A Deep Reinforcement Learning Framework for Architectural Exploration: A Routerless NoC Case Study

Ting-Ru Lin[1*], Drew Penney[2*], Massoud Pedram[1], Lizhong Chen[2]

[1]University of Southern California, Los Angeles, California, USA
[2]Oregon State University, Corvallis, Oregon, USA
[1]{tingruli, pedram}@usc.edu, [2]{penneyd, chenliz}@oregonstate.edu

ABSTRACT

Machine learning applied to architecture design presents a promising opportunity with broad applications. Recent deep reinforcement learning (DRL) techniques, in particular, enable efficient exploration in vast design spaces where conventional design strategies may be inadequate. This paper proposes a novel deep reinforcement framework, taking routerless networks-on-chip (NoC) as an evaluation case study. The new framework successfully resolves problems with prior design approaches, which are either unreliable due to random searches or inflexible due to severe design space restrictions. The framework learns (near-)optimal loop placement for routerless NoCs with various design constraints. A deep neural network is developed using parallel threads that efficiently explore the immense routerless NoC design space with a Monte Carlo search tree. Experimental results show that, compared with conventional mesh, the proposed deep reinforcement learning (DRL) routerless design achieves a 3.25x increase in throughput, 1.6x reduction in packet latency, and 5x reduction in power. Compared with the state-of-the-art routerless NoC, DRL achieves a 1.47x increase in throughput, 1.18x reduction in packet latency, 1.14x reduction in average hop count, and 6.3% lower power consumption.

Keywords

machine learning; network-on-chip; routerless

1. INTRODUCTION

Improvements in computational capabilities are increasingly reliant upon advancements in many-core chip designs. These designs emphasize parallel resource scaling and consequently introduce many considerations beyond those in single core processors. As a result, traditional design strategies may not scale efficiently with this increasing parallelism. Early machine learning approaches, such as simple regression and neural networks, have been proposed as an alternative design strategy. More recent machine learning developments leverage deep reinforcement learning to provide improved design space exploration. This capability is particularly promising in broad design spaces, such as network-on-chip (NoC) designs.

NoCs provide a basis for communication in many-core chips that is vital for system performance [9]. NoC design involves many trade-offs between latency, throughput, wiring resources, and other overhead. Exhaustive design space exploration, however, is often infeasible in NoCs and architecture in general due to immense design spaces. Thus, intelligent exploration approaches would greatly improve NoC designs.

Applications include recently proposed routerless NoCs [2, 29]. Conventional router-based NoCs incur significant power and area overhead due to complex router structures. Routerless NoCs eliminate these costly routers by effectively using wiring resources while achieving comparable scaling to router-based NoCs. Prior research has demonstrated up to 9.5x reduction in power and 7x reduction in area compared with mesh [2], establishing routerless NoCs as a promising alternative for NoC designs. Like many novel concepts and approaches in architecture, substantial ongoing research is needed to explore the full potential of the routerless NoC design paradigm and help advance the field. Design challenges for routerless NoCs include efficiently exploring the huge design space (easily exceeding 10^{12}) while ensuring connectivity and wiring resource constraints. This makes routerless NoCs an ideal case study for intelligent design exploration.

Prior routerless NoC design has followed two approaches. The first, isolated multi-ring (IMR) [29], uses an evolutionary approach (genetic algorithm) for loop design based on random mutation/exploration. The second approach (REC) [2] recursively adds loops strictly based on the NoC size, severely restricting broad applicability. Briefly, neither approach guarantees efficient generation of fully-connected routerless NoC designs under various constraints.

In this paper, we propose a novel deep reinforcement learning framework for design space exploration, and demonstrate a specific implementation using routerless NoC design as our case study. Efficient design space exploration is realized using a Monte-Carlo tree search (MCTS) that generates training data to a deep neural network which, in turn, guides the search in MCTS. Together, the framework self-learns loop placement strategies obeying design constraints. Evaluation shows that the proposed deep reinforcement learning design (DRL) achieves a 3.25x increase in throughput, 1.6x reduction in packet latency, and 5x reduction in power compared with a conventional mesh. Compared with REC, the state-of-the-art routerless NoC, DRL achieves a 1.47x increase in throughput, 1.18x reduction in packet latency, 1.14x reduction in average hop count, and 6.3% lower power consumption. When scaling from a 4x4 to a 10x10 NoC under synthetic workloads, the throughput drop is also reduced dramatically from 31.6% in REC to only 4.7% in DRL.

Key contributions of this paper include:

- Fundamental issues are identified in applying deep reinforcement learning to routerless NoC designs;
- An innovative deep reinforcement learning framework is proposed and implementation is presented for routerless NoC design with various design constraints;
- Cycle-accurate architecture-level simulations and circuit-level implementation are conducted to evaluate the design in detail;
- Broad applicability of the proposed framework with several possible examples is discussed.

*Equal contribution.

2378-203X/20/$31.00 ©2020 IEEE
DOI 10.1109/HPCA47549.2020.00018

The rest of the paper is organized as follows: Section 2 provides background on NoC architecture, reinforcement learning, and design space complexity; Section 3 describes issues in prior routerless NoC design approaches and the need for a better method; Section 4 details the proposed deep reinforcement learning framework; Section 5 illustrates our evaluation methodology; Section 6 provides simulation results; Section 7 reviews related work; Section 8 concludes.

2. BACKGROUND

2.1 NoC Architecture

Single-ring NoCs: Nodes in a single-ring NoC communicate using one ring connecting all nodes.[1] Packets are injected at a source node and forwarded along the ring to a destination node. An example single-ring NoC is seen in Figure 1(a). Single-ring designs are simple, but have low bandwidth, severely restricting their applicability in large-scale designs. Specifically, network saturation is rapidly reached as more nodes are added due to frequent end-to-end control packets [1]. Consequently, most single-ring designs only scale to a modest number of processors [22].

Router-based NoCs: NoC routers generally consist of input buffers, routing and arbitration logic, and a crossbar connecting input buffers to output links. These routers enable a decentralized communication system in which routers check resource availability before packets are sent between nodes [2]. Mesh (or mesh-based architectures) have become the *de facto* choice due to their scalability and relatively high bandwidth [29]. The basic design, shown in Figure 1(b), features a grid of nodes with a router at every node. These routers can incur 11% chip area overhead [13] and, depending upon frequency and activity, up to 28% chip power [7, 16] overhead, although some recent work [5,33] has shown much smaller overhead using narrow links and shallow/few buffers with high latency cost; this indirectly shows that routers are the main cost in existing NoCs. Hierarchical-ring, illustrated in Figure 1(c), instead uses several local rings connected by the dotted global ring. Routers are only needed for nodes intersected by the global ring as they are responsible for packet transfer between ring groups [3]. Extensive research has explored router-based NoC optimization [7, 17, 44], but these solutions only slightly reduce power and area overhead [29].

Routerless NoCs: Significant overhead associated with router-based topologies has motivated routerless NoC designs. Early proposals [44] used bus-based networks in a hierarchical approach by dividing the chip into multiple segments, each with a local broadcast bus. Segments are connected by a central bus with low-cost switching elements. These bus-based networks inevitably experience contention on local buses and at connections with the central bus, resulting in poor performance under heavy traffic. Recently, isolated multi-ring (IMR) NoCs have been proposed that exploit additional interconnect wiring resources in modern semiconductor processes [29]. Nodes are connected via at least one ring and packets are forwarded from source to destination without switching rings. IMR improves over mesh-based designs in terms of power, area, and latency, but requires significant buffer resources: each node has a dedicated input buffer for each ring passing through its interface, thus a single node may require many packet-sized buffers [2, 29]. Recent routerless NoC design (REC) [2] has mostly eliminated these costly buffers by adopting shared packet-size buffers among loops.

[1]Note that rings and loops are used interchangeably in this paper.

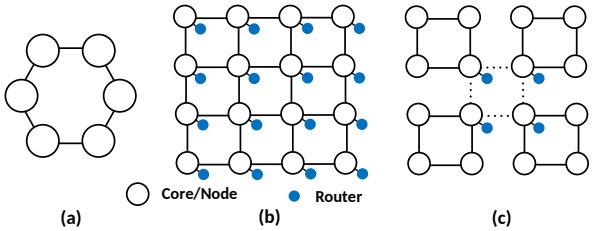

Figure 1: NoC Architecture. (a) Single-Ring (b) Mesh (c) Hierarchical Ring

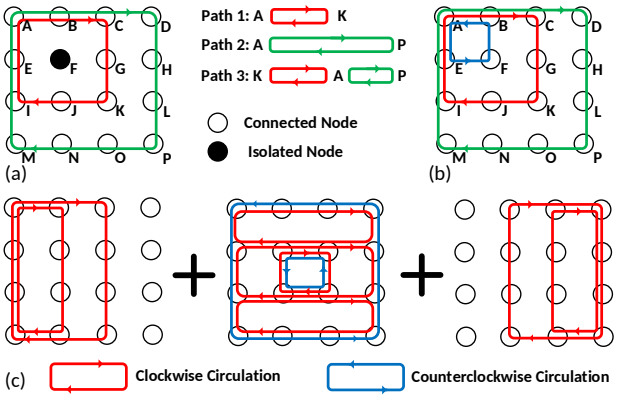

Figure 2: A 4x4 NoC with rings. (a) A NoC with one isolated node. (b) A NoC without isolated nodes. (c) A 4x4 routerless NoC with rings.

