IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 20, 2022, accepted June 7, 2022, date of publication June 13, 2022, date of current version June 23, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3182500

Deep Reinforcement Learning Enabled Self-Configurable Networks-on-Chip for High-Performance and Energy-Efficient Computing Systems

MD FARHADUR REZA, (Member, IEEE)
Department of Mathematics and Computer Science, Eastern Illinois University, Charleston, IL 61920, USA
e-mail: mreza2@eiu.edu

This work is supported by my startup fund in the Department of Mathematics and Computer Science at the Eastern Illinois University.

ABSTRACT Network-on-Chips (NoC) has been the superior interconnect fabric for multi/many-core on-chip systems because of its scalability and parallelism. On-chip network resources can be dynamically configured to improve the energy efficiency and performance of NoC. However, large and complex design space in heterogeneous NoC architectures becomes difficult to explore within a reasonable time for optimal trade-offs of energy and performance. Furthermore, reactive resource management is not effective in preventing problems, such as thermal hotspots, from happening in adaptive systems. Therefore, we propose machine learning (ML) techniques to provide proactive solutions within an instant in NoC-based computing systems. We present a deep reinforcement learning (deep RL) technique to configure voltage/frequency levels of NoC routers and links for both high performance and energy efficiency while meeting the global energy budget constraint. Distributed RL agents technique has been proposed, where an RL agent configures a NoC router and associated links intelligently based on system utilization and application demands. Additionally, neural networks are used to approximate the actions of distributed RL agents. Simulations results for NoC sizes ranging from 16 to 256 cores under real applications and synthetic traffic show that the proposed self-configurable and scalable approach, on average, improves energy-delay product (EDP) by 30-40% (up to 80%) and by 8% (up to 17%) compared to existing non-ML and ML based solutions, respectively.

INDEX TERMS Network-on-chip (NoC), multicore architecture, mancore processor, machine learning (ML), reinforcement learning (RL), distributed RL, deep reinforcement learning (Deep RL), Q-learning, neural networks (NNs), self-configurable, energy-efficiency, high-performance.

I. INTRODUCTION

Multiprocessor System-on-Chips (MPSoCs) and chip multiprocessors (CMP) are moving towards the integration of hundreds to thousands of cores on a chip. Manycore on-chip systems have better power efficiency, interconnection, and latency compared to traditional local area network (LAN) based systems [28]. Industry and academia researchers are working on manycore single chip solution that could replace the traditional big data-center solution and/or that can be used as the base chip for supercomputer with million cores [1], [8], [10]. For example, a single chip with 850K independent cores has been developed by Cerebras [1], and each processor chip of Sunway TaihuLight with 40960 processors contains 256 cores [10]. As on-chip systems can contain hundreds to thousand of cores, network-on-chip (NoC) has been adopted as a standard solution to manage the complex on-chip communication among cores, where cores are running the tasks of applications. NoC offers several important benefits over traditional bus in terms of scalability, parallelism, throughput, and power efficiency for on-chip communication in multi/many-core systems [5], [14], [16]. The number of NoC router and link resources to support the increased number of cores in on-chip system has also increased significantly. This results in significant increase in NoC power consumption relative to the total chip power. Several NoC prototypes have shown

The associate editor coordinating the review of this manuscript and approving it for publication was Seifedine Kadry.

that on-chip network consumes 10-40% of the total chip power, including 30% in the Intel 80-core Terascale chip [24] and 40% in the MIT RAW chip [48]. Furthermore, [29] demonstrated that data movement consumes 25% of the total energy. Because of the gap between transistor density and transistor power efficiency in process technology (failure of Dennard Scaling [17]), manycore chip faces power budget problem [47]. High power consumption also affects the lifespan of systems, due to increased heat buildup.

