical models proposed in [102, 136] for a 6×6 mesh with (a) $p_d = 0.1$ and (b) $p_d = 0.3$.

further with larger deflection probability, as depicted in Figure 7.7(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 [136] 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 7.7.

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 [88]. 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 [102] underestimates the latency at low injection rates and significantly overestimates it under high traffic load similar to the 6×6 results in Figure 7.7. Similarly, the analytical model presented in [136] 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 7.7.

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 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 (λ), 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 λ [106]. 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 [102, 136].

The estimation error of all three performance analysis techniques is reported in Table 7.2 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 [102] severely overestimate the latency similar to the results presented in Section 7.3. 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 [136] underestimates the latency because it does not model bursty traffic.

The right-hand side of Table 7.2 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 [102] overestimate the latency, whereas the models in [136] 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 [77], SPEC CPU®2017 benchmark suites [32], and the SYSmark®2014 application [15]. 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 7.3.

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 7.8. 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 7.8(a) and Figure 7.8(b), respectively. These plots show the average packet latencies normalized to the smallest latency from the 6×1 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 [102, 136] underestimate the latency, since the injection rates of these applications are in the range

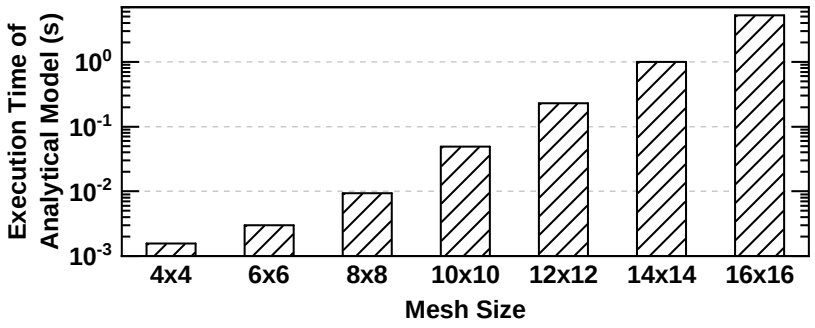


Figure 7.9: Execution time of the proposed analytical model (in seconds) for different mesh sizes.

of 0.02–0.1 flits/cycle/source (low injection region).

We observe similar results for the 6×1 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× 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 7.9 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 PERFORMANCE ANALYSIS OF NOCS WITH WEIGHTED ROUND ROBIN ARBITRATION

8.1 Background and Motivation

Networks-on-chip continue playing a central role as many-core processors start dominating the server [9, 56], deep learning [192, 141, 155] and medical industry [222]. As the commercial solutions scale up, the latency, area, and power consumption overheads of NoCs become increasingly crucial. Designers need analytical power-performance models to guide complex design decisions during the architecture development and implementation phases. After that, the same models are required by virtual platforms, commonly used to develop and evaluate the software ecosystem and applications [45]. 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 [152, 52, 148, 49]. Most 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 forward the incoming flits if they can and deflect otherwise. Bufferless NoCs save significant buffer area and enable ultra-fast, as low as single-cycle routing decisions [152, 52]. 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 [187, 56]. These solutions give priority to the packets already in the network to enable predictable and fast communication while stalling the newly gen-

erated 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 [163, 168]. Shared buffering resources, such as input and output channels, require arbitrating among different requesters. For example, suppose that packets in different input channels request the same output channel. An arbiter needs to resolve the conflicts and grant access to one of the requesters to meet performance target. The architectures proposed to date predominantly employ basic round-robin (RR) arbiter to provide fairness to all requesters [185, 122, 211]. Although the decisions are locally fair, the number of arbitrations a packet goes through grows with its path length. Hence, RR arbitration is globally unfair. 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 basic 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 [172, 76, 220]. Indeed, WRR arbitration enables higher throughput than RR arbitration [76]. *Despite its potential, WRR arbitration has not been*

analyzed theoretically, especially for large-scale NoCs. A large body of literature has proposed performance analysis techniques for buffered and bufferless NoCs since analytical models play a crucial role in fast design space exploration and pre-silicon evaluation [205, 174, 101, 135, 139, 14]. 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.

This work presents a fast, accurate, and scalable performance analysis technique for NoCs with WRR arbitration. To the best of our knowledge, it is the first performance analysis technique for NoCs with weighted round-robin arbitration. Furthermore, the proposed technique supports bursty core traffic observed in real applications, which is typically ignored due to its complexity. It first estimates the effective service time of the packets in the queue due to WRR arbitration. Then, it utilizes the effective service time to obtain the average waiting time of the packets. We also propose a decomposition technique to extend the analytical model for any size of NoC. Extensive experimental evaluations show that the proposed analytical model has less than 5% with real applications and 10% error with synthetic traffic having different burstiness levels congesting the NoC.

The major contributions of the work are listed below:

- A novel performance analysis technique for NoCs that employ WRR arbitration,
- A decomposition technique to obtain a scalable analytical model for NoC of any size.
- Experimental evaluations with multiple NoC configurations with different traffic scenarios showing less than 5% error for real applications.

Novel Contributions

Analytical models are required to estimate the NoC performance for fast design space exploration and pre-silicon evaluation. Multiple prior studies have proposed NoC performance analysis techniques with basic round-robin arbitration. [158, 63, 174]. Authors in [158] first construct a contention matrix between multiple flows in the NoC. Then, the average waiting time of the packets corresponding to each flow is computed. Support vector regression-based analytical model for NoCs is proposed in [174]. The analytical model proposed in [63] estimates the mean service time of the flows with RR arbitration. The estimated mean service time is used to find the average waiting time of the flows. However, none of these techniques are applicable in the presence of both bursty traffic and WRR arbitration.

Analytical modeling of round-robin arbitration has also been studied outside NoC domain [30, 71, 205]. The techniques presented in [30, 71] incorporate a polling model to approximate the effective service time of a queue in the presence of RR arbitration. However, none of these approaches are applicable when the input distribution to the queue is not geometric. A Markov chain-based analytical model is proposed in [205] to account for bursty input traffic. However, the technique is not scalable for a network of queues. Moreover, none of these techniques are applicable for discrete-time queuing systems. Since each transaction in NoC happens at discrete clock cycles, the analytical models need to incorporate discrete-time queuing systems. The major drawbacks of the prior approaches are summarized in Table 8.1.

The basic round-robin arbitration cannot provide fairness when requesters have widely varying data rate requirements and priorities. Therefore, weighted round-robin arbitration, i.e., WRR, has been used in on-chip communication architectures [172, 76]. Qian et al. compute delay bounds for different channels with different weights to assign appropriate weight

Table 8.1: Comparison of prior research and our novel contribution.

Research	**Approach**	**WRR**	**Bursty Traffic**	**Scalable**	**Discrete Time**
Boxma et al. [30]	Polling model	**No**	No	Yes	No
Wim et al. [71]	Extended polling model	**No**	No	Yes	No
Wang et al. [205]	Markov chain	**No**	Yes	No	No
Fischer et al. [63]	Heuristic	**No**	No	Yes	No
Vanlerbergee et al. [199]	Moment generating function	**No**	No	No	Yes
This work	**Queue decomposition**	**Yes**	**Yes**	**Yes**	**Yes**

to each input channel of the NoC [172]. They show that WRR delivers better quality of service than NoCs with strict priority-based arbitration. Authors in [76] propose a WRR-based scheduling policy. The proposed technique assigns larger bandwidth to input channels with higher weights. It achieves higher throughput compared to round-robin arbitration. *Although WRR has shown promise, no analytical modeling approach exists for NoCs with WRR to date.*

This work presents the first performance analysis technique for NoCs with WRR arbitration. It fills an essential gap since WRR can address the shortcomings of priority-based bufferless NoC architectures and the basic round-robin arbitration. Furthermore, the proposed technique supports bursty traffic observed in real applications, which is typically ignored due to its complexity. Hence, it is a vital step towards comprehending the theoretical underpinnings of NoCs with WRR arbitration and enabling their deployment in industrial designs.

Background and Overview

Weighted Round-Robin Arbitration

This work uses weighted round-robin arbitration in the NoC routers. The basic operating principle of WRR arbitration is illustrated in Figure 8.1 for three traffic classes. Packets from each class are first written to a dedicated input queue (also known as a channel). Suppose there are N input queues $Q_1, Q_2, \ldots, Q_N$. WRR technique assigns Q_i a positive integer weight denoted as $\omega_i \in \mathbb{Z}^+$ for $1 \leqslant i \leqslant N$. The WRR arbiter serves up to ω_i consecutive packets from Q_i before moving to the next queue. If Q_i has less than ω_i packets, then WRR serves Q_i until it becomes empty. Then, the WRR arbiter serves the subsequent queues following the same principle. After a cycle is completed, the weights are reset to their initial values, as illustrated in Figure 8.1, and the same arbitration cycle is repeated. In NoCs with WRR arbitration, whenever two or more requesters compete for the same resource, they are arbitrated following WRR. WRR arbitration can be used both for arbitrating different virtual channels and different ports in the network.

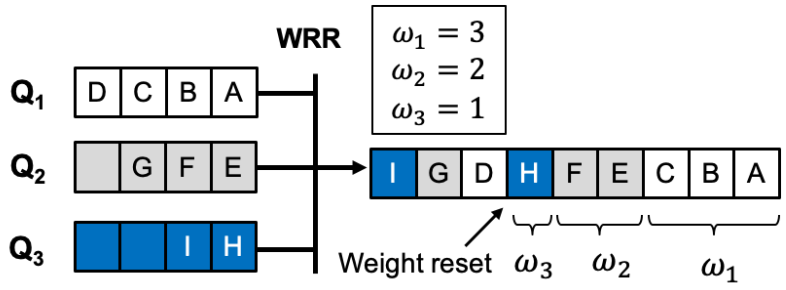


Figure 8.1: Illustration of weighted round-robin arbitration. 'A' is served first, 'I' last. In this example it is assumed that no new packets arrived until all prior packets (A-I) have been served.