REC uses just a single flit-sized buffer for each loop, along with several shared extension buffers to provide effectively the same functionality as dedicated buffers [2].

Both IMR and REC designs differ from prior approaches in that no routing is performed during traversal, so packets in one loop cannot be forwarded to another loop [2, 29]. Both designs must therefore satisfy two requirements: every pair of nodes must be connected by at least one loop and all routing must be done at the source node. Figure 2 delineates these requirements and highlights differences between router-based and routerless NoC designs. Figure 2(a) depicts an incomplete 4x4 ring-based NoC with three loops. These loops are unidirectional so arrows indicate the direction of packet transfer for each ring. Node F is isolated and cannot communicate with other nodes since no ring passes through its interface. Figure 2(b) depicts the NoC with an additional loop through node F. If routers are used, such as at node A, this ring would complete the NoC, as all nodes can communicate with ring switching. Packets from node K, for example, can be transferred to node P using path 3, which combines $paths1$ and $path2$. In a routerless design, however, there are still many nodes that cannot communicate as packets must travel along a *single* ring from source to destination. That is, packets from node K cannot communicate with node P because $path1$ and $path2$ are isolated from each other. Figure 2(c) depicts an example 4x4 REC routerless NoC [2]. Loop placement for larger networks is increasingly challenging.

Routerless NoCs can be built with simple hardware interfaces by eliminating crossbars and VC allocation logic. As a result, current state-of-the-art routerless NoCs have achieved 9.5x power reduction, 7.2x area reduction, and 2.5x reduction in zero-load packet latency compared with conventional mesh topologies [2]. Packet latency, in particular, is greatly improved by single-cycle delays per hop, compared with stan-

dard mesh, which usually requires two cycles for the router alone. Hop count in routerless designs can asymptotically approach the optimal mesh hop count using additional loops at the cost of power and area. Wiring resources, however, are finite, meaning that one must restrict the total number of overlapping rings at each node (referred to as node overlapping) to maintain physical realizability. In Figure 2 (b), node overlapping at node A, for example, is three, whereas node overlapping at node F is one. Wiring resource restriction is one of the main reasons that make routerless NoC design substantially more challenging. As discussed in Section 3, existing methods either do not satisfy or do not enforce these potential constraints. We therefore explore potential applications and advantages of machine learning.

2.2 Reinforcement Learning

Reinforcement Learning Background: Reinforcement learning is a branch of machine learning that explores actions in an environment to maximize cumulative returns/rewards. Fundamental to this exploration is the environment, $\mathscr{E}$, in which a software agent takes actions. In our paper, this environment is represented by a routerless NoC design. The agent attempts to learn an optimal policy π for a sequence of actions $\{a_t\}$ from each state $\{s_t\}$, acquiring returns $\{r_t\}$ at different times t in $\mathscr{E}$ [42]. Figure 3 depicts the exploration process, in which the agent learns to take an action a_t (adding a loop) given a state s_t (information about an incomplete routerless NoC) with the goal of maximizing returns (minimizing average hop count). At each state, there is a transition probability, $P(s_{t+1}; s_t, a_t)$, which represents the probability of transitioning from s_t to s_{t+1} given a_t. The learned value function $V^{\pi}(s)$ under policy π is represented by

$$V^{\pi}(s) = \mathbb{E}[\sum_{t \geq 0} \gamma^t * r_t; s_0 = s, \pi] \quad (1)$$

$$R = \sum_{t \geq 0} \gamma^t * r_t \quad (2)$$

where γ is a discount factor (≤ 1) and R is the discounted cumulative return.

The goal of reinforcement learning is to maximize cumulative returns R and, in case of routerless NoC design, to minimize average hop count. To this end, the agent attempts to learn the optimal policy π^* that satisfies

$$\pi^* = arg\max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t * r_t; s_0 = s, \pi]. \quad (3)$$

Equation 1 under π^* thus satisfies the Bellman equation

$$V^*(s) = \mathbb{E}[r_0 + \gamma V^*(s_1); s_0 = s, \pi^*] \quad (4)$$

$$= p(s_0) \sum_{a_0} \pi^*(a_0; s_0) \sum_{s_1} P(s_1; s_0, a_0)[r(s_0, a_0) + \gamma V^*(s_1)] \quad (5)$$

where $p(s_0)$ is the probability of initial state s_0. The general form of $\pi(a_0; s_0)$ is interpreted as the probability of taking action a_0 given state s_0 with policy π. Equation 5 suggests that an agent, after learning the optimal policy function π^*, can minimize the average hop count of a NoC.

Deep Reinforcement Learning: Breakthroughs in deep learning have spurred researchers to rethink potential applications for deep neural networks (DNNs) in diverse domains. One result is deep reinforcement learning, which synthesizes DNNs and reinforcement learning concepts to address complex problems [35, 40, 41]. This synthesis mitigates data reliance without introducing convergence problems via efficient data-driven exploration based on DNN output. Recently, these concepts have been applied to Go, a grid-based strategy game involving stone placement. In this model, a trained policy DNN learns optimal actions by searching a Monte Carlo tree that records actions suggested by the DNN during training [40, 41]. Deep reinforcement learning can outperform typical reinforcement learning by generating a sequence of actions with better cumulative returns [35, 40, 41].

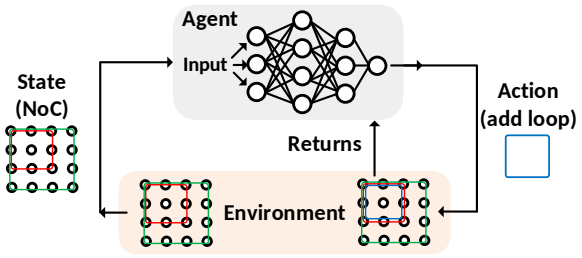

Figure 3: Reinforcement learning framework.

2.3 Design Space Complexity

Design space complexity in routerless NoCs poses a significant challenge requiring efficient exploration. A small 4x4 NoC using 10 loops chosen from all 36 possible rectangular loops has $\binom{36}{10} \approx 10^8$ total designs. This design space increases rapidly with NoC size; an 8x8 NoC with 50 loops chosen from 784 possible rectangular loops has $\binom{784}{50} \approx 10^{79}$ designs. It can be shown that the complexity of routerless NoC designs exceeds the game of Go. Similar to AlphaGo, deep reinforcement learning is needed here and can address this complexity by approximating actions and their benefits, allowing search to focus on high-performing configurations.

3. MOTIVATION

3.1 Design Space Exploration

Deep reinforcement learning provides a powerful foundation for design space exploration using continuously refined domain knowledge. This capability is advantageous since prior methods for routerless NoC designs have limited design space exploration capabilities. Specifically, the evolutionary approach [29] evaluates generations of individuals and offspring. Selection uses an objective function while evolution relies on random mutation, leading to an unreliable search since past experiences are ignored. Consequently, exploration can be misled and generate configurations with high average hop count and long loops (48 hops) in an 8x8 NoC [2]. The recursive layering approach (REC) overcomes these reliability problems but strictly limits design flexibility. Latency improves as the generated loops pass through fewer nodes on average [2], but hop count still suffers in comparison to router-based NoCs as it is restricted by the total number of loops. For an 8x8 NoC, the average hop count is 5.33 in mesh and 8.32 in the state-of-the-art recursive layering design, a 1.5x increase [2].

Both approaches are also limited by their inability to enforce design constraints, such as node overlapping. In IMR, ring selection is based solely on inter-core-distance and ring lengths [29] so node overlapping may vary significantly based on random ring mutation. Constraints could be built into the fitness function, but these constraints are likely to be violated to achieve better performance. Alternatively, in REC, loop configuration for each network size is strictly defined. A 4x4 NoC must use exactly the loop structure shown in Figure 2 (c)

so node overlapping cannot be changed without modifying the algorithm itself. These constraints must be considered during loop placement since an optimal design will approach these constraints to allow many paths for packet transfer.

3.2 Reinforcement Learning Challenges

Several challenges apply to deep reinforcement learning in any domain. To be more concrete, we discuss these considerations in the context of routerless NoC designs.

Specification of States and Action: State specification must include all information for the agent to determine optimal loop placement and should be compatible with DNN input/output structure. An agent that attempts to minimize average hop count, for example, needs information about the current hop count. Additionally, information quality can impact learning efficiency since inadequate information may require additional inference. Both state representation and action specification should be a constant size throughout the design process because the DNN structure is invariable.

Quantification of Returns: Return values heavily influence NoC performance so they need to encourage beneficial actions and discourage undesired actions. For example, returns favoring large loops will likely generate a NoC with large loops. Routerless NoCs, however, benefit from diverse loop sizes; large loops help ensure high connectivity while smaller loops may lower hop counts. It is difficult to achieve this balance since the NoC will remain incomplete (not fully connected) after most actions. Furthermore, an agent may violate design constraints if the return values do not appropriately deter these actions. Returns should be conservative to discourage useless or illegal loop additions.

Functions for Learning: Optimal loop configuration strategies are approximated by learned functions, but these functions are notoriously difficult to learn due to high data requirements. This phenomenon is observed in AlphaGo [40] where the policy function successfully chooses from 19^2 possible moves at each of several hundred steps, but requires more than 30 million data samples. An effective approach must consider this difficulty, which can be potentially addressed with optimized data efficiency and parallelization across threads, as discussed later in our approach.