With the advancement of technology, computation becomes more energy-efficient compared to communication. In 7*nm* transistor technology, energy per unit communication consumes 6-time more energy than that of computation [9]. The fraction of time spent in communication for an application increases with the increase in the number of processing cores in systems and it (fraction of time spent in communication) can be more than 50% for large-scale systems [6], which significantly hampers the parallelism performance (as computations need to wait for data transferred in communication). The key element to scalable chip performance is the on-chip interconnects connecting the cores/memory. Therefore, a challenging research problem is to design energy-efficient and high-performance NoC for multi/many-core computing systems.

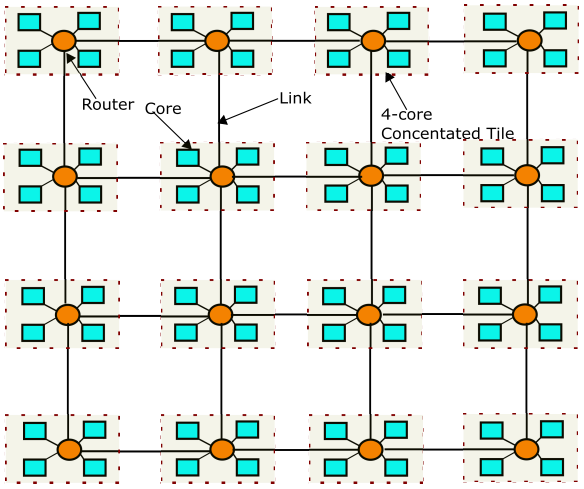

FIGURE 1. 64-core architecture connected through 4 × 4 concentrated 2D-Mesh NoC.

Our objective is to obtain high-performance NoC while achieving energy-efficiency. In this work, besides small-scale 16-core NoC, we consider large-scale 256-core architecture that is placed in a 8×8 concentrated mesh (CMesh) topology. A 64-core CMesh architecture is illustrated in Figure 1, where cores are placed in a 4 × 4 2D-Mesh topology. 4-core are concentrated with a single router in a tile, where every core can be heterogeneous with different computational capacities. Each core has an individual L1 cache and each router has an L2 cache shared among the four cores connected to each router. Routers are connected to each other via links to form the NoC, where every router can be heterogeneous with different communication capacities, including varying bandwidth links and buffer counts.

Different tasks of an application can have different computation and communication demands, for example, one task can be computation-intensive where another task can be communication-intensive. Furthermore, multiple applications with different demands can run simultaneously on a multi/many-core chip. Multi/many-core systems can contain heterogeneous computing nodes, for example, GPUs, accelerators, and CPUs. NoC can be designed with heterogeneous capacities in different parts of NoC. For example, routers with more buffers and links with higher bandwidth can be designed to support computing nodes with higher communication requirements. Because of the diverse traffic patterns of application(s), different routers and links in NoC can carry different amounts of data by merging traffic flows from various tasks of the application(s). To cope with the heterogeneity of traffic workloads and heterogeneity of computing resources, NoC can be configured heterogeneously for energy efficiency and high performance. Heterogeneous NoC configurations include V/F-scaling of routers and links. Prior research has shown that dynamic voltage and frequency scaling (DVFS) can reduce dynamic energy consumption of NoC routers and/or links at run-time [33], [35]. Many works have effectively applied voltage/frequency (V/F) scaling on on-chip systems and networks for energy reduction [4], [11], [22], [51], [53]. With DVFS, supply voltage is increased during high NoC traffic to meet the NoC performance requirements while supply voltage needs to be decreased during low traffic for energy savings. Because of the large and complex NoC design space (V/F-levels, heterogeneity of cores and routers, multicores, topology, task-core mapping, etc.), reactive solutions using heuristics or linear optimization solvers are not effective at run-time systems due to the high time complexity for exploring many designs and configuration parameters for optimal/near-optimal solutions. Large problem instances cannot be computed using linear programming optimizers (e.g., integer linear programming solver) for optimal solutions since they fail to compute results in time for run-time configuration decisions. Heuristics may not cover all possible cases under heterogeneous tasks (and traffic) and heterogeneous resources. Therefore, reactive or ad-hoc resource configuration may not be an effective technique in preventing problems, such as creating thermal hotspots and exceeding chip power budget, from happening in adaptive systems. For example, reactive/slow solutions may not increase or decrease V/F-levels properly to meet the demands of the applications while providing both high-performance and energy-efficient solution. Machine learning (ML) can help by predicting NoC resource requirements (before it requires at that instant) for different applications in advance and configure the NoC accordingly to minimize power and thermal variations while avoiding any loss in performance.