Usage of the Proposed Performance Analysis Technique

WRR arbitration is promising for NoCs since it can tailor the communication bandwidth to different traffic classes. Furthermore, it provides 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 8×8 2D mesh with 5-port routers would have 320 WRR weights. Due to this ample design space, the current practice is limited to assigning two weights to each router (e.g., one weight to local ports and another weight 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 in the vast design space described in the previous paragraph. *This work focuses on constructing the proposed analysis technique and its evaluation against cycle-accurate simulation for two reasons.* First, the theoretical analysis is complex, and its evaluation deserves a dedicated treatment on its own. Moreover, its application for solving optimization problems requires demonstration of its fidelity first. Multi-objective optimization of the WRR weights using the proposed model is one of our future research directions. The remaining details of the work is described in Appendix E of the thesis and in the reference [144].

8.2 Proposed Methodology and Approach

Figure 8.2 shows a weighted round-robin arbiter with N queues. The packets that belong to traffic class-i are stored in queue Q_i. The corresponding average arrival rate of class-i packets is denoted by λ_i, as summarized in

Table 8.2. The burstiness of traffic class-i can be captured by their squared coefficient of variation of inter-arrival time, denoted by C_{Ai} [107, 172]. Finally, the weight assigned to class-i is denoted as $\omega_i \in \mathbb{Z}^+$ for $1 \leqslant i \leqslant N$, as illustrated in Figure 8.2.

To provide a step-by-step derivation, Section 8.2 starts with a particular case of the proposed technique tailored to the basic round-robin arbitration, i.e., all weights are set to one. Then, we extend the formulation to weighted round-robin arbitration.

Analytical Model for Basic Round-Robin Arbitration

In the basic round-robin arbitration, all weights are equal to one, i.e., $\omega_i = 1$, $1 \leqslant i \leqslant N$. Each traffic class experiences an additional delay until the arbiter serves the head packets in the other queues. The proposed technique has two steps:

1. The first step is to compute the first two moments of the *effective service time* for each class-i ($\hat{T}_i$, $\hat{C}_{Si}$) of the fully decomposed queue nodes illustrated in Figure 8.2.

2. In the second step, we use the transformed effective service times ($\hat{T}_i$, $\hat{C}_{Si}$, $1 \leqslant i \leqslant N$) to find the total waiting time (including the queuing

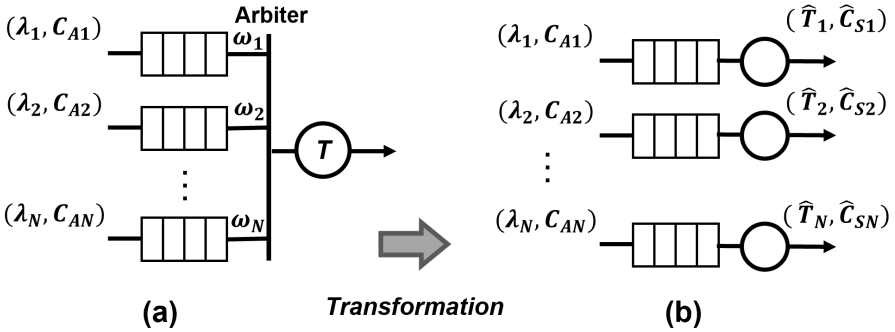


Figure 8.2: The original WRR arbiter with its traffic parameters (on the left) and a transformed WRR (on the right) that comprises fully decomposed queue nodes using our effective service time transformations.

Table 8.2: List of the important parameters used in this work.

N	Number of traffic classes to an arbiter
λ_i	Injection rate of class-i
ω_i	Weight assigned to class-i
T	Original mean service time of all traffic classes
$\widehat{T}_i$	Mean value of effective service time of class-i
ρ_i, $\widehat{\rho}_i$	$(\rho_i = \lambda_i T_i)$ and $(\widehat{\rho}_i = \lambda_i \widehat{T}_i)$ server utilizations of class-i
C_{Ai}	Squared coeff. of variation of inter-arrival time of class-i
C_S	Squared coeff. of variation of orig. service time (all classes)
$\widehat{C}_{Si}$	Squared coeff. of variation of eff. service time of class-i
C_A	Sq. coeff. of variation of inter-arrival time (merged traffic)
C_D	Squared coeff. of variation of inter-departure time
n_i	Mean queue occupancy of class-i
$p(n_i > k \mid \widehat{T}_j)$	Probability that queue occupancy of class-$i > k$ given a mean effective service time of class-$j = \widehat{T}_j$
W_i	Average waiting time of class-i
$\widehat{R}_i$	Mean effective residual service time of class-i

delay) of each traffic class.

For the clarity of notation and illustration, the derivation below assumes the original two moments of service time (T, C_S) are the same across all traffic classes.

Mean effective service time of RR $(\widehat{T}_i)$

The effective service time accounts for the delay experienced by the packets at the head of each queue. This delay includes its own service time and the service time of the packet at the head of other queues since the round-robin arbiter serves them one by one. When all queues have packets, class-i packets will be served every $N \times T$ cycles, i.e., they will wait for $(N-1) \times T$ cycles after being served before winning the arbitration again. In general, a traffic class will only contribute to the extra service time if it has a packet

waiting for service. Thus, assuming packet arrival of different classes are independent, we can express the mean effective service time as:

$$\begin{aligned}\widehat{T}_i &= T + \sum_{j=1, j\neq i}^{N} p(n_i > 0|\widehat{T}_i)p(n_j > 0|\widehat{T}_i)T \\ &= T + Tp(n_i > 0|\widehat{T}_i)\sum_{j=1, j\neq i}^{N} p(n_j > 0|\widehat{T}_i), 1 \leqslant i \leqslant N \end{aligned} \tag{8.1}$$

where $p(n_j > 0|\widehat{T}_i)$ denotes the probability that the occupancy of class-j is greater than zero given a mean effective service period of class-i is $\widehat{T}_i$, for $1 \leqslant i, j \leqslant N$. We approximate $p(n_j > 0|\widehat{T}_i)$ using Little's law as $\lambda_j \widehat{T}_i$ following the busy-cycle approach [30]. Little's law gives the expected number of class-j packets that arrive during the period of $\widehat{T}_i$, which approximates the probability that the occupancy of class-j packets is greater than zero during this period. To ensure that the probability term does not exceed one, we approximate $p(n_j > 0|\widehat{T}_i)$ as $min(1, \lambda_j \widehat{T}_i)$. Therefore, Equation 8.1 can be rewritten as:

$$\widehat{T}_i = T + Tmin(1, \lambda_i \widehat{T}_i) \sum_{\substack{j=1 \\ j\neq i}}^{N} min(1, \lambda_j \widehat{T}_i),\ 1 \leqslant i \leqslant N \tag{8.2}$$

We solve Equation 8.2 through a simple iterative approach described in Algorithm 5. First, Equation 8.2 is solved by removing the nonlinearity introduced by the min operation (lines 4 and 5). We consider the smaller solution out of the two solutions, since we observed through experiments that the smaller solution is more accurate. The solution is used as the initial estimate of the iterative approach. Then, this estimate is plugged into Equation 8.2 within an iteration loop to obtain a better estimate (line 7). The iterations continue until the change in the effective service time is within a user-provided $Tolerance$ value, set to 0.01 in this work. Algorithm 5 typically takes less than ten iterations due to the quadratic nature

Algorithm 5: Basic round-robin arbitration: Effective service time ($\widehat{T}_i$) computation.

Input: Injection rate of each class (λ), service time (T), number of classes (N), Convergence $Tolerance$
Output: Effective service time of each class ($\widehat{T}$)
for *i = 1:N* **do**
    Find smaller root of the quadratic equation derived from Equation 8.2:
    $T + T\lambda_i(\widehat{T}_i)^2 \sum_{j=1,j\neq i}^{N} \lambda_j - \widehat{T}_i = 0$
    $\delta = Tolerance, k = 1$
    while $\delta \geqslant Tolerance$ **do**
        $\widehat{T}_i^{k+1} \leftarrow T + Tmin(1, \lambda_i \widehat{T}_i^k) \sum_{\substack{j=1 \\ j\neq i}}^{N} min(1, \lambda_j \widehat{T}_i^k)$
        $\delta = \widehat{T}_i^{k+1} - \widehat{T}_i^k$
        $k \leftarrow k + 1$
    end
end

of Equation 8.2, which takes only a few microseconds to compute. After the iterations are completed, Algorithm 5 returns the effective service time $\widehat{T}_i$.

Coefficient of variation of effective service time for RR ($\widehat{C}_{Si}$)

This section derives $\widehat{C}_{Si}$ by leveraging the conservation of work principle and property of equal residual service time of individual classes [30]. We leverage these properties to compute the mean effective residual service time of class-i ($\widehat{R}_i$). Then, $\widehat{R}_i$ is used to calculate $\widehat{C}_{Si}$ using the relation found in [107]:

$$\widehat{C}_{Si} = \frac{1}{\widehat{\rho}_i}\left(\frac{2\widehat{R}_i}{\widehat{T}_i} + 1 - C_{Ai} - \widehat{\rho}_i\right) \tag{8.3}$$

where $\widehat{\rho}_i = \lambda_i \widehat{T}_i$ is the effective server utilization. In addition, we show that leveraging the conservation of work principle gives our model the capability of handling bursty traffic.