Guided Design Space Search: An ideal routerless NoC would maximize performance while minimizing loop count based on constraints. Similar hop count improvement can be achieved using either several loops or a single loop. Intuitively, the single loop is preferred to reduce NoC resources, especially under strict overlapping constraints. This implies benefits from ignoring/trimming exploration branches that add loops with suboptimal performance improvement.

4. PROPOSED SCHEME

4.1 Overview

The proposed deep reinforcement learning framework is depicted in Figure 4. Framework execution begins by initializing the Monte Carlo Tree Search (MCTS) with an empty tree and a neural network without *a priori* training. The whole process consists of many exploration cycles. Each cycle begins with a blank design (e.g., a completely disconnected NoC). Actions are continuously taken to modify this design. The DNN (dashed "DNN" box) selects a good initial action, which directs the search to a particular region in the design space; several actions are taken by following MCTS (dashed "MCTS" box) in that region. The MCTS starts from the current design (a MCTS node), and tree traversal selects actions using either greedy exploration or an "optimal" action until a

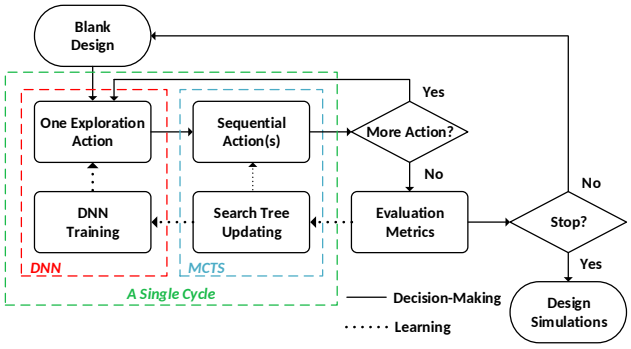

Figure 4: Deep reinforcement learning framework.

leaf (one of many explored designs) is reached. Additional actions can be taken, if necessary, to complete the design. Finally, an overall reward is calculated ("Evaluation Metrics") and combined with information on state, action, and value estimates to train the neural network and update the search tree (the dotted "Learning" lines). The exploration cycle repeats to optimize the design. Once the search completes, full system simulations are used to verify and evaluate the design. In the framework, the DNN generates coarse designs while MCTS efficiently refines these designs based on prior knowledge to continuously generate more optimal configurations. Unlike traditional supervised learning, the framework does not require a training dataset; instead, the DNN and MCTS gradually train themselves from past exploration cycles.

Framework execution in the specific case of routerless NoCs is as follows: each cycle begins with a completely disconnected routerless NoC; the DNN suggests an initial loop addition; following this initial action, one or more loops are added ("Sequential Action") by the MCTS; rewards are provided for each added loop; the DNN and MCTS continuously add loops until no more loops can be added without violating constraints; the completed routerless NoC configuration is evaluated by comparing average hop count to that of mesh to generate a cumulative reward; overall rewards, along with information on state, action, and value estimates, are used to train the neural network and update the search tree; finally, these optimized routerless NoC configurations are tested.

The actions, rewards, and state representations in the proposed framework can be generalized for design space exploration in router-based NoCs and in other NoC-related research. Several generalized framework examples are discussed in Section 6.8. The remainder of this section addresses the application of the framework to routerless NoC design as a way to present low-level design and implementation details. Other routerless NoC implementation details including deadlock, livelock, and starvation are addressed in previous work [2, 29] so are omitted here.

4.2 Routerless NoCs Representation

Representation of Routerless NoCs (States): State representation in our framework uses a hop count matrix to encode current NoC state as shown in Figure 5. A 2x2 routerless NoC with a single clockwise loop is considered for simplicity. The overall state representation is a 4x4 matrix composed of four 2x2 submatrices, each representing hop count from a specific node to every node in the network. For example, in the upper left submatrix, the zero in the upper left square corresponds to distance from the node to itself. Moving clockwise with loop direction, the next node is one hop away, then two, and three hops for nodes further along the loop. All other

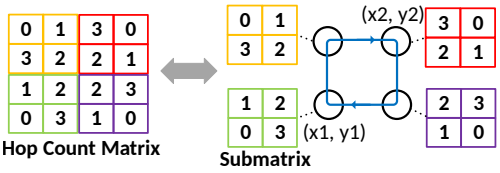

Figure 5: Hop count matrix of a 2x2 routerless NoC.

submatrices are generated using the same procedure. This hop count matrix encodes current loop placement information using a fixed size representation to accommodate fixed DNN layer sizes. In general, the input state for an $N \times N$ NoC is an $N^2 \times N^2$ hop count matrix. Connectivity is also implicitly represented in this hop count matrix by using a default value of $5 * N$ for unconnected nodes.

Representation of Loop Additions (Actions): Actions are defined as adding a loop to an $N \times N$ NoC. We restrict loops to rectangles to minimize the longest path. With this restriction, the longest path will be between diagonal nodes at the corners of the NoC, as in REC [2]. Actions are encoded as $(x1, y1, x2, y2, dir)$ where $x1, y1, x2$ and $y2$ represent coordinates for diagonal nodes $(x1, y1)$ and $(x2, y2)$ and dir indicates packet flow direction within a loop. Here, $dir = 1$ represents clockwise circulation for packets and $dir = 0$ represents counterclockwise circulation. For example, the loop in Figure 5 represents the action $(0, 0, 1, 1, 1)$. We enforce rectangular loops by checking that $x1 \neq x2$ and $y1 \neq y2$.

4.3 Returns After Loop Addition

The reward function encourages exploration by rewarding zero for all valid actions, while penalizing repetitive, invalid, or illegal actions using a negative reward. A repetitive action refers to adding a duplicate loop, receiving a -1 penalty. An invalid action refers to adding a non-rectangular loop, receiving a -1 penalty. Finally, illegal actions involve additions that violate the node overlapping constraint, resulting in a severe $-5 * N$ penalty. The agent receives a final return to characterize overall performance by subtracting average hop count in the generated NoC from average mesh hop count. Minimal average hop count is therefore found by minimizing the magnitude of cumulative returns.

4.4 Deep Neural Network

Residual Neural Networks: Sufficient network depth is essential and, in fact, leading results have used at least ten DNN layers [14, 40, 41]. High network depth, however, can cause overfitting for many standard DNN topologies. Residual networks offer a solution by introducing additional shortcut connections between layers that allow robust learning even with network depths of 100 or more layers. A building block for residual networks is shown in Figure 6(a). Here, the input is X and the output, after two weight layers, is $F(X)$. Notice that both $F(X)$ and X (via the shortcut connection) are used as input to the activation function. This shortcut connection provides a reference for learning optimal weights and mitigates the vanishing gradient problem during back propagation [14]. Figure 6(b) depicts a residual box (Res) consisting of two convolutional (conv) layers. Here, the numbers 3x3 and 16 indicate a 3x3x16 convolution kernel.

DNN architecture: The proposed DNN uses the two-headed architecture shown in Figure 6(c), which learns both the policy function and the value function. This structure has been proven to reduce the amount of data required to learn the optimal policy function [41]. We use convolutional layers because loop placement analysis is similar to spa-

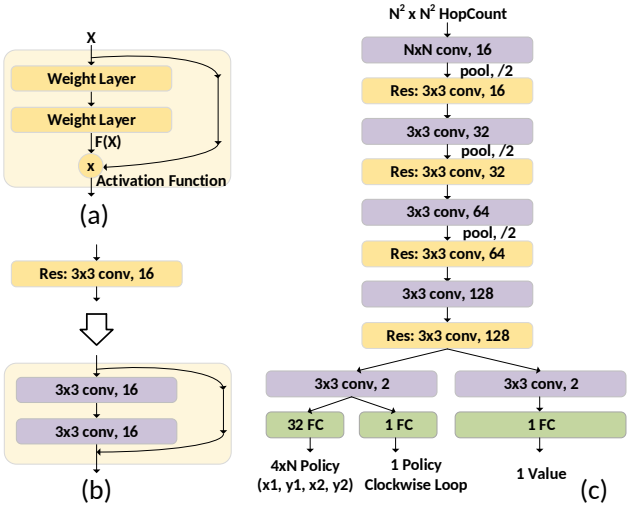

Figure 6: Deep residual networks. (a) A generic building block for residual networks. (b) A building block for convolutional residual networks. (c) Proposed network.

tial analysis in image segmentation, which performs well on convolutional neural networks. Batch normalization is used after convolutional layers to normalize the value distribution and max pooling (denoted "pool") is used after specific layers to select the most significant features. Finally, both policy and value estimates are produced at the output as the two separate heads. The policy, discussed in section 4.2, has two parts: the four dimensions, $x1, y1, x2, y2$, which are generated by a softmax function following a ReLU and dir, which is generated separately using a tanh function. Tanh output between -1 and 1 is converted to a direction using $dir > 0$ as clockwise and $dir \leq 0$ as counterclockwise. Referring to Figure 6(c), the softmax input after ReLU is $\{a_{ij}\}$ where $i = 1, 2, 3, 4$ and $j = 1, ..., N$. Dimensions $x1$ and $y1$ are $max_j(exp(a_{1j})/\sum_j exp(a_{1j}))$ and $max_j(exp(a_{2j})/\sum_j exp(a_{2j}))$. The same idea applies to $x2$ and $y2$. The value head uses a single convolutional layer followed by a fully connected layer, without an activation function, to predict cumulative returns.