In this paper, we propose the use of deep reinforcement learning (deep RL) to dynamically configure NoC resources based on precise system utilization and application demands.

RL agent is used to monitor the states (in terms of features) and state transitions and to take actions and evaluate the rewards of the NoC configuration actions. RL agent reinforces the positive or negative reward of the taken decision, which helps the system to learn, take proper actions and achieve the optimization objectives of energy efficiency and high performance. We take advantage of neural networks (NNs) to learn the patterns in application demands and to approximate NoC configuration decision accordingly. The major contributions of this work are outlined below:

- Self-configurable NoC using Reinforcement Learning: RL agents are trained to automatically configure the NoC resources (routers and links) to maximize NoC performance while ensuring energy efficiency. V/F-levels of the NoC routers and links are dynamically configured at run-time based on the application demand using RL while global energy budget constraint is used to limit energy consumption of NoC resources. RL agents select exploration (random-action) or exploitation (best-action) policy with random probability ϵ to get the global optimal solution.
- Distributed Reinforcement Learning Agents and Neural Network Approximators: Distributed RL techniques for NoC configurations are proposed and implemented to make the proposed ML-enabled technique feasible for large-scale NoC-based systems with reasonable hardware overhead (for ML). NNs are used to approximate the actions for RL agents based on the learning of the state (features), where traditional table-based learning approach is not feasible because of the need for large state-action mapping tables and high convergence time for learning.
- Evaluation on Real and Synthetic Benchmarks: The proposed approach is evaluated using both large-scale (64-core and 256-core) and small-scale (16-core) NoC architectures on a real system simulator. Both real benchmarks (with large and small applications) and synthetic traffic are used for evaluating the proposed approach compared to existing non-machine-learning (non-ML) based solution. Simulation results under real (COSMIC and E3S) and synthetic benchmarks show that the proposed approach, on average, improves latency by 25% and improves energy and throughput by 6% (improves EDP by 30-40%) compared to a non-ML based solution [40] and improves EDP by 8% compared to an RL-based solution [38].

The paper is organized as follows. We discuss the related work on NoC configuration in Section II. Self-configurable NoC configuration strategy using RL and NN approximator is presented in Section III. Simulation results are presented in Section IV.

II. RELATED WORK

ML techniques, such as RL, regressions, and NNs, have been proposed for design and optimization challenges, including energy and performance, of multi/many-core systems and on-chip networks. Reference [31] proposes an auotomated data-driven framework to quickly configure and design manycore systems for a wide-range of application and operating scenarios. The authors proposed to use ML for both design-time and run-time decisions to create fully-adaptive systems.

A few existing works using RL for NoC optimization and configuration are discussed here. Reference [49] used distributed RL for simultaneously optimizing performance, energy efficiency and reliability of NoC in manycore systems, where each router independently takes the decisions. Reference [38] proposed RL to configure NoC link-bandwidths dynamically by scaling V/F-levels for energy savings. Multiple wires on a link (between routers) are activated or deactivated to configure the required link-width based on the dynamic communication requirements between the tasks of an application. Reference [52] presented a deep RL approach for efficient NoC arbitration. The proposed self-learning decision making mechanism reduces packet latency, which results in improved NoC throughput. In [25], the authors proposed cooperative multi-agent RL-based co-optimization techniques to jointly perform different performance and power optimization involving cache partitioning and DVFS of NoC, core and cache.