We exploit the *conservation of work principle*, stating that the same

amount of packets are served regardless of how the bandwidth is divided between the traffic flows assuming all packets have same moments of service time. Due to this fundamental principle, the total queue occupancy n_{sum} is invariant under the arbitration policy. Furthermore, this sum can be computed with near-perfect accuracy using maximum entropy-based analytical queuing model of multi-class given in [107]. One strong aspect of this model is that it can take bursty traffic that conforms to General Geometric distribution (GGeo). By leveraging this model, we can embed the capability of handling bursty traffic into our analytical model. The total queue occupancy (n_{sum}) can be calculated as:

$$n_{sum} = \frac{1}{2}\sum_{i=1}^{N}\left(\rho_i(C_{Ai}-1) + \frac{\sum_{k=1}^{N}\frac{\lambda_i}{\lambda_k}\rho_k^2(C_{Ak}+C_S)}{1-\sum_{k=1}^{N}\rho_k}\right) \tag{8.4}$$

using the parameters summarized in Table 8.2. We do not delve into the derivation of this equation for clarity since it is not our major focus. The detailed derivation can be found in the seminal work by Kouvatsos et al. [107].

Next, we derive n_{sum} using the first two moments of transformed effective service time. The occupancy of each queue Q_i can be expressed using Little's law as: $n_i = \lambda_i W_i$, where W_i is the average waiting time of class-i. Therefore, we obtain:

$$n_{sum} = \sum_{i=1}^{N}\lambda_i W_i \tag{8.5}$$

The average waiting time W_i consists of two components. The first one is the waiting time in the queue till the packet reaches the head of the queue ($\widehat{W_i}$), i.e. until all other class-i packets leave the server completely. The second component accounts for the waiting time at the head of the queue when the server is busy serving other classes (class-j, $j \neq i$). This

component is the additional time captured by the effective service time $\hat{T}_i$, i.e., the difference between the effective and original service times: $\Delta T_i = \widehat{T}_i - T$. Furthermore, the waiting $\widehat{W_i}$ is expressed as a function of the residual time [19]. Hence,

$$W_i = \frac{\widehat{R}_i}{1 - \lambda_i \widehat{T}_i} + \Delta T_i, \text{ for } 1 \leqslant i \leqslant N \tag{8.6}$$

We leverage the property that all individual mean effective residual service time of classes are equal [30]; hence we write ($R = \widehat{R}_i, 1 \leqslant i \leqslant N$). Thus, we can rewrite the total occupancy by plugging Equation 8.6 into Equation 8.5 and then solve for the residual time R:

$$\begin{aligned} n_{sum} &= \sum_{i=1}^{N} \lambda_i \Big(\frac{R}{1 - \lambda_i \widehat{T}_i} + \Delta T_i \Big) \\ \implies R &= \Big(n_{sum} - \sum_{i=1}^{N} \lambda_i \Delta T_i \Big) \Big(\sum_{i=1}^{N} \frac{\lambda_i}{1 - \lambda_i \widehat{T}_i} \Big)^{-1} \end{aligned} \tag{8.7}$$

Then, we can compute the coefficient of variation $\widehat{C}_{Si}$ by substituting the residual time R from Equation 8.7 in Equation 8.3.

Average waiting time of RR (W_i)

Finally, we compute the mean waiting time of individual classes by plugging $\widehat{T}_i$ and $\widehat{C}_{Si}$ found for RR arbitration into Equation 8.3 and Equation 8.6 as:

$$W_i = \frac{0.5 \widehat{T}_i (\widehat{\rho}_i - 1 + C_{Ai} + \widehat{\rho}_i \widehat{C}_{Si})}{1 - \lambda_i \widehat{T}_i} + \Delta T_i, \; 1 \leqslant i \leqslant N \tag{8.8}$$

Analytical Model for Weighted Round-Robin Arbitration

This section extends the analytical model constructed in Section 8.2 to WRR using the same steps. It first describes the derivation of the first two

moments of effective service time. Then, it will use these moments to find the average waiting time.

Mean effective service time of WRR ($\widehat{T}_i$)

In WRR, the weight of each class can be larger than or equal to one, i.e., $\omega_i \geqslant 1$, $1 \leqslant i \leqslant N$. To illustrate the generalization, we start with a system of two queues, where $\omega_1 > 1$ and $\omega_2 = 1$. Then, we generalize it to N queues with arbitrary weights.

Mean effective service time of ***class-1*** *with* $\omega_1 > 1$ *and* $\omega_2 = 1$: Since $\omega_1 > 1$, a batch of up to ω_1 number of class-1 packets are served without interruption. Let $\widehat{T}_i^b$ be the total service time of ω_1 class-1 packets. When there is a class-2 packet in Q_2, the next batch of ω_1 class-1 packet will wait. Therefore, we can express $\widehat{T}_1^b$ as:

$$\widehat{T}_1^b = \omega_1 T + p(n_1 > 0|\widehat{T}_1^b)p(n_2 > 0|\widehat{T}_1^b)T \tag{8.9}$$

Note that this equation generalizes Equation 8.1 to arbitrary ω_1 for two queues. Using Little's law and the methodology described in Section 8.2, we approximate $\widehat{T}_1^b$ as:

$$\widehat{T}_1^b = \omega_1 T + \frac{T}{\omega_1} min(1, \lambda_1 \widehat{T}_1^b) min(1, \lambda_2 \widehat{T}_1^b) \tag{8.10}$$

Mean effective service time of ***class-2*** *with* $\omega_1 > 1$ *and* $\omega_2 = 1$: The batch size of class-2 packet is one in this case since $\omega_2 = 1$. They need to wait in the queue for a maximum of ω_1 class-1 packets. Therefore, $\widehat{T}_2$ is expressed as:

$$\widehat{T}_2 = T + \sum_{i=0}^{\omega_1 - 1} p(n_1 > i|\widehat{T}_2)p(n_2 > 0|\widehat{T}_2)T \tag{8.11}$$

We can approximate $p(n_1 > i|\widehat{T}_2)$ as $\frac{1}{i+1}p(n_1 > 0|\widehat{T}_2)$ to obtain:

$$\widehat{T}_2 = T + \sum_{i=0}^{\omega_1 - 1} \frac{1}{i+1} p(n_1 > 0 | \widehat{T}_2) p(n_2 > 0 | \widehat{T}_2) T$$

$$= T + T min\Big(1, \sum_{i=0}^{\omega_1 - 1} \frac{1}{i+1} \lambda_1 \widehat{T}_2\Big) min(1, \lambda_2 \widehat{T}_2) \tag{8.12}$$

To get to the final form of Equation 8.12 above, we approximate the conditional probabilities based on [30].

Generalization to N *queues and arbitrary weights:* In general, there are N classes with weights $\omega_i, 1 \leqslant i \leqslant N$. Consider the class-i packets with the total service time $\widehat{T}_i^b$ for a batch of ω_i packets. The first part of the effective service time will be due to the *packets of same class*. This effect is captured in our two-queue illustration for class-1 packets in Equation 8.9. The second part of the effective service time will be due to the *packets of other classes*. This effect is captured in our two-queue illustration for class-2 packets in Equation 8.11. By combining them, we can express the generalized effective service time as:

$$\widehat{T}_i^b = \omega_i T + p(n_i > 0 | \widehat{T}_i^b) \sum_{j=1, j \neq i}^{N} \Big(\sum_{k=0}^{\omega_j - 1} p(n_j > k | \widehat{T}_i^b) \Big) T \tag{8.13}$$

Note that this equation reduces to Equation 8.9 for $i = 1, N = 2, \omega_2 = 1$, and to Equation 8.11 for $i = 2, N = 2, \omega_2 = 1$.

Finally, we can obtain the generalized effective service times by replacing $p(n_i > 0)$ with $min(1, \lambda_i \widehat{T}_i^b)$ as follows:

$$\widehat{T}_i^b = \omega_i T + \frac{1}{\omega_i} min(1, \lambda_i \widehat{T}_i^b) \sum_{\substack{j=1 \\ j \neq i}}^{N} min\Big(1, \sum_{k=0}^{\omega_j - 1} \frac{1}{k+1} \lambda_j \widehat{T}_i^b\Big)$$

$$\widehat{T}_i = \frac{\widehat{T}_i^b}{\omega_i} \tag{8.14}$$

Due to the non-linearity introduced by the min operation, we compute $\widehat{T}_i$ using the iterative Algorithm 6 just like the basic round-robin case. The

Algorithm 6: Obtaining effective service time ($\widehat{T}_i$) of weighted round-robin

Input: Injection rate of each class (λ), WRR weights (ω_i), service time (T), number of classes (N)
Output: Effective service time of each class ($\widehat{T}$)
for $i = 1{:}N$ **do**
  Find smaller root of the quadratic equation derived from Equation 8.14 for $\widehat{T}_i^b$:
  $(\widehat{T}_i^b)^2 \frac{1}{\omega_i}\lambda_i \sum_{\substack{j=1 \\ j\neq i}}^{N} \sum_{k=0}^{\omega_j - 1} \frac{1}{k+1}\lambda_j - \widehat{T}_i^b + \omega_i T = 0$
  $\delta = Tolerance, k = 1$
  while $\delta \geqslant Tolerance$ **do**
    $Z = \sum_{\substack{j=1 \\ j\neq i}}^{N} min\left(1, \sum_{k=0}^{\omega_j - 1} \frac{1}{k+1}\lambda_j \widehat{T}_i^{b,k}\right)$
    $\widehat{T}_i^{b,k+1} \leftarrow \omega_i T + \frac{1}{\omega_i} min(1, \lambda_i \widehat{T}_i^{b,k}) Z$
    $\delta = \widehat{T}_i^{b,k+1} - \widehat{T}_i^{b,k}$
    $k \leftarrow k + 1$
  end
  $\widehat{T}_i = \frac{\widehat{T}_i^b}{\omega_i}$
end

quadratic nature of Equation 8.14 enables a fast convergence within 10 iterations under 5µs.