Gradients for DNN Training: In this subsection we derive parameter gradients for the proposed DNN architecture.[2] We define τ as the search process for a routerless NoC in which an agent receives a sequence of returns $\{r_t\}$ after taking actions $\{a_t\}$ from each state $\{s_t\}$. This process τ can be described a sequence of states, actions, and returns:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, s_2, ...). \tag{6}$$

A given sequence of loops is added to the routerless NoC based on $\tau \sim p(\tau; \theta)$. We can then write the expected cumulative returns for one sequence as

$$\mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] = \int_\tau r(\tau) p(\tau; \theta) d\tau \tag{7}$$

$$p(\tau; \theta) = p(s_0) \prod_{t \geq 0} \pi(a_t; s_t, \theta) P(s_{t+1}; s_t, a_t), \tag{8}$$

where $r(\tau)$ is a return and θ is DNN weights/parameters we want to optimize. Following the definition of π in section

[2]Although not essential for understanding the work, this subsection provides theoretical support and increases reproducibility.

2.2, $\pi(a_0; s_0, \theta)$ is the probability of taking action a_0 given state s_0 and parameter θ. We then differentiate the expected cumulative returns for parameter gradients

$$\nabla \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] = \nabla_\theta \int_\tau r(\tau) p(\tau;\theta) d\tau \tag{9}$$

$$= \int_\tau (r(\tau) \nabla_\theta \log p(\tau;\theta)) p(\tau;\theta) d\tau \tag{10}$$

$$= \mathbb{E}_{\tau_\theta \sim p(\tau;\theta)}[r(\tau) \nabla_\theta \log p(\tau;\theta)]. \tag{11}$$

Notice that transition probability $P(s_{t+1,r_t}; s_t, a_t)$ is independent of θ so we can rewrite Equation 11 as

$$\mathbb{E}_{\tau_\theta \sim p(\tau;\theta)}[r(\tau) \nabla_\theta \log p(\tau;\theta)] \tag{12}$$

$$= \mathbb{E}_{\tau_\theta \sim p(\tau;\theta)}[r(\tau) \nabla_\theta \Sigma \log \pi(a_t; s_t, \theta)] \tag{13}$$

$$\approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi(a_t; s_t, \theta). \tag{14}$$

The gradient in equation 14 is proportional to raw returns (a constant value based on the *past* search trajectory). We therefore substitute $r(\tau)$ with A_t as

$$\nabla_\theta \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] \approx \sum_{t \geq 0} A_t \nabla_\theta \log \pi(a_t; s_t, \theta) \tag{15}$$

$$A_t = \sum_{t' > t} \gamma^{t'-t} r_{t'} - V(s_t; \theta), \tag{16}$$

where the first term in Equation 16 represents the returns from the *future* trajectory at time t. We also subtract $V(s_t; \theta)$ to reduce the variance when replacing a constant with a prediction. This approach is known as advantage actor-critic learning where the actor and the critic represent the policy function and value function, respectively [42]. In a two-headed DNN, θ consists of θ_π and θ_v for the policy function and the value function, respectively. Gradients for these two sets of parameters are directly obtained by representing Equation 15 as time intervals, rather than as a summation over time. These gradients are then given as

$$d\theta_\pi = (\sum_{t' > t} \gamma^{t'-t} r_{t'} - V(s_t; \theta_v)) \nabla_{\theta_\pi} \log \pi(a_t; s_t, \theta_\pi) \tag{17}$$

$$d\theta_v = \nabla_{\theta_v} (\sum_{t' > t} \gamma^{t'-t} r_{t'} - V(s_t; \theta_v))^2. \tag{18}$$

The whole training procedure repeats the following equations

$$\theta_\pi = \theta_\pi + \gamma * d\theta_\pi \tag{19}$$

$$\theta_v = \theta_v + c * \gamma * \theta_v, \tag{20}$$

where γ is a learning rate and c is a constant.

4.5 Routerless NoC Design Exploration

An efficient approach for design space exploration is essential for routerless NoC design due to the immense design space. Deep reinforcement learning approaches are therefore well-suited for this challenge as they can leverage recorded states while learning. Some work uses experience replay, which guides actions using random samples. These random samples are useful throughout the entire learning process, so improve collected state efficiency [35], but break the correlation between states. Another approach is the Monte Carlo tree search (MCTS), which is more closely correlated to human learning behavior based on experience. MCTS stores previously seen routerless NoC configurations as nodes in a tree structure. Each node is then labeled with the expected returns for exploration starting from that node. As a result,

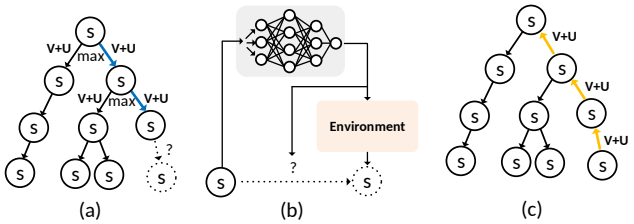

Figure 7: Monte Carlo tree search. (a) Search. (b) Expansion+evaluation using DNN. (c) Backup.

MCTS can provide additional insight during state exploration and help narrow the scope of exploration to a few promising branches [40] to efficiently learn optimal loop placement.

In our implementation, each node s in the tree represents a previously seen routerless NoC and each edge represents an additional loop. Additionally, each node s stores a set of statistics: $\overline{V}(s_{next})$, $P(a_i; s)$, and $N(a_i; s)$. $\overline{V}(s_{next})$ is the mean cumulative return from s_{next} and is used to approximate the value function $V^\pi(s_{next})$. $P(a_i; s)$ is the prior probability of taking action a_i based on $\pi(a = a_i; s)$. Lastly, $N(a_i; s)$ is the visit count, representing the number of times a_i was selected at s. Exploration starts from state s, then selects the best action a^* based on expected exploration returns given by

$$a^* = arg \max_{a_i} (U(s, a_i) + \overline{V}(s_{next})) \tag{21}$$

$$U(s, a_i) = c * P(a_i; s) \frac{\sqrt{\sum_j N(a_j; s)}}{1 + N(a_i; s)}, \tag{22}$$

where $U(s, a_i)$ is the upper confidence bound and c is a constant [39]. The first term in Equation 21 encourages broad exploration while the second emphasizes fine-grained exploitation. At the start, $N(a_i; s)$ and $\overline{V}(s_{next})$ are similar for most routerless NoCs so exploration is guided by $P(a_i; s) = \pi(a = a_i; s)$. Reliance upon DNN policy decreases with time due to an increasing $N(a_i; s)$, which causes the search to asymptotically prefer actions/branches with high mean returns [41]. Search is augmented by an ε-greedy factor where the best action is ignored with probability ε to further balance exploration and exploitation.

There are three phases to the MCTS algorithm shown in Figure 7: search, expansion+evaluation, and backup. (1) Search: an agent selects the optimal action (loop placement) by either following Equation 21 with probability $1 - \varepsilon$ or using a greedy search with probability ε. Algorithm 1 details the greedy search that evaluates the benefit from adding various loops and selects the loop with the highest benefit. *CheckCount*() returns the total number of nodes that can communicate after adding a loop with diagonal nodes at $(x1, y1)$ and $(x2, y2)$. Next, the *Imprv*() function returns the preferred loop direction based on the average hop count improvement. The tree is traversed until reaching a leaf node (NoC configuration) without any children (further developed NoCs). (2) Expansion+evaluation: the leaf state is evaluated using the DNN to determine an action for rollout/expansion. Here, $\pi(a = a_i; s)$ is copied, then later used to update $P(a_i; s)$ in Equation 22. A new edge is then created between s and s_{next} where s_{next} represents the routerless NoC after adding the loop to s. (3) Backup: After the final cumulative returns are calculated, statistics for the traversed edges are propagated backwards through the tree. Specifically, $\overline{V}(s_{next})$, $P(a_i; s)$, and $N(s, a_i)$ are all updated.

Algorithm 1 Greedy Search

```
Initialization: bestLoop = [0, 0, 0, 0], bestCount = 0, bestImprv = 0, and dir = 0
for x1 = 1;+1;N do
    for y1 = 1:+1;N do
        for x2 = x1+1:+1;N do
            for y2 = y1+1:+1;N do
                count = CheckCount(x1,y1,x2,y2)
                if count > bestCount then
                    bestCount = count
                    bestLoop = [x1, y1, x2, y2]
                    bestImpv, dir = Imprv(x1,y1,x2,y2)
                else if return == bestCount then
                    imprv', dre' = Imprv(x1,y1,x2,y2)
                    if imprv' > bestImprv then
                        bestLoop = [x1, y1, x2, y2]
                        bestImprv = imprv'
                        dir = dir'
return bestRing, dir
```

4.6 Multi-threaded Learning

The framework incorporates a multi-threaded approach, in which many threads independently explore the design space while collaboratively updating global parameters [34]. Figure 8 depicts the proposed framework with multi-threaded exploration. At the start, thread 0 creates a parent DNN with initial weights/parameters θ, then creates many child threads (1 to n) that create their own child DNNs, each of which acts as an individual learning agent. The parent thread sends DNN parameters to child threads and receives parameter gradients from child threads. Convergence is stabilized by averaging both large gradients and small gradients during training [34]. The parent thread additionally maintains a search tree that records past child thread actions for each MCTS query. While not needed for correctness, the multi-threaded approach facilitates more efficient exploration as shown in evaluation and is very useful in practice.