In addition to RL, other ML techniques (e.g., decision tree) have also been proposed for NoC optimization and configuration. Reference [30] proposed an imitation learning (IL) based methodology for dynamic V/F-island (cluster of nodes/links) control in manycore systems. [19] leveraged decision trees to predict and mitigate errors before the fault affects NoC based systems. The proposed decision tree model achieves reduction in packet re-transmission and energy savings. Reference [27] presented an NN-based intelligent hotspot prediction mechanism that was used with a congestion-control mechanism to handle hotspot formations efficiently. Reference [39] proposed run-time predictive configuration of node voltage-levels and link widths of NoC using NNs for energy-savings while addressing both power and temperature constraints of manycore NoC. Reference [32] proposed NN based predictive routing algorithm for NoC which uses network state and congestion information to estimate routing costs and perform low-latency routing of traffic. Reference [13] proposed ML-enabled energy-aware dynamic V/F scaling for NoC architectures. The proposed work relies on an offline trained regression model and provides a wide variety of V/F pairs. Reference [20] extended this work by adopting RL for selecting DVFS mode directly, which removes the need for labelling in linear regression. Some works focus only on core resources of the multi/many-core systems instead of uncore (including NoC) resources. Reference [46] proposed RL-enabled online power management technique that learns the best power management policy that gives the minimum power consumption for a given performance constraint without any prior information of workload. Reference [21] proposed an RL based I/O management for energy-efficient communication between manycore

processor and memory, instead of transmitting data under a fixed large voltage-swing. Reference [12] presented an on-line distributed RL based DVFS control algorithm for manycore system under power constraints. Per-core RL method is used to learn the optimal control policy of the V/F-levels in the system. At the coarser grain, an efficient global power budget reallocation algorithm is used to maximize the overall performance. In [25], the authors proposed cooperative multi-agent RL-based co-optimization techniques to jointly perform different performance and power optimization involving cache partitioning and DVFS of core and uncore. Reference [15] proposed ML to intelligently explore the design space of 3D NoC to optimize the placement of both planar and vertical communication links for energy efficiency. Reference [26] developed a learning-based framework using lasso regression to enable fast and accurate transient thermal prediction in chip multiprocessor.

However, unlike previous works that focus mostly on small-scale NoCs, this work (which is the extended version of [37]) focuses to provide solutions for both small-scale and large-scale NoCs using deep RL techniques. Furthermore, this work focuses to improve both energy and performance using online RL technique for predicting V/F-levels of the NoC routers and links depending on the computation and communication requirements of various applications running in multi/many-core systems. Besides performance improvement and energy efficiency, the main advantage of RL is its self-adaptive nature to configure NoC at an instant to meet the real time requirements of application(s), where supervised ML technique need complete retraining to adapt to changes in systems/applications and traditional non-ML based algorithms and optimization solvers fail to provide solutions within a reasonable time (because of high time complexity).

III. DYNAMIC NoC CONFIGURATION USING DEEP REINFORCEMENT LEARNING

In this work, model-free RL, namely Q-learning, is adapted. Q-learning directly estimates the optimal Q-values of each action in each state, from which a policy is derived (instead of learning a model of the environment). When applications are running in the NoC-based multi/many-core systems, RL is used to configure the NoC V/F-levels dynamically based on the communication demands of the tasks to improve NoC performance. We have selected four different voltage-levels for NoC configurations: 0.8V, 0.9V, 1.0V, and 1.1V. We limited our voltage-levels as too many levels will have high overhead for voltage regulator(s). However, we chose voltage-levels that meet the energy (power) budget constraint. With each voltage (V) level, a frequency (F) is also selected to form a V/F pair. The following V/F pairs are used in this work: 0.8 V/1 GHz, 0.9 V/1.5 GHz, 1.0 V/2 GHz, and 1.1V/2.5 GHz. RL agents are trained to maximize performance while global energy budget constraint is used to limit the V/F-levels of NoC resources.