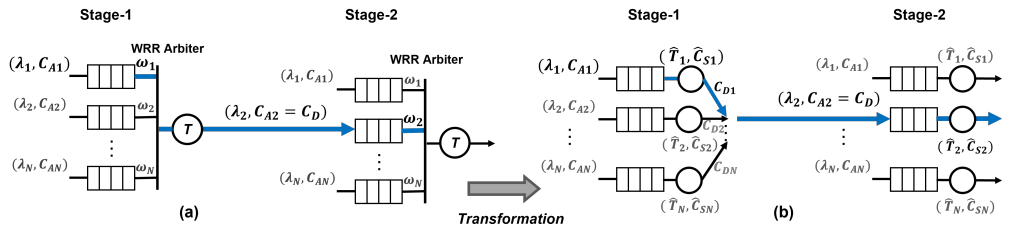


Figure 8.3: Illustration of extending the proposed analysis to multiple stages. Only two consecutive stages are shown for clarity. The departure statistics at a given stage become the arrival statistics of the subsequent stage. The blue line denotes that the class which wins the arbitration goes to the next stage.

Coefficient of variation of effective service time of WRR ($\widehat{C}_{Si}$) [1]

In the basic round-robin case, the residual service time R_i accounts for the second-order moment of the service time, i.e., C_{Si}. However, effective mean residual service time of different classes are not necessarily equal to each other for arbitrary weights, which is the case in WRR arbitration. The mean effective residual service time of class-i packets is expressed as:

$$\widehat{R}_i = 0.5\widehat{T}_i(\widehat{\rho}_i - 1 + C_{Ai} + \widehat{\rho}_i\widehat{C}_{Si}) \tag{8.15}$$

where $\widehat{T}_i$ is found from Algorithm 6 based on Equation 8.14. Note that the residual service time depends on the first two moments of service time, which are the average service time $\widehat{T}_i$ and the coefficient of variation $\widehat{C}_{Si}$, respectively. To approximate $\widehat{C}_{Si}$ for WRR, we leverage its monotonically decreasing behavior as a function of the corresponding weight. This behavior is expected since a larger weight corresponds to fewer arbitration stages, decreasing the effective service time variation. This observation indicates that the coefficient of variation for round-robin arbitration is an upper bound of coefficient of variation for weighted round-robin arbitration. This upper bound, $\widehat{C}_{Si}(RR)$ for class-i, is computed using Equation 8.3. Given this, we first apply linear approximation to calculate the $\widehat{C}_{Si}$ under WRR as $\alpha\frac{\widehat{C}_{Si}(RR)}{\omega_i^2}$, note we use the squared value of ω_i since $\widehat{C}_{Si}$ is the square of coefficient of variation as defined in Table 8.2. We add the parameter α to tighten the approximation. Next, we recall Equation 8.4 and replace C_{Sk} by $\alpha\frac{\widehat{C}_{Sk}(RR)}{\omega_k^2}$ to obtain:

$$n_{sum} = \frac{1}{2}\sum_{i=1}^{N}\left(\widehat{\rho}_i(C_{Ai} - 1) + \frac{\sum_{k=1}^{N}(\frac{\lambda_i}{\lambda_k})\widehat{\rho}_k^2(C_{Ak} + \alpha\frac{\widehat{C}_{Sk}(RR)}{\omega_k^2})}{1 - \sum_{k=1}^{N}\widehat{\rho}_k}\right) \tag{8.16}$$

[1] We use the same notation for RR and WRR arbitration since WRR results are generalizations of basic RR expressions.

The left hand side of the Equation 8.16 (n_{sum}) is obtained by applying Equation 8.4. Therefore, α is the only unknown in Equation 8.16. Once we obtain α (by solving Equation 8.16), we compute $\widehat{C}_{Si}$ under WRR as $\alpha\frac{\widehat{C}_{Si(RR)}}{\omega_i^2}$. This expression captures traffic burstiness addressing one of the significant drawbacks of the prior work.

Average waiting time of WRR (W_i)

So far, we obtained first two moments of effective service time, i.e., $\widehat{T}_i$ (Equation 8.14) and $\widehat{C}_{Si}$. We obtain the average waiting time under WRR arbitration by plugging them to Equation 8.17:

$$W_i = \frac{0.5\widehat{T}_i(\widehat{\rho}_i - 1 + C_{Ai} + \widehat{\rho}_i\widehat{C}_{Si})}{1 - \lambda_i\widehat{T}_i} + \Delta T_i \tag{8.17}$$

Estimation of end-to-end latency

The previous section described the canonical model for WRR arbitration, where all traffic classes go through a single arbiter. The packets in NoCs go through a sequence of WRR arbiters while traveling from their sources towards destinations. For example, an 8×8 2D mesh has 64 routers with at least one arbiter per output queue in each of them. Figure 8.3(a) illustrates an example with two stages, where the flow that wins the first stage is routed to the second one. The major challenge in modeling multiple stages is obtaining the inter-departure distribution, which becomes the arrival flow at the subsequent stages. For instance, the coefficient of variation of arrival flow 2 in the second stage is equal to the coefficient of variation of the departing flow from stage-1 (C_D) in Figure 8.3(a). We first find the squared co-efficient of variation of inter-departure time for each traffic class (C_{Di}):

$$C_{Di} = \rho_i^2(\widehat{C}_{Si} + 1) + (1 - \rho_i)C_{Ai} + \rho_i(1 - 2\rho_i), \forall 1 \leqslant i \leqslant N \tag{8.18}$$

Algorithm 7: Estimation of end-to-end latency

Input: NoC size ($n \times m$), injection rates and co-efficient of variation of inter-arrival times for all classes (λ, C_A), Service time (T), WRR weights
Output: Average NoC ($\bar{L}$) and source–dest. latencies (L_{sd})
Number of routers $N_p \leftarrow n \times m$, $\lambda_{sum} \leftarrow 0$, $L_{sum} \leftarrow 0$
for *s = 1:*N_p **do**
    for *d = 1:*N_p **do**
        $W_{sd} = compute\_waiting\_time(s, d)$
        $L_{sd} = W_{sd} + free\_packet\_delay(s, d)$
        $L_{sum} \leftarrow L_{sum} + L_{sd}$, $\lambda_{sum} \leftarrow \lambda_{sum} + \lambda_{sd}$
    end
end
$\bar{L} = \frac{L_{sum}}{\lambda_{sum}}$
function *compute_waiting_time*(s, d)
    $S \leftarrow$ The number of stages for the flow with source s and destination d, $W_{sd} \leftarrow 0$
    for *j = 1:S* **do**
        $W_j =$ Waiting time at stage j using Equation 8.17
        $W_{sd} \leftarrow W_{sd} + W_j$
        $C_D =$ Coefficient of variation using Equation 8.19
        Use C_D as the C_A in the next stage
    end
    return W_{sd}
end

Next, we apply the decomposition technique [171] to obtain the departure distribution at each stage (C_D).

$$C_D = \frac{\sum_{i=1}^{N} \lambda_i C_{Di}}{\sum_{i=1}^{N} \lambda_i} \tag{8.19}$$

The departure distribution (C_D) is the arrival distribution to the next stage (C_{A2}). For a given source-destination pair, these calculations are performed at each arbitration stage on the path of the packets.

Algorithm 7 presents a step-by-step procedure to obtain end-to-end latency of a given NoC with WRR arbitration and deterministic routing.

The input to the algorithm is the NoC size, injection rate, and coefficient of variation of inter-arrival time of all classes, service time of the queues, and weights assigned to each arbiter. The output of the algorithm is the latency of each source-destination pair (L_{sd}) as well as the average latency $\bar{L}$.

Algorithm 7 first computes the waiting time of each source-destination pair (W_{sd}). The function *compute_waiting_time* in lines 12–22 describes the procedure for this computation. The procedure starts with finding the number of stages (S) for the flow from source s to destination d. At each stage, it computes the average waiting time using Equation 8.17 (line 16). Next, it adds the waiting time at the current state to the cumulative waiting time found so far (line 17). Then, it computes the coefficient of variation of inter-departure time (C_D) using Equation 8.18 and C_D is used as the coefficient of variation of inter-arrival time in the next stage (lines 18–19). Finally, the procedure returns the total waiting time from source s to destination d (W_{sd}) (line 21). After retrieving the total waiting times, Algorithm 7 adds the free packet delay (t_{sd}), i.e., the latency due to the links and router micro-architecture, which can be found using the number of hops and the router pipeline depth [162, 49]. Then, end-to-end latencies and flow rates are accumulated (line 8). Finally, the average latency of the network is obtained by computing the weighted average of the latency of all source-destination pairs (line 11). We note that different WRR arbiters in the network can be assigned different arbitration weights.