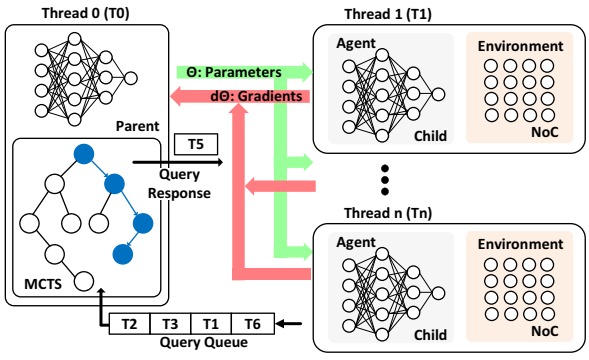

Figure 8: Multi-threaded framework.

5. METHODOLOGY

We evaluate the proposed deep reinforcement learning (DRL) routerless design against the previous state-of-the-art routerless design (REC) [2] and several mesh configurations. All simulations use Gem5 with Garnet2.0 for cycle-accurate simulation [6]. For synthetic workloads, we test uniform random, tornado, bit complement, bit rotation, shuffle, and transpose traffic patterns. Performance statistics are collected for 100,000 cycles across a range of injection rates, starting from 0.005 flits/node/cycle and incremented by 0.005 flits/node/cycle until the network saturates. Results for PARSEC are collected after benchmarks are run to completion with either sim-large or sim-medium input sizes.[3] Power and area estimations are based on Verilog post-synthesis simulation, following a similar VLSI design flow as in REC that synthesizes the Verilog implementation in Synopsys Design Compiler and conducts place & route in Cadence Encounter under 15nm NanGate FreePDK15 Open Cell Library [36].

We regard node overlapping as a more appropriate measure than link overlapping (i.e., the number of links between adjacent nodes) for manufacturing constraints. REC can only generate NoCs with a single node overlapping value for a given NoC size, whereas DRL designs are possible with many values. Comparisons between REC and DRL therefore consider both equal overlapping (demonstrating improved loop placement for DRL) and unequal overlapping (demonstrating improved design capabilities for DRL).

For synthetic and PARSEC workloads, REC and DRL variants use identical configurations for all other parameters, matching prior testing [2] for comparable results. Results nevertheless differ slightly due to differences between Gem5 and Synfull [4], used in REC testing. In REC and DRL, each input link is attached to a flit-sized buffer with 128-bit link width. Packet injection and forwarding can each finish in a single cycle up to 4.3 GHz. For mesh simulations, we use a standard two-cycle router delay in our baseline (Mesh-2). We additionally test an optimized one-cycle delay router (Mesh-1) and, in PARSEC workloads, an "ideal" router with zero router delay (Mesh-0) leaving only link/contention delays. These mesh configurations use 256-bit links, 2 VCs per link, and 4-flit input buffer. 128-bit links were considered, but exhibited a sub-optimal trade-off between power/area and performance (so would not provide a strong comparison against DRL). Packets are categorized into control and data packets, with 8 bytes and 72 bytes, respectively. The number of flits per packet is then given as packet size divided by link width. Therefore, in REC and DRL simulations, control packets are 1 flit and data packets are 5 flits. Similarly, in mesh simulations, control packets are 1 flit while data packets are 3 flits. For PARSEC workloads, L1D and L1I caches are set to 32 KB with 4-way associativity and the L2 cache is set to 128 KB with 8-way associativity. Link delay is set to one cycle per hop for all tests.

6. RESULTS & ANALYSIS

6.1 Design Space Exploration

Exploration starts without *a priori* experience or training data. Over time, as the search tree is constructed, the agent explores more useful loop configurations, which provide increased performance. Configurations satisfying design criteria can be found in seconds and minutes for 4x4 and 10x10 NoCs, respectively. Figure 9 illustrates a 4x4 DRL design. Different from REC [2], the generated topology replaces one inner loop with a larger loop and explores different loop directions. The resulting topology is completely symmetric and far more regular than IMR. We observe similar structure for larger topologies, but omit these due to space constraints.

Multi-threaded exploration efficacy is verified by comparing designs generated using either single or multi-threaded search. For a 10x10 NoC, after a 10 hour period, single-threaded search found 6 valid designs, whereas multi-threaded search found 49 valid designs. Moreover, multi-threaded search generates designs with 44% lower standard deviation

[3]Several workloads exhibit compatibility issues with our branch of Gem5, but we include all workloads that execute successfully.

Table 1: Hyperparameter Exploration

Epsilon (ε)	0.05	0.10	0.20	0.30
# Valid designs	25	27	11	2
Min Hop Count	5.59	5.60	5.61	5.53
SD for Hop Count	0.140	0.065	0.050	0.040

(SD) for hop count (decreasing from 0.027 to 0.015). This demonstrates the benefits of multi-threaded search to efficiently achieve more consistent results.

We further evaluate changes in the hyperparameter ε, which balances search exploration and exploitation. Results after a five hour period using 8x8 NoCs are summarized in Table 1. High values for ε can quickly generate more optimal configurations, but may frequently explore invalid actions and thus suffer under strict constraints. We therefore select the best value for ε in subsequent evaluations based on the time allocated to exploration as well as the rigor of constraints. In most cases, $\varepsilon = 0.1$ generates high-performing designs given adequate time.

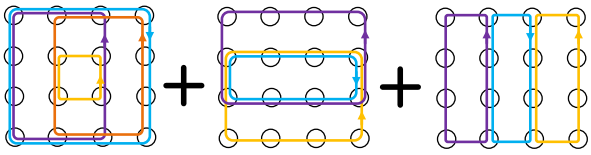

Figure 9: A 4x4 NoC topology generated by DRL.

6.2 Framework Capabilities

The proposed DRL framework can automatically generate NoC designs under various constraints so can be adapted to available design resources for any NoC size. In contrast, REC generates only a single design for each NoC size and, consequently, cannot be adapted to design goals, thus severely restricting real-world applicability. In the following, we exemplify the broad design capabilities of the DRL framework, none of which are possible with REC.

Generate feasible designs for larger NoCs: REC design does not work if node overlapping is less than $2*(N-1)$. Conversely, the proposed DRL framework can generate NoCs with smaller node overlapping across many sizes. For example, with a fixed node overlapping of 18, REC cannot generate NoCs larger than 10x10. Our DRL framework, however, has successfully generated configurations for 12x12, 14x14, 16x16 and 18x18 routerless NoCs. Note that 18x18 is the theoretical max routerless NoC size that can be fully connected with a node overlapping of 18. In a 20x20 NoC, there must be *at least* 19 rectangular loops passing through the bottom left node to connect to all other columns. As summarized in Table 2, the average hop count of DRL designs is still close to N, even when N approaches the node overlapping limit, showing the effectiveness of the DRL framework.

Utilize additional wiring resources: DRL is able to exploit additional wiring resources, when available, to improve performance, whereas REC cannot use any wires beyond $2*(N-1)$. Table 3 and Table 4 illustrate the hop count advantage of DRL over REC with various node overlappings. For example, a 10x10 DRL NoC with a node overlapping of 20 achieves a 20.4% reduction in hop count compared with the only possible REC 10x10 NoC.

Facilitate routerless NoC implementation in industry: Routerless NoCs offer a promising approach to achieve multi-fold savings in hardware cost compared with router-based NoCs, but the strict wiring requirements in the previous REC designs may hinder adoption in industry. The proposed DRL framework provides high flexibility to explore many combinations of NoC sizes and constraints that are not possible with REC. This flexibility can greatly aid future NoC research and implementation in industry by adapting to other constraints, such as maximum loop length or maximum hop count, which can also be integrated into the reward function.

Table 2: DRL supports larger NoCs with 18 overlapping.

NoC Size	10x10	12x12	14x14	16x16	18x18
REC Hop Count	9.64	N/A	N/A	N/A	N/A
DRL Hop Count	7.94	12.25	15.11	18.03	21.01

Table 3: DRL utilizes additional wiring resources; 8x8.

Topology	REC	DRL	DRL	DRL	DRL
Node overlapping	14	14	16	18	20
Hop count	7.33	6.22	5.94	5.82	5.80
Improve over REC	N/A	15.14%	18.96%	20.60%	20.87%

Table 4: DRL utilizes additional wiring resources; 10x10.

Topology	REC	DRL	DRL	DRL	DRL
Node overlapping	18	18	20	22	24
Hop count	9.64	7.94	7.67	7.59	7.55
Improve over REC	N/A	17.64%	20.44%	21.27%	21.68%

6.3 Synthetic Workloads

Performance evaluations in this and next subsections use a node overlapping constraint of $2*(N-1)$ for both REC and DRL because that is the *only* possible constraint for REC. Alternative DRL configurations, such as those shown in Tables 2 to 4, can nevertheless provide additional benefits while satisfying various design goals.