8.3 Experimental Results

Experimental Setup

This section presents a thorough evaluation of the proposed analytical modeling approach under various traffic scenarios and NoC configurations. We employ geometric and bursty traffic distributions in addition to real

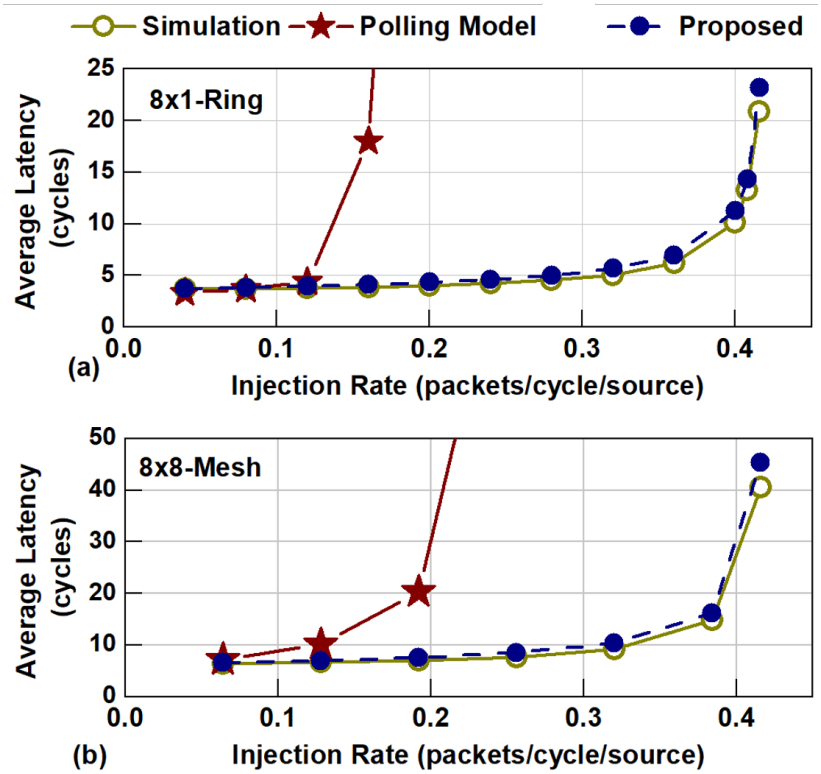


Figure 8.4: Verification of the analytical model for basic round-robin with (a) 8×1 and (b) 8×8 NoC.

application traces feeding 1×8, 6×6, and 8×8 NoCs (note that the typical size of state-of-the-art industrial NoCs is 6×6 [187, 56]). The synthetic traffic simulations use 100% last-level cache (LLC) hit and 100% LLC miss scenarios, commonly used corner cases. Furthermore, applications from SPEC CPU 2017 [32] and PARSEC [23] benchmark suits demonstrate the effectiveness of the proposed technique in a broader range of scenarios. The latency results of our analytical models are compared against an in-house cycle-accurate simulator, which is calibrated with an industrial simulator [159]. All simulations run for 200K cycles to reach a steady state and have 20K cycles of warm-up. The source code of the simulator and the analytical model are publicly released at [191].

Results with Basic Round-Robin Arbitration

We first evaluate the proposed analytical model under basic round-robin arbitration. Figure 8.4 depicts representative average end-to-end latency comparisons between analytical model and the cycle-accurate simulations for 8×1 ring and 8×8 mesh NoC. The average injection rates for both NoC configurations are varied until the NoC becomes highly congested, following geometric distribution. On average, the proposed technique incurs only 5% error for 8×1 ring and 7% error for 8×8 mesh. We also compare the proposed analytical model with a polling-based model for round-robin arbitration [71]. The polling-based model grossly overestimates the latency both for 8×1 and 8×8 NoC. A comprehensive summary for other NoC sizes and traffic patterns is presented in Section 8.3. These results demonstrate that the proposed approach is accurate for difference NoC, traffic, and WRR parameters.

Results with Weighted Round-Robin Arbitration

The most important aspect of the proposed technique is considering WRR arbitration. Figure 8.5 shows the average end-to-end latency comparison of our analytical model against cycle-accurate simulations with an 8×8 mesh for three different weight configurations. The proposed analytical model incurs on average 8% error when the WRR weights associated with the traffic channels of external input to the NoC and channels connected to internal routers are 1 and 2, respectively. The average error slightly increases to 9% when the arbitration weights of the packets in the NoC increase to 3. This configuration decreases the network congestion by providing higher priority to the packets already in the NoC. Hence, the average waiting time decreases. Since there are no other analysis approaches for WRR, this section does not provide comparisons to state-of-the-art.

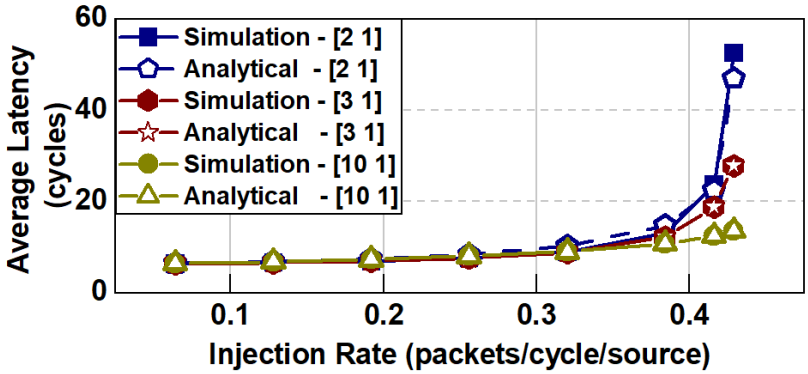


Figure 8.5: Verification of the analytical model for weighted round-robin with 8×8 NoC. [x y] denotes that the WRR weights associated with channels connected to the internal routers is x and traffic channels of external input to the NoC is y.

Results with Bursty Traffic and Real Applications

This section evaluates the proposed model in the presence of bursty traffic. We define probability of burstiness as p_{burst} of the general geometric distribution [107]. Increasing p_{burst} denotes increasing burstiness in the

Table 8.3: Summary of results for synthetic applications with **100% hit**.

Topo.	$\omega_{ring}=1, \omega_{src}=1$				$\omega_{ring}=3, \omega_{src}=1$			
	$p_{burst}=0$		$p_{burst}=0.3$		$p_{burst}=0$		$p_{burst}=0.3$	
	$\lambda=0.1$	$\lambda=0.3$	$\lambda=0.1$	$\lambda=0.3$	$\lambda=0.1$	$\lambda=0.3$	$\lambda=0.1$	$\lambda=0.3$
8×1	1.4%	11%	3.6%	7.8%	1.5%	13%	9%	11%
6×6	5.5%	7.2%	7.4%	11%	5.2%	11%	5.9%	12%
8×8	2.6%	7.8%	5.2%	10%	3.5%	11%	4.8%	7.2%

Table 8.4: Summary of results for synthetic applications with **100% miss**.

Topo.	$\omega_{ring}=1, \omega_{src}=1$				$\omega_{ring}=3, \omega_{src}=1$			
	$p_{burst}=0$		$p_{burst}=0.3$		$p_{burst}=0$		$p_{burst}=0.3$	
	$\lambda=0.1$	$\lambda=0.2$	$\lambda=0.1$	$\lambda=0.2$	$\lambda=0.1$	$\lambda=0.2$	$\lambda=0.1$	$\lambda=0.2$
8×1	4.3%	4.6%	5.3%	5.7%	4.4%	4.6%	4.5%	4.8%
6×6	5.0%	5.4%	5.5%	6.0%	5.0%	5.4%	5.0%	5.8%
8×8	7.1%	8.0%	7.4%	7.6%	7.4%	13%	8.1%	9.0%

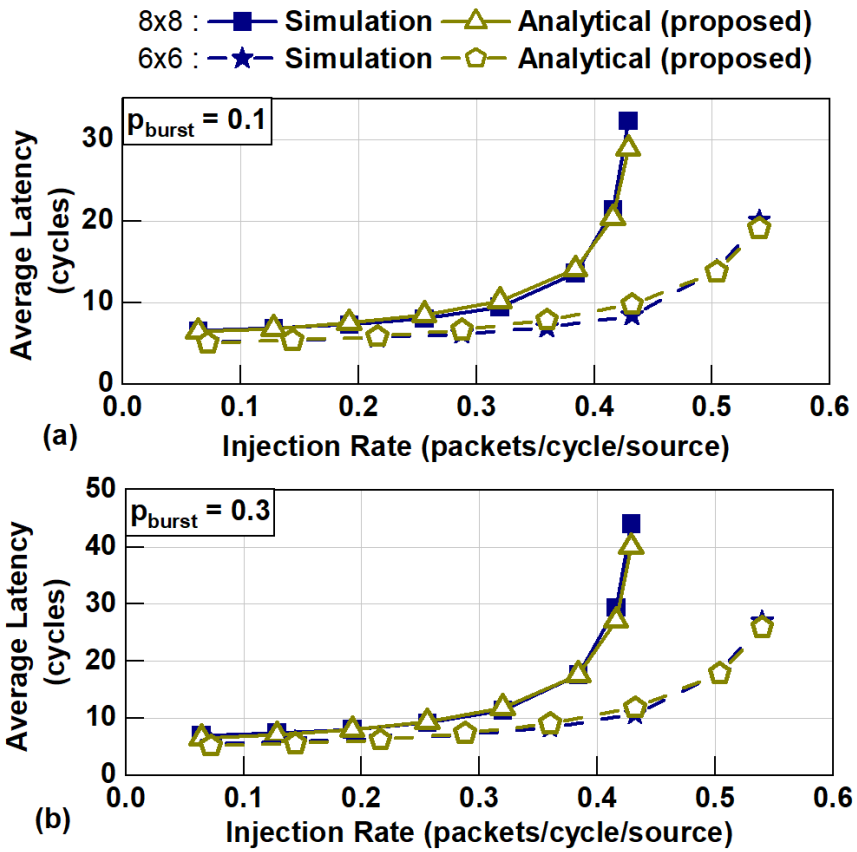


Figure 8.6: Verification of analytical model for bursty traffic with (a) $p_{burst} = 0.1$ and $p_{burst} = 0.3$.

traffic. We note that no other prior work considered weighted round-robin arbitration together with bursty traffic. Figure 8.6 compares the end-to-end average latency results against cycle-accurate simulations for 6×6 and 8×8 mesh when the arbitration weights associated with the packets in the NoC and external packets arrivals are 3 and 1, respectively. When the burst probability is $p_{burst} = 0.1$ (Figure 8.6(a)), the average modeling errors for 6×6 and 8×8 mesh are 7% and 4%, respectively. When the burst probability increases to 0.3 (Figure 8.6(b)), the average errors become 6% and 5%, respectively.

Finally, we evaluate the proposed analysis technique while running applications from the SPEC CPU2017 [32] and PARSEC benchmark [23].

The applications from SPEC CPU2017 benchmarks show burstiness in the range of $p_{burst} = 0.1 - 0.6$. The applications from PARSEC benchmark are 16-threaded and do not show any burstiness. Figure 8.7 compares the analysis results against cycle-accurate simulations on an 8×8 NoC. The WRR weights of packets already in the NoC are set to 3, while new packets from cores have a lower priority set with a weight of 1. We observe that our proposed analysis technique consistently achieves a modeling error of less than 5% for these applications. These results demonstrate that the proposed analysis technique can accurately model the end-to-end NoC latency under WRR arbitration and bursty traffic.

Summary of the Evaluation Results

This section summarizes the accuracy of our performance analysis technique systematically for different NoC sizes, WRR weights, and traffic burstiness. To capture the most commonly observed corner cases, we provide results for two extreme cases – 100% LLC hit and 100% LLC miss traffic. In both cases, cores send packets to each LLC at an equal rate.

Table 8.3 summarizes the comparisons between analysis and simulations for 100% LLC hit. When $p_{burst} = 0$ and the traffic load is moderate ($\lambda = 0.1$), the modeling error ranges from 1.4% to 5.5% considering both RR and WRR arbitration. Even when the traffic load increases congesting the NoC ($\lambda = 0.3$), the modeling error remains below 11% for all configurations. A higher level of burstiness to $p_{burst} = 0.3$ increases the NoC load. Consequently, the error range increases 3.6%–9.0% for the moderate traffic load ($\lambda = 0.1$). Increasing the load further congests the NoC ($\lambda = 0.3$) pushing the worst-case error only to 13%, which is acceptable since practical systems hardly operate this load due to congestion control mechanisms.

Finally, Table 8.4 summarizes the modeling error of the proposed technique for 100% LLC miss. In this case, the missed core requests are for-

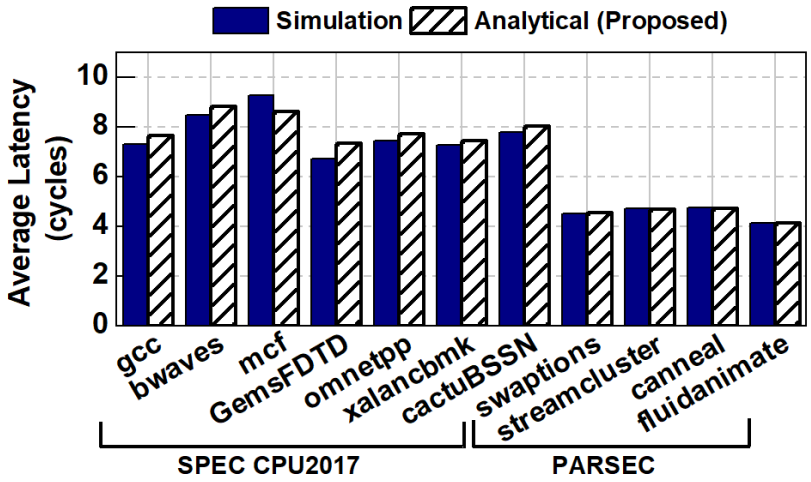


Figure 8.7: Verification against real applications.

Table 8.5: Analysis on execution time of the proposed model.

NoC size	4×4	6×6	8×8	16×16	32×32
Exe. time (ms)	0.56	1.41	2.25	13.52	45.07

warded to memory controllers (one in 8×1 NoC and two in 6×6 and 8×8 NoCs). Hence, the memory controllers become hotspots. Despite the high degree of skewness, the error for moderate NoC loads ranges from 4.3% to 8.1% for all burstiness levels, WRR weight configurations, and NoC sizes. Even when the NoC is pushed to congestion, the modeling error remains under 9.0% for all configurations, except for 8×8 NoC with WRR weights 3 and 1. The error in this specific case is 13%; the error for 8×1 and 6×6 NoCs are 4.6% and 5.4%, respectively.

Execution Time of the Proposed Analysis Technique

We implemented the proposed technique in C++ to obtain end-to-end latency. The computational complexity of the proposed model is $O(n^2 \times m^2 \times p)$, for $n \times m$ NoC ($n \geqslant m$) and WRR arbiters with maximum p ports.

The execution time as a function of NoC sizes with 3 output ports per router are summarized in Table 8.5. We observe that even for 32×32 NoC, the analytical model takes only 45.07 ms to execute, while the simulation time of the 8×8 NoC is in the order of minutes. These results show that the proposed analysis technique is lightweight and provides four orders of magnitude speed-up compared to cycle-accurate simulations.

9 CONCLUSION OF THE THESIS AND FUTURE WORK

In this thesis, 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. 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.

Then we present a communication-aware in-memory computing architecture for GCNs. Besides accelerating the computation using custom compute elements (CE) and in-memory computing, the proposed GCN accelerator aims at minimizing the intra- and inter-CE communication in GCN operations to optimize the performance and energy efficiency. Experimental evaluations with widely used datasets show up to 105$\times$ improvement in energy consumption compared to state-of-the-art GCN accelerator.

We also 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.

BIBLIOGRAPHY

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

[2] Nvidia quadro rtx 8000. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/NVIDIA-Quadro-RTX-8000-PCIe-Server-Card-PB-FINAL-1219.pdf.

[3] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón. Computing Graph Neural Networks: A Survey from Algorithms to Accelerators. *arXiv preprint arXiv:2010.00130*, 2020.

[4] A. Abbas et al. A Survey on Energy-Efficient Methodologies and Architectures of Network-on-chip. *Computers & Electrical Engineering*, 40(8):333–347, 2014.

[5] V. Adhinarayanan, I. Paul, J. L. Greathouse, W. Huang, A. Pattnaik, and W.-c. Feng. Measuring and Modeling on-Chip Interconnect Power on Real Hardware. In *IEEE International Symposium on Workload Characterization*, pages 1–11.

[6] 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.

[7] A. B. Ahmed, A. B. Abdallah, and K. Kuroda. Architecture and Design of Efficient 3D Network-on-chip (3D NoC) for Custom Multicore SoC. In *Intl. Conf. on Broadband, Wireless Computing, Communication and Applications*, pages 67–73, 2010.

[8] S. Angizi, J. Sun, W. Zhang, and D. Fan. GraphS: A graph Processing Accelerator Leveraging SOT-MRAM. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 378–383, 2019.

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

[10] A. I. Arka, J. R. Doppa, P. P. Pande, B. K. Joardar, and K. Chakrabarty. ReGraphX: NoC-enabled 3D Heterogeneous ReRAM Architecture for Training Graph Neural Networks. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1667–1672. IEEE, 2021.

[11] A. I. Arka, B. K. Joardar, J. R. Doppa, P. P. Pande, and K. Chakrabarty. DARe: DropLayer-Aware Manycore ReRAM architecture for Training Graph Neural Networks. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

[12] A. I. Arka, B. K. Joardar, J. R. Doppa, P. P. Pande, and K. Chakrabarty. Performance and Accuracy Tradeoffs for Training Graph Neural Networks on ReRAM-Based Architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(10):1743–1756, 2021.

[13] 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.

[14] R. Ayoub, M. Kishinevsky, S. K. Mandal, and U. Y. Ogras. Analytical Modeling of NoCs for Fast Simulation and Design Exploration. In *Proceedings of the Workshop on System-Level Interconnect: Problems and Pathfinding Workshop*, pages 1–1, 2020.

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

[16] 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.

[17] S. Bell et al. Tile64-processor: A 64-core SoC with Mesh Interconnect. In *Intl. Solid-State Circuits Conf.-Digest of Technical Papers*, pages 88–598, 2008.

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

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

[20] S. Bharadwaj, J. Yin, B. Beckmann, and T. Krishna. Kite: A Family of Heterogeneous Interposer Topologies Enabled via Accurate Interconnect Modeling. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[21] G. Bhat, S. K. Mandal, U. Gupta, and U. Y. Ogras. Online Learning for Adaptive Optimization of Heterogeneous SoCs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 1–6, 2018.

[22] G. Bhat, S. K. Mandal, S. T. Manchukonda, S. V. Vadlamudi, A. Agarwal, J. Wang, and U. Y. Ogras. Per-Core Power Modeling for Heterogenous SoCs. *Electronics*, 10(19):2428, 2021.

[23] 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.

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

[25] 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.

[26] 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.