Packet Latency: Figure 10 plots the average packet latency of four synthetic workloads for a 10x10 NoC. Tornado and shuffle are not shown as their trends are similar to bit rotation. Zero-load packet latency for DRL is the lowest in all workloads. For example, with uniform random traffic, zero-load packet latency is 9.89, 11.67, 19.24, and 26.85 cycles for DRL, REC, Mesh-1, and Mesh-2, respectively, corresponding to a 15.2%, 48.6%, and 63.2% latency reduction by DRL. Across all workloads, DRL reduces zero-load packet latency by 1.07x, 1.48x and 1.62x compared with REC, Mesh-1, and Mesh-2, respectively. This improvement for both REC and DRL over mesh configurations results from reduced per hop latency (one cycle). DRL improves over REC due to additional connectivity and better loop placement. For example, in a 10x10 NoC, DRL provides four additional paths.

Throughput: DRL provides substantial throughput improvements for all traffic patterns. For uniform traffic, throughput is approximately 0.1, 0.125, 0.195, and 0.305 for Mesh-2, Mesh-1, REC, and DRL, respectively. Notably, in transpose, DRL improves throughput by 208.3% and 146.7% compared with Mesh-2 and Mesh-1. Even in bit complement where mesh configurations perform similarly to REC, DRL still provides a 42.8% improvement over Mesh-1. Overall, DRL improves throughput by 3.25x, 2.51x, and 1.47x compared with Mesh-2, Mesh-1, and REC, respectively. Again, additional loops with greater connectivity in DRL allow a greater throughput compared with REC. Furthermore, improved path diversity provided by these additional loops allows much higher throughput compared with mesh configurations.

6.4 PARSEC Workloads

We compare real-world performance of REC, DRL, and three mesh configurations for 4x4 and 8x8 NoCs on a set of PARSEC benchmarks. We generate Mesh-0 results by

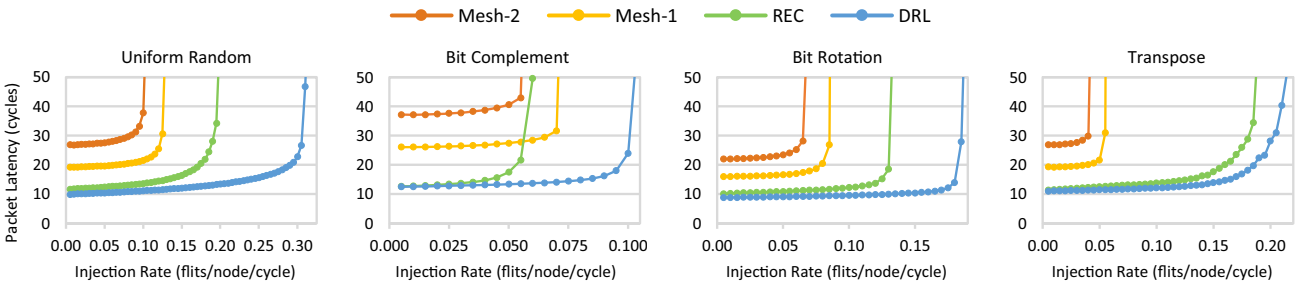

Figure 10: Average packet latency for synthetic workloads in 10x10 NoC.

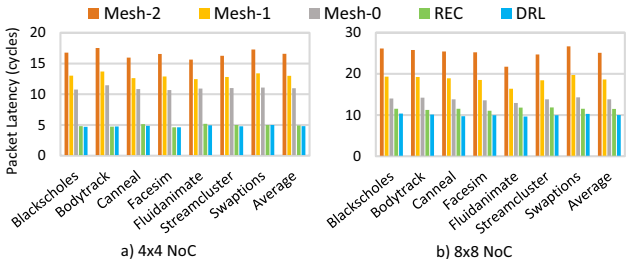

Figure 11: Packet latency for PARSEC workloads.

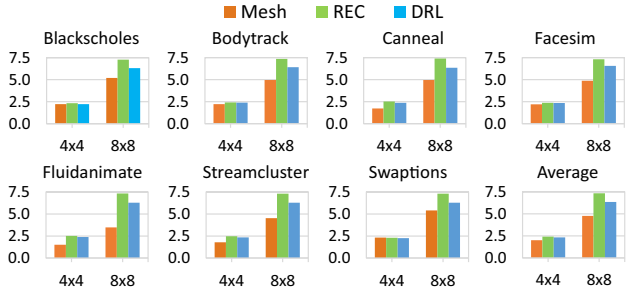

Figure 12: Average hop count for PARSEC workloads.

artificially reducing packet latency by the hop count for every recorded flit since such a configuration is difficult to simulate otherwise. As a result, performance is slightly worse than an "ideal" zero-cycle-router mesh.

Packet Latency: As shown in Figure 11, for the 4x4 network, variations in loop configuration are relatively small, being heavily influenced by full-connectivity requirements. Nevertheless, in the 4x4 NoC, DRL improves performance over REC in all but two applications where performance is similar. For example, DRL reduces packet latency by 4.7% in fluidanimate compared with REC. Improvements over mesh configurations for fluidanimate are greater with a 68.5%, 60.4%, and 54.9% improvement compared with Mesh-2, Mesh-1, and Mesh-0. On average, DRL reduces packet latency by 70.7%, 62.8%, 56.1%, and 2.6% compared with Mesh-2, Mesh-1, Mesh-0, and REC, respectively.

DRL improvements are more substantial in 8x8 NoCs as DRL can explore a larger design space. For example, in fluidanimate, average packet latency is 21.7, 16.4, 12.9, 11.8, and 9.7 in Mesh-2, Mesh-1, Mesh-0, REC, and DRL, respectively. This corresponds to a 55.6%, 41.0%, 25.3%, and 18.2% improvement for DRL compared with Mesh-2, Mesh-1, Mesh-0, and REC. On average, DRL reduces packet latency by 60.0%, 46.2%, 27.7%, and 13.5% compared with Mesh-2, Mesh-1, Mesh-0, and REC, respectively.

Table 5: 8x8 PARSEC workload execution time (ms)

Workload	NoC Type			
	Mesh-2	Mesh-1	REC	DRL
Blackscholes	4.4	4.2	4.0	4.0
Bodytrack	5.4	5.3	5.1	5.1
Canneal	7.1	6.4	6.1	6.0
Facesim	626.0	587.0	515.2	512.3
Fluidanimate	35.3	29.2	25.2	24.4
Streamcluster	11.0	11.0	11.0	11.0

Hop Count: Figure 12 compares the average hop count for REC, DRL, and Mesh-2 for 4x4 and 8x8 NoCs. Only Mesh-2 is considered as differences in hop count are negligible between mesh configurations (they mainly differ in per-hop delay). For 4x4 networks, REC and DRL loop configurations are relatively similar so improvements are limited, but DRL still provides some improvement in all workloads compared with REC. In streamcluster, average hop count is 1.79, 2.48, and 2.34 for mesh, REC, and DRL, respectively. On average, DRL hop count is 22.4% higher than mesh and 3.8% less than REC. For larger network sizes, we again observe the benefit from increased flexibility in loop configuration that DRL exploits. This optimization allows more loops to be generated, decreasing average hop count compared with REC by a minimum of 12.7% for bodytrack and a maximum of 14.3% in fluidanimate. On average, hop count for DRL is 13.7% less than REC and 35.7% higher than mesh.

Execution Time: Execution times for 8x8 PARSEC workloads are given in Table 5. Reductions in hop count and packet latency may not necessarily translate to reduced execution time as applications may be insensitive to NoC performance (notably streamcluster). Nevertheless, in fluidanimate, a NoC sensitive workload, DRL reduces execution time by 30.7%, 16.4%, and 3.17% compared with Mesh-2, Mesh-1, and REC, respectively. Overall, DRL provides the smallest execution time for every workload. Note that NoC traffic for PARSEC workloads is known to be light, so the significant throughput advantage of DRL over mesh and REC (Figure 10) is not fully reflected here. Additionally, as mentioned earlier, this evaluation restricts DRL to use the only overlapping value that works for REC. Larger benefits can be achieved with other DRL configurations, as shown next.

6.5 Power

The proposed DRL framework can generate diverse NoCs based on different objectives. Figure 13 demonstrates this capability as a tradeoff between power and performance (average hop count) for 8x8 NoCs. Each point represents one possible design and is labeled with the allowed node overlapping; REC therefore represents just a single design point. DRL with a node overlapping of 10 exhibits 1% lower hop count than REC while reducing power consumption by 15.9% due to reduced hardware complexity. Additionally, DRL with

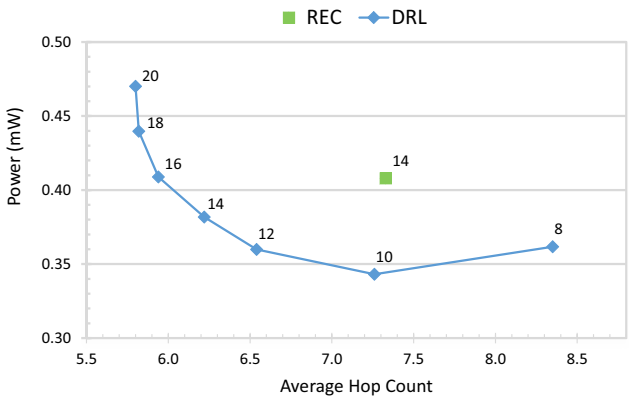

Figure 13: Power-performance tradeoffs for 8x8 (labels on the data points are node overlapping caps).

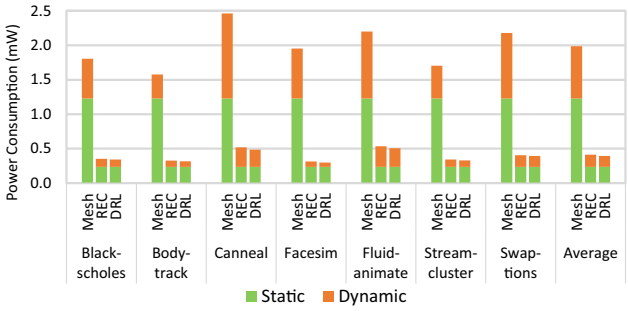

Figure 14: Power consumption for PARSEC workloads.

a node overlapping of 16 reduces the average hop count by 18.9% with nearly equal power consumption (within 0.2%) due to more efficient loop placement. Overall, DRL is more flexible and efficient than the fixed REC scheme.

We additionally compare the power consumption of mesh (Mesh-2), REC, and DRL (both with the same node overlapping of 14) across PARSEC workloads. Results are generated after place & route in Cadence Encounter under 15nm technology node [36]. Global activity factor is estimated from link utilization statistics in Gem5 simulations. A clock frequency of 2.0 GHz is used, comparable to commercial many-core processors. As seen in Figure 14, static power for REC and DRL is 0.23mW, considerably lower than the 1.23mW of mesh. Dynamic power is the lowest for DRL due to improved loop configuration, leading to lower hop count and therefore lower dynamic power than REC in all workloads. DRL also provides significant savings over mesh due to reduced routing logic and fewer buffers. On average, dynamic power for DRL is 80.8% and 11.7% less than mesh and REC, respectively.

6.6 Area

Node area in routerless NoCs is determined by the node overlapping cap. In practice, to reduce design and verification effort, the same node interface can be reused if the node overlapping cap is the same. Figure 15 therefore compares the node area for 8x8 mesh (Mesh-2), REC/DRL with an overlapping of 14 (equal area due to equal overlapping), and DRL with an overlapping of 10. DRL (10) is selected for comparison here because it has very similar hop count to REC, as shown in Figure 13. As can be seen, DRL (10) has the smallest area at 5,860 μm^2 due to an efficient design,

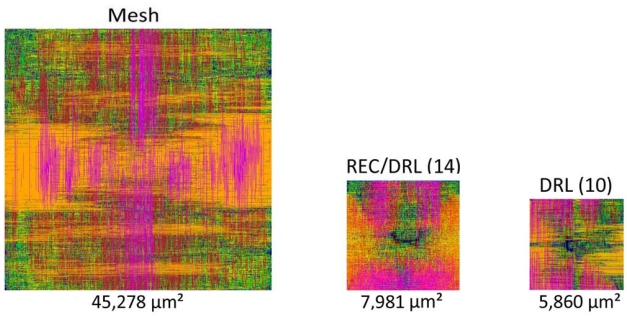

Figure 15: Area comparison (after P&R).

while providing equivalent performance to REC. Both REC and DRL with an overlapping of 14 have a slightly increased area at 7,981 μm^2. Finally, mesh area is much higher at 45,278 μm^2. This difference is mainly attributed to routerless NoCs eliminating both crossbars and virtual channels. Note that results for REC and DRL already include the small look-up table at source. This table is needed to identify which loop to use for each destination (if multiple loops are connected), but each entry has only a few bits [2]. Area for the table and related circuitry is 443 μm^2, equivalent to only 0.9% of the mesh router (power is 0.028mW or 1.13% of mesh). We also evaluated the additional repeaters necessary to support DRL. Total repeater area is 0.159 mm^2 for DRL (14), so overhead compared with REC represents just 1.1% of mesh.

6.7 Discussion

Comparison with IMR: Evaluation by Alazemi et al. [2] showed that REC is superior to IMR in all aspects. In synthetic testing, REC achieves an average 1.25x reduction in zero-load packet latency and a 1.61x improvement in throughput over IMR. Similarly, in real benchmarks, REC achieves a 41.2% reduction in average latency. Both static and dynamic power are also significantly lower in REC due to reduced buffer requirements and more efficient wire utilization. Finally, REC area is just 6,083 μm^2 while IMR area is 20,930 μm^2, corresponding to a 2.4x increase. Comparisons between REC and DRL were therefore the primary focus in previous subsections since REC better represents the current state-of-the-art in routerless NoCs. The large gap between IMR and REC also illustrates that traditional design space search (e.g., genetic algorithm in IMR) is far from sufficient, which calls for more intelligent search strategies.

Reliability: Reliability concerns for routerless NoC stem from the limited path diversity since wiring constraints restrict the total number of loops. For a given node overlapping, DRL designs provide more loops and thus more paths between nodes as more nodes approach the node overlapping cap. In the 8x8 NoC, there are, on average, 2.77 paths between any two nodes in REC. This increases to 3.79 paths, on average, between any two nodes in DRL (using equal overlapping). DRL can therefore tolerate more link failures before the NoC fails.

Scalability: DRL scales well compared with both REC and mesh configurations. For PARSEC workloads, shown in Figure 11, DRL exhibits 2.6% lower packet latency than REC for a 4x4 NoC, improving to a 13.5% reduction for an 8x8 NoC. Average hop count, shown in Figure 12, exhibits a similar trend. DRL improves average hop count by 3.8% in a 4x4 NoC and 13.7% in an 8x8 NoC. Scaling improvements are more evident in synthetic workloads. Figure 16,

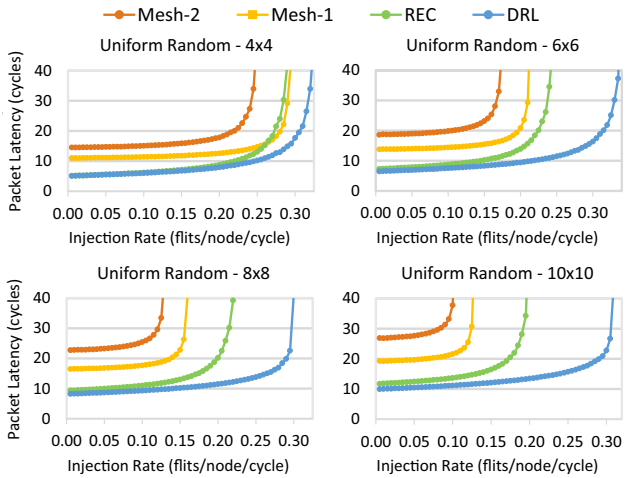

Figure 16: Synthetic Scaling for NoC Configurations.

for example, shows scaling results for 4x4 to 10x10 NoC sizes with uniform random workloads. Note that the same axis values are used for all NoC sizes to emphasize scaling performance. Whereas REC throughput decreases from 0.285 flits/node/cycle to 0.195 flits/node/cycle, corresponding to a 31.6% decrease, the throughput for DRL only changes slightly from 0.32 to 0.305 flits/node/cycle, corresponding to a 4.7% reduction. This shows a significant improvement in scalability from REC to DRL. Increasing the NoC size also allows more flexibility in loop exploration, and thus more effective use of wiring resources for a given node overlapping constraint. Additionally, loop design for $N \times M$ NoCs using DRL is straightforward to implement, only requiring modifications to the DNN for dimension sizes.

6.8 Broad Applicability

Routerless NoC design represents just one possible application for the framework presented in this paper. This framework, with modifications to state/action representations, could also be applied to router-based NoC designs. Specifically, one related application is in 3-D NoCs where higher dimensionality encourages novel design techniques. Prior work has explored small-world router-based designs [10, 11] using a relatively limited learning-based approach. The design space exploration would be more effective using our framework. Specifically, state representation using hop count remains compatible with the current DNN structure by concatenating matrices for each 2D layer. Actions can involve adding links between nodes in the same layer (intra-layer links) or different layers (inter-layer links). One DNN can be used for each action type to achieve an efficient deep reinforcement learning process with a smaller design space. A significant advantage of our framework is that strict constraints can be enforced on link addition, such as 3-D distance, to meet timing/manufacturing capabilities.

The proposed framework can also be generalized for other NoC-related research problems. While detailed exploration is beyond the scope of this paper, we briefly mention a few promising examples that can benefit from our framework. Future work may exploit underutilized wiring resources in silicon interposers [21, 26] and explore better ways to connect CPU cores and stacked memories. The framework could similarly be used to improve the latency and throughput of chiplet networks [31, 48] by exploring novel interconnects structures that are non-intuitive and hard for human to conceive. NoCs for domain-specific accelerators (e.g., TPU [25], Eyeriss [8], and others) are another possible application. Due to their data-intensive nature, accelerators can benefit from high-performance [27] and possibly reconfigurable [15] NoCs, where the framework can explore connectivity among processing elements (PEs) and between PEs and memory.

7. RELATED WORK

Research on routerless NoCs has been limited to two methods. IMR uses a genetic algorithm with random mutations to generate loop configuration. REC constructs layers recursively, generating an exact structure for a given NoC size. Our approach fundamentally differs from IMR and REC as it can guarantee fully connected loop configurations with various design constraints. This advantage is crucial to allow improved flexibility in diverse applications.