[27] A. Bojchevski and S. Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

[28] 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.

[29] 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.

[30] O. J. Boxma and B. W. Meister. Waiting-time Approximations in Multi-queue Systems with Cyclic Service. *Performance Evaluation*, 7(1):59–70, 1987.

[31] 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.

[32] 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.

[33] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka, and T. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

[34] N. Challapalle, M. Chandran, S. Rampalli, and V. Narayanan. X-VS: Crossbar-Based Processing-in-Memory Architecture for Video Summarization. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 592–597, 2020.

[35] N. Challapalle, S. Rampalli, M. Chandran, G. Kalsi, S. Subramoney, J. Sampson, and V. Narayanan. Psb-rnn: A Processing-in-Memory Systolic Array Architecture using Block Circulant Matrices for Recurrent Neural Networks. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 180–185. IEEE, 2020.

[36] N. Challapalle, S. Rampalli, N. Jao, A. Ramanathan, J. Sampson, and V. Narayanan. FARM: A Flexible Accelerator for Recurrent and Memory Augmented Neural Networks. *Journal of Signal Processing Systems*, 92(11):1247–1261, 2020.

[37] N. Challapalle, S. Rampalli, L. Song, N. Chandramoorthy, K. Swaminathan, J. Sampson, Y. Chen, and V. Narayanan. Gaas-x: Graph Analytics Accelerator Supporting Sparse Data Representation using Crossbar Architectures. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 433–445, 2020.

[38] N. Challapalle, K. Swaminathan, N. Chandramoorthy, and V. Narayanan. Crossbar based Processing in Memory Accelerator Architecture for Graph Convolutional Networks. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

[39] C.-H. O. Chen et al. SMART: a single-cycle reconfigurable NoC for SoC applications. In *Proc. of the Conf. on Design, Autom. and Test in Europe*, pages 338–343, 2013.

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

[41] 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.

[42] X. Chen et al. Rubik: A hierarchical architecture for efficient graph learning. *arXiv preprint arXiv:2009.12495*, 2020.

[43] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An Energy-efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE JSSC*, 52(1):127–138, 2016.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] W. J. Dally and B. Towles. Route Packets, not Wires: On-chip Interconnection Networks. In *Proc. of the Design Automation Conf.*, pages 684–689, 2001.

[49] W. J. Dally and B. P. Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[50] S. Das, J. R. Doppa, P. P. Pande, and K. Chakrabarty. Monolithic 3D-enabled High Performance and Energy Efficient Network-on-chip. In *Proc. of Intl. Conf. on Computer Design*, pages 233–240, 2017.

[51] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon. Demystifying 3D ICs: The Pros and Cons of Going Vertical. *IEEE Design & Test of Computers*, 22(6):498–510, 2005.

[52] B. K. Daya, L.-s. Peh, and A. P. Chandrakasan. Quest For High-Performance Bufferless Nocs With Single-Cycle Express Paths And Self-Learning Throttling. In *Proc. of Design Automation Conf.*, pages 1–6, 2016.

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

[54] 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.

[55] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt. SPARTA: Runtime Task Allocation for Energy Efficient Heterogeneous Many-cores. In *Proc. of the Intl. Conf. on Hardware/Software Codesign and Sys. Syn.*, page 27, 2016.

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

[57] J. Duato, S. Yalamanchili, and L. Ni. Interconnection Networks: An Engineering Approach, M. Kaufmann Pub. *Inc., USA*, 2002.

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

[59] 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.

[60] 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.

[61] 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.

[62] B. S. Feero and P. P. Pande. Networks-on-chip in a Three-Dimensional Environment: A Performance Evaluation. *IEEE Transactions on Computers*, 58(1):32–45, 2008.

[63] E. Fischer and G. P. Fettweis. An accurate and scalable analytic model for round-robin arbitration in network-on-chip. In *Networks on Chip (NoCS), IEEE/ACM Intl. Symp. on*, pages 1–8, 2013.

[64] T. Geng et al. AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 922–936.

[65] 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.

[66] 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.

[67] P. Ghosh, A. Sen, and A. Hall. Energy Efficient Application Mapping to NoC Processing Elements Operating at Multiple Voltage Levels. In *Intl. Symp. on Networks-on-Chip*, pages 80–85, 2009.

[68] 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.

[69] S. Gopal, S. Das, D. Heo, and P. P. Pande. Energy and Area Efficient Near Field Inductive Coupling: A Case Study on 3D NoC. In *Proc. of IEEE/ACM International Symposium on Networks-on-Chip*, pages 1–8, 2017.

[70] 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.

[71] W. P. Groenendijk and H. Levy. Performance Analysis of Transaction Driven Computer Systems via Queueing Analysis of Polling Models. *IEEE Computer Architecture Letters*, 41(04):455–466, 1992.

[72] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees. In *International Symposium on Computer Architecture*, pages 401–412, 2011.

[73] U. Gupta, S. K. Mandal, M. Mao, C. Chakrabarti, and U. Y. Ogras. A deep q-learning approach for dynamic management of heterogeneous processors. *IEEE Computer Architecture Letters*, 18(1):14–17, 2019.

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

[75] 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.

[76] 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.

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

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

[79] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro*, 27(5):51–61, 2007.

[80] J. Hu and R. Marculescu. DyAD: Smart Routing for Networks-on-chip. In *Proc. of Design Automation Conf.*, pages 260–263, 2004.

[81] J. Hu, U. Y. Ogras, and R. Marculescu. System-level Buffer Allocation for Application-Specific Networks-On-Chip Router Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2919–2933, 2006.

[82] 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.

[83] J. Huang, C. Buckl, A. Raabe, and A. Knoll. Energy-aware Task Allocation for Network-On-Chip based Heterogeneous Multiprocessor Systems. In *Intl. Euromicro Conf. on Parallel, Distributed and Network-Based Processing*, pages 447–454, 2011.

[84] 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.

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

[86] 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.

[87] A. Jantsch and H. Tenhunen. *Networks on Chip*, volume 396. Springer, 2003.

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

[89] N. E. Jerger, T. Krishna, and L.-S. Peh. On-chip Networks. *Synthesis Lectures on Computer Architecture*, 12(3):1–210, 2017.

[90] 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.

[91] 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.

[92] B. K. Joardar, A. I. Arka, J. R. Doppa, P. P. Pande, H. Li, and K. Chakrabarty. Heterogeneous Manycore Architectures Enabled by Processing-in-Memory for Deep Learning: From CNNs to GNNs:(ICCAD Special Session Paper). In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–7. IEEE, 2021.

[93] B. K. Joardar, A. Deshwal, J. R. Doppa, P. P. Pande, and K. Chakrabarty. High-throughput Training of Deep CNNs on ReRAM-based Heterogeneous Architectures via Optimized Normalization Layers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[94] B. K. Joardar, J. R. Doppa, P. P. Pande, H. Li, and K. Chakrabarty. AccuReD: High Accuracy Training of CNNs on ReRAM/GPU Heterogeneous 3-D Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(5):971–984, 2020.

[95] B. K. Joardar, R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu. Learning-based Application-agnostic 3D NoC Design for Heterogeneous Manycore Systems. *IEEE Transactions on Computers*, 68(6):852–866, 2018.

[96] B. K. Joardar, B. Li, J. R. Doppa, H. Li, P. P. Pande, and K. Chakrabarty. REGENT: A Heterogeneous ReRAM/GPU-based Architecture Enabled by NoC for Training CNNs. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 522–527. IEEE, 2019.

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

[98] N. Karmarkar. A New Polynomial-time Algorithm for Linear Programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[99] 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.

[100] 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.

[101] 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.

[102] 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.

[103] J. S. Kim, M. B. Taylor, J. Miller, and D. Wentzlaff. Energy Characterization of a Tiled Architecture Processor with On-chip Networks. In *Proc. of Intl. Symp. On Low Power Electronics And Design*, pages 424–427, 2003.

[104] K. Kiningham, C. Re, and P. Levis. Grip: A graph neural network acceleratorarchitecture. *arXiv preprint arXiv:2007.13828*, 2020.

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

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

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

[108] T. Krishna, A. Kumar, P. Chiang, M. Erez, and L.-S. Peh. NoC with Near-ideal Express Virtual Channels using Global-Line Communication. In *IEEE Symp. on High Performance Interconnects*, pages 11–20, 2008.

[109] A. Krishnakumar, S. E. Arda, A. A. Goksoy, S. K. Mandal, U. Y. Ogras, A. L. Sartor, and R. Marculescu. Runtime Task Scheduling Using Imitation Learning for Heterogeneous Many-Core Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4064–4077, 2020.

[110] 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.

[111] 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.

[112] G. Krishnan, S. K. Mandal, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao. Impact of On-chip Interconnect on In-memory Acceleration of Deep Neural Networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(2):1–22, 2021.

[113] G. Krishnan, S. K. Mandal, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao. System-Level Benchmarking of Chiplet-based IMC Architectures for Deep Neural Network Acceleration. In *2021 IEEE 14th International Conference on ASIC (ASICON)*, pages 1–4, 2021.

[114] G. Krishnan, S. K. Mandal, M. Pannala, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao. SIAM: Chiplet-based Scalable In-Memory Acceleration with Mesh for Deep Neural Networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–24, 2021.

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

[116] 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.

[117] A. Kumary, P. Kunduz, A. P. Singhx, L.-S. Pehy, and N. K. Jhay. A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS. In *Intl. Conf. on Computer Design*, pages 63–70, 2007.

[118] 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.

[119] 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.

[120] 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.