Many studies have explored machine learning applied to architecture and related tools [12, 18, 19, 20, 23, 24, 28, 30, 32, 37, 38, 43, 45, 46, 47, 49], but none have explored application to routerless NoCs. Jiménez et al. [43] used a perceptron-based approach for last level cache reuse prediction. Pattnaik et al. [37] demonstrated near-optimal scheduling for a processing-in-memory architecture using two regression models. Wang et al. [45] demonstrated a holistic design framework for NoCs using reinforcement learning. Margaritov et al. [32] proposed a highly-accurate scheme for virtual address translation in TLBs using a two-level neural network. Similar research is limited to specific architectural components, so is complementary to our work on routerless NoCs.

Machine learning has also been used to address NoC design concerns such as congestion. Ipek et al. [20] use reinforcement learning to mitigate traffic congestion with an approximate return function. The learned function allowed improved path selection for packet transfer using current traffic statistics such as queue lengths. That work, however, uses a single learned function and does not enforce specific design constraints. In contrast, our framework involves both a policy and value function, using a two-headed DNN structure, both of which are subject to strict design constraints.

8. CONCLUSION

Design space exploration using deep reinforcement learning promises broad application to architectural design. Current routerless NoC designs, in particular, have been limited by their ability to search design space, making routerless NoCs an ideal case study to demonstrate our innovative framework. The proposed framework integrates deep learning and Monte Carlo search tree with multi-threaded learning to efficiently explore large design space under constraints. Full system simulations shows that, compared with state-of-the-art routerless NoC, our proposed deep reinforcement learning NoC can achieve a 1.47x increase in throughput, 1.18X reduction in packet latency, 1.14x reduction in average hop count, and 6.3% lower power consumption. The proposed framework has broad applicability to diverse NoC design problems and enables intelligent design space exploration in future work.

ACKNOWLEDGMENT

We sincerely thank the reviewers for their helpful comments and suggestions. This research was supported, in part, by the National Science Foundation (NSF) grant #1750047, and the Software and Hardware Foundations program.

9. REFERENCES

[1] T. W. Ainsworth and T. M. Pinkston, "On characterizing performance of the cell broadband engine element interconnect bus," in *International Symposium on Networks-on-Chip*, May 2007.

[2] F. Alazemi, A. Azizimazreah, B. Bose, and L. Chen, "Routerless networks-on-chip," in *IEEE International Symposium on High Performance Computer Architecture*, Feb. 2018.

[3] R. Ausavarungnirun, C. Fallin, X. Yu, K. K.-W. Chang, G. Nazario, R. Das, G. H. Loh, and O. Mutlu, "Design and evaluation of hierarchical rings with deflection routing," in *International Symposium on Computer Architecture and High Performance Computing*, Oct. 2014.

[4] M. Badr and N. E. Jerger, "Synfull: synthetic traffic models capturing cache coherent behaviour," in *International Symposium on Computer Architecture*, 2014.

[5] J. Balkind, M. McKeown, Y. Fu, T. M. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "Openpiton: An open source manycore research framework," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Feb 2016.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basil, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *Computer Architecture News*, vol. 39, pp. 1–7, 2011.

[7] L. Chen and T. M. Pinkston, "Nord: Node-router decoupling for effective power-gating of on-chip routers," in *International Symposium on Microarchitecture*, Dec. 2012.

[8] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *International Symposium on Computer Architecture*, June 2016.

[9] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Design Automation Conference*, June 2001.

[10] S. Das, J. R. Doppa, D. H. Kim, P. P. Pande, and K. Chakrabarty, "Optimizing 3d noc design for energy efficiency: A machine learning approach," in *International Conference on Computer-Aided Design*, Nov. 2015.

[11] S. Das, J. R. Doppa, P. P. Pande, and K. Chakrabarty, "Energy-efficient and reliable 3d network-on-chip (noc): Architectures and optimization algorithms," in *International Conference on Computer-Aided Design*, Nov. 2016.

[12] Q. Fettes, M. Clark, R. Bunescu, A. Karanth, and A. Louri, "Dynamic voltage and frequency scaling in nocs with supervised and reinforcement learning techniques," *IEEE Transactions on Computers*, Oct. 2018.

[13] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger, "Implementation and evaluation of on-chip network architecture," in *International Conference on Computer Design*, Nov. 2007.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.

[15] B. Hong, Y. Ro, and J. Kim, "Multi-dimensional parallel training of winograd layer on memory-centric architecture," in *International Symposium on Microarchitecture*, Oct. 2018.

[16] Y. Hoskote, S. Vangal, A. Singh, H. Borkar, and S. Borkar, "A 5-ghz mesh interconnect for a teraflops processor," in *IEEE Micro*, Nov. 2007.

[17] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. V. D. Wijngaart, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling," in *IEEE Journal of Solid-State Circuits*, Jan. 2011.

[18] E. Ipek, S. A. McKee, B. de Supinski, M. Schulz, and R. Caruana, "Efficiently exploring architectural design spaces via predictive modeling," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2006.

[19] E. Ipek, S. A. McKee, K. Singh, R. Caruana, B. de Supinski, and M. Schulz, "Efficient architectural design space exploration via predictive modeling," in *ACM Transactions on Architecture and Code Optimization*, Jan. 2008.

[20] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana, "Self-optimizing memory controller: A reinforcement learning approach," in *International Symposium on Computer Architecture*, July 2008.

[21] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "Noc architectures for silicon interposer systems," in *International Symposium on Microarchitecture*, Dec. 2014.

[22] N. E. Jerger, T. Krishna, and L.-S. Peh, *On-Chip Networks*, 2nd ed. Morgan Claypool, 2017.

[23] D. A. Jiménez, "An optimized scaled neural branch predictor," in *International Conference on Computer Design*, Oct. 2011.

[24] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *International Symposium on High Performance Computer Architecture*, Jan. 2001.

[25] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, and J. Ross, "In-datacenter performance analysis of a tensor processing unit," in *International Symposium on Computer Architecture*, Dec. 2017.

[26] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *International Symposium on Microarchitecture*, Dec. 2015.

[27] H. Kwon, A. Samajdar, and T. Krishna, "Rethinking nocs for spatial neural network accelerators," in *International Symposium on Networks-on-Chip*, Oct. 2017.

[28] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *International Symposium on Computer Architecture*, June 2019.

[29] S. Liu, T. Chen, L. Li, X. Feng, Z. Xu, H. Chen, F. Chong, and Y. Chen, "Imr: High-performance low-cost multi-ring nocs," in *IEEE Transactions on Parallel Distributed Systems*, June 2016.

[30] Y. Liu, E. Z. Zhang, and X. Shen, "A cross-input adaptive framework for gpu program optimizations," in *International Symposium on Parallel & Distributed Processing*, July 2009.

[31] G. H. Loh, N. E. Jerger, A. Kannan, and Y. Eckert, "Interconnect-memory challenges for multi-chip, silicon interposer systems," in *International Symposium on Memory Systems*, Oct. 2015.

[32] A. Margaritov, D. Ustiugov, E. Bugnion, and B. Grot, "Virtual address translation via learned page tables indexes," in *Conference on Neural Information Processing Systems*, Dec. 2018.

[33] M. McKeown, A. Lavrov, M. Shahrad, P. J. Jackson, Y. Fu, J. Balkind, T. M. Nguyen, K. Lim, Y. Zhou, and D. Wentzlaff, "Power and energy characterization of an open source 25-core manycore processor," in *International Symposium on High Performance Computer Architecture*, Feb 2018.

[34] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, June 2016.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, Dec. 2013.

[36] Nangate Inc., "Nangate freepdk15 open cell library," [Online]. Available: http://www.nangate.com.

[37] A. Pattnaik, X. Tang, A. Jog, O. KayÄśran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling techniques for gpu architectures with processing-in-memory capabilities," in *International Conference on Parallel Architecture and Compilation Techniques*, Sept. 2016.

[38] B. Reagen, J. M. Hernández-Lobato, R. Adolf, M. Gelbart, P. Wahtmoug, G.-Y. Wei, and D. Brooks, "A case for efficient accelerator design space exploration via bayesian optimization," in *International Symposium on Low Power Electronics and Design*, July 2017.

[39] C. D. Rosin, "Multi-armed bandits with episode context," in *Annals of Mathematics and Artificial Intelligence*, Mar. 2011.

[40] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," in *Nature*, Jan. 2016.

[41] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," in *Nature*, Oct. 2017.

[42] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, USA: MIT Press, 1998.

[43] E. Teran, Z. Wang, and D. A. Jiménez, "Perceptron learning for reuse prediction," in *International Symposium on Microarchitecture*, Oct. 2016.

[44] A. N. Udipi, N. Muralimanohar, and R. Balasubramonian, "Towards scalable, energy-efficient, bus-based on-chip networks," in *International Symposium on High-Performance Computer Architecture*, Jan. 2010.

[45] K. Wang, A. Louri, A. Karanth, and R. Bunescu, "Intellinoc: A holistic design framework for energy-efficient and reliable on-chip communication for manycores," in *International Symposium on Computer Architecture*, June 2019.

[46] J.-Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management," in *International Symposium on High Performance Computer Architecture*, June 2014.

[47] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Neural acceleration for gpu throughput processors," in *International Symposium on Microarchitecture*, Dec. 2015.

[48] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. S. B. Altaf, N. E. Jerger, and G. H. Loh, "Modular routing design for chiplet-based systems," in *International Symposium on Computer Architecture*, June 2018.

[49] Y. Zeng and X. Guo, "Long short term memory based hardware prefetcher," in *International Symposium on Memory Systems*, Oct. 2017.