[121] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu. On-chip Communication Architecture Exploration: A Quantitative Evaluation of Point-To-Point, Bus, and Network-On-Chip Approaches. *ACM Transactions on Design Automation of Electronic Systems*, 12(3):1–20, 2008.

[122] 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.

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

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

[125] D. Li and J. Wu. Energy-efficient Contention-aware Application Mapping and Scheduling on NoC-based MPSoCs. *Journal of Parallel and Distributed Computing*, 96:1–11, 2016.

[126] G. Li, S. K. Mandal, U. Y. Ogras, and R. Marculescu. FLASH: Fast Neural Architecture Search with Hardware Optimization. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–26, 2021.

[127] Z. Li, N. Challapalle, A. K. Ramanathan, and V. Narayanan. IMC-Sort: In-Memory Parallel Sorting Architecture using Hybrid Memory Cube. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pages 45–50, 2020.

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

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

[130] S. Liu, A. Jantsch, and Z. Lu. Analysis and Evaluation of Design Trade-Offs Between Circuit Switched NoC and Packet Switched NoC. In *Euromicro Conf. on Digital System Design*, pages 21–28, 2013.

[131] 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.

[132] 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.

[133] 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.

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

[135] 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.

[136] 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.

[137] S. K. Mandal, G. Bhat, J. R. Doppa, P. P. Pande, and U. Y. Ogras. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM Transactions on Design Automation of Electronic Systems*, 25(3):1–26, 2020.

[138] S. K. Mandal et al. Dynamic Resource Management of Heterogeneous Mobile Platforms via Imitation Learning. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 2019.

[139] 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.

[140] 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.

[141] 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.

[142] S. K. Mandal, G. Krishnan, A. A. Goksoy, G. R. Nair, Y. Cao, and U. Y. Ogras. COIN: Communication-Aware In-Memory Acceleration for Graph Convolutional Networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2022.

[143] S. K. Mandal, U. Y. Ogras, J. R. Doppa, R. Z. Ayoub, M. Kishinevsky, and P. P. Pande. Online Adaptive Learning for Runtime Resource Management of Heterogeneous SoCs. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[144] S. K. Mandal, J. Tong, R. Ayoub, M. Kishinevsky, A. Abousamra, and U. Y. Ogras. Theoretical Analysis and Evaluation of NoCs with Weighted Round-Robin Arbitration. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

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

[146] 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.

[147] 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.

[148] R. Marculescu et al. Outstanding Research Problems In Noc Design: System, Microarchitecture, And Circuit Perspectives. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst.*, 28(1):3–21, 2008.

[149] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[150] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis. Evaluating Bufferless Flow Control for On-Chip Networks. In *Intl. Symp. on Networks-on-Chip*, pages 9–16, 2010.

[151] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, and M. Pedram. An Empirical Investigation of Mesh and Torus NoC Topologies under Different Routing Algorithms and Traffic Models. In *Euromicro Conf. on Digital System Design Architectures, Methods and Tools*, pages 19–26, 2007.

[152] 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.

[153] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 Network Architecture. *IEEE Micro*, 22(1):26–35, 2002.

[154] S. Murali, C. Seiculescu, L. Benini, and G. De Micheli. Synthesis of Networks on Chips For 3D Systems on Chips. In *Proc. of Asia and South Pacific Design Automation Conf.*, pages 242–247, 2009.

[155] S. M. Nabavinejad et al. An overview of Efficient Interconnection Networks for Deep Neural Network Accelerators. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(3):268–282, 2020.

[156] V. Narayanan, N. Challapalle, I. Okafor, S. Srinivasa, and N. Jao. Monolithic 3D Enabled Processing-in-SRAM Memory. In *2020 China Semiconductor Technology International Conference (CSTIC)*, pages 1–2. IEEE, 2020.

[157] 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.

[158] 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.

[159] 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.

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

[161] U. Y. Ogras and R. Marculescu. Energy-and Performance-driven NoC Communication Architecture Synthesis using a Decomposition Approach. In *Proc. of Design, Automation and Test in Europe*, pages 352–357, 2005.

[162] U. Y. Ogras and R. Marculescu. "It's a Small World After All": Noc Performance Optimization via Long-Range Link Insertion. *IEEE Trans. on Very Large Scale Integration Systems*, 14(7):693–706, 2006.

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

[164] 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.

[165] 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.

[166] S. Pasricha, R. Ayoub, M. Kishinevsky, S. K. Mandal, and U. Y. Ogras. A Survey on Eenergy Management for Mobile and IoT Devices. *IEEE Design & Test*, 2020.

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

[168] 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.

[169] 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.

[170] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. Efficient Synthesis of Networks on Chip. In *Proc of Intl. Conf. on Computer Design*, pages 146–150, 2003.

[171] 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.

[172] 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.

[173] 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.

[174] 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.

[175] 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.

[176] A.-M. Rahmani, K. Latif, P. Liljeberg, J. Plosila, and H. Tenhunen. Research and Practices on 3D Networks-on-Chip Architectures. In *NORCHIP*, pages 1–6, 2010.

[177] R. S. Ramanujam, V. Soteriou, B. Lin, and L.-S. Peh. Design of a High-throughput Distributed Shared-buffer NoC Router. In *ACM/IEEE Intl. Symp. on Networks-on-Chip*, pages 69–78, 2010.

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

[179] 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.

[180] S. K. Mandal et al. Communication-Aware In-Memory Acceleration for Graph Convolutional Networks: An Open Source Release. `https://github.com/sumitkmandal/GCN_accelerator_in-memory`, 2021. Accessed 10 April 2022.

[181] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar. A 2 Tb/s 6×4 Mesh Network for a Single-Chip Cloud Computer With DVFS in 45 nm CMOS. *IEEE Journal of Solid-State Circuits*, 46(4):757–766, 2011.

[182] S. Sarangi and B. Baas. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[183] 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.

[184] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, et al. Simba: Scaling Deep-learning Inference with Multi-chip-module-based Architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–27, 2019.

[185] 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.

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

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

[188] 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.

[189] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen. GraphR: Accelerating Graph Processing using ReRAM. In *2018 IEEE International Symposium on HPCA*, pages 531–543.

[190] A. Stillmaker and B. Baas. Scaling Equations for the Accurate Prediction of CMOS Device Performance from 180 nm to 7 nm. *Integration*, 58:74–81, 2017.

[191] Sumit K. Mandal et al. Theoretical Analysis and Evaluation of NoCs with Weighted Round-Robin Arbitration: An Open Source Release. `https://github.com/sumitkmandal/WRR_NoC_analytical_model`, 2021. Accessed 12 August 2021.

[192] H. Sun et al. Reliability-Aware Training and Performance Modeling for Processing-In-Memory Systems. In *26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 847–852. IEEE, 2021.

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

[194] M. B. Taylor et al. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro*, 22(2):25–35, 2002.

[195] 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.

[196] 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.

[197] N. Vallina-Rodriguez and J. Crowcroft. Energy Management Techniques in Modern Mobile Handsets. *IEEE Comm. Surveys & Tutorials*, (99):1–20, 2012.

[198] 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.

[199] J. Vanlerberghe. *Analysis and Optimization of Discrete-time Generalized Processor Sharing Queues*. PhD thesis, Ghent University, 2018.

[200] A. Varghese, B. Edwards, G. Mitra, and A. P. Rendell. Programming the adapteva epiphany 64-core network-on-chip coprocessor. *The Intl. Journal of High Performance Computing Applications*, 31(4):285–302, 2017.

[201] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, C. Fuguet, I. Miro-Panades, G. Moritz, J. Durupt, C. Bernard, D. Varreau, et al. IntAct: A 96-core Processor with Six Chiplets 3D-stacked on an Active Interposer with Distributed Interconnects and Integrated Power Management. *IEEE Journal of Solid-State Circuits*, 56(1):79–97, 2020.

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

[203] H. Wang, L.-S. Peh, and S. Malik. Power-driven Design of Router Microarchitectures in On-chip Networks. In *Proc. of Intl. Symp. on Microarchitecture*, pages 105–116, 2003.

[204] 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.

[205] 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.

[206] M. Wang et al. Deep graph library: Towards efficient and scalable deep learning on graphs. 2019.

[207] Z. Wang et al. GNN-PIM: A Processing-in-Memory Architecture for Graph Neural Networks. In *Conference on Advanced Computer Architecture*, pages 73–86. Springer, 2020.

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

[209] 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.

[210] 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.

[211] 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.

[212] 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.

[213] 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.

[214] S. Yan and B. Lin. Design of Application-Specific 3D Networks-on-chip Architectures. In *3D Integration for NoC-based SoC Architectures*, pages 167–191. 2011.

[215] 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.

[216] 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.

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

[218] 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.

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

[220] 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.

[221] W. Zhao and Y. Cao. New Generation of Predictive Technology Model for Sub-45 nm Early Design Exploration. *IEEE Transactions on Electron Devices*, 53(11):2816–2823, 2006.

[222] J. Zhou, S. K. Mandal, B. L. West, S. Wei, U. Y. Ogras, O. D. Kripfgans, J. B. Fowlkes, T. F. Wenisch, and C. Chakrabarti. Front–End Architecture Design for Low-Complexity 3-D Ultrasound Imaging Based on Synthetic Aperture Sequential Beamforming. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(2):333–346, 2020.

[223] P. Zhou, P.-H. Yuh, and S. S. Sapatnekar. Application-specific 3D Network-on-Chip Design using Simulated Allocation. In *Proc. of Asia and South Pacific Design Automation Conf.*, pages 517–522, 2010.

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

[225] 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.