A. REINFORCEMENT LEARNING AND Q-LEARNING

In RL, training data is given as a feedback to the program's actions in a dynamic environment. Feedback is given in forms of rewards and punishments to reinforce the actions, such as scaling V/F-levels and link bandwidth in NoC. RL is very much suitable for intelligent decisions in autonomic and dynamic systems because of the following reasons [45]: firstly, an autonomic system learns (using RL) what actions to take to maximize the long-term rewards from a specific state. Secondly, RL properly treats the dynamic phenomena as it can take into account delayed consequences (decision and states) for current action in the environment. Because of the autonomic property of RL, RL is very effective to handle run-time changes (link/router fault, change in application demands, etc.) by interacting with the system and observing the costs and then optimize the system. Another advantage of RL is that it does not need the labelling of the training data (output label), which is required in supervised learning (e.g., NNs, regressions). Therefore, RL is adapted in this work to dynamically configure V/F-levels depending on the application demands to maximize the performance and energy efficiency of NoC. Our proposed approach apply online learning because it allows the algorithm to learn as data becomes available instead of learning from a static data set. Performance and utilization data are collected in each interval after taking action, and that data is used for training RL agent.

Q-learning is used as an RL technique for finding optimal policy for selecting NoC V/F-level configuration actions. The core of the Q-learning is a simple value iteration update, using an iterative algorithm, and Q-value in the learning algorithm is calculated by using the weighted average of the old value and the new information. The Q-value of taking action a in state s at current time step t is denoted $Q(s_t, a_t)$. Q-learning does not require building explicit representations of the state transitions and the expected reward to estimate Q-value. This helps the system as initially the system is not aware of the probability of state transitions and the reward. Q-learning applies incremental updates with the current state, the next state, the action, and the immediate reward to approximate new Q-value. That's why Q-learning is an efficient algorithm for any environment especially large-scale environment. Based on the NoC state s_t (utilization of NoC resources) at current time step t, there may be several possible V/F actions to take. An RL agent chooses the action a that has the highest (currently estimated) Q-value among all possible actions (or chooses a random action with some probability). After taking the action, the agent transitions to a new state s_{t+1} (new utilization values of NoC resources) while in the meantime the NoC environment provides a reward r_t (to maximize NoC performance and energy efficiency). Q-learning algorithm tries to maximize the expected cumulative reward achievable from a given state-action pair (s_t, a_t), and it can approximate the optimal solution from Bellman Equation using the following iterative update for Q-value

$Q(s_t, a_t)$ in equation 1:

$$Q(s_t, a_t) = (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a)) \quad (1)$$

where, r_t is the reward observed for the current state s_t, α is the learning rate ($0 \leq \alpha \leq 1$) and $\max Q(s_{t+1}, a)$ is the maximum Q value over all possible actions a in next state s_{t+1}. The learning rate α determines the importance of new experience compared to past experience. A factor of 0 makes the agent learn nothing from new experience, while a factor of 1 makes the agent consider only the most recent information. Higher learning rate learns more from new experience and gives less priority to old information, where a learning rate of 1 makes the agent consider only the most recent information. The discount factor γ determines the importance of future rewards. Lower discount rate gives more importance to current rewards, where a factor of 0 only considers current rewards. A factor of 0 will make the agent considers only current rewards, while a factor approaching 1 will make it strive for a long-term high reward.

B. DEEP REINFORCEMENT LEARNING

The major disadvantage of RL is that the agent needs to maintain a mapping table for states, actions, and rewards, and this table grows exponentially with the increase in problem size in multi/many-core systems. Traditional Q-learning uses a table, namely Q-table, to store the Q-value $Q(s, a)$ for each state-action pair, as shown in Figure 2. As NoC features have continuous values, state-action space and corresponding Q-table can be large. For large-scale NoC with many resources (cores/routers/links), the system needs large number of agents and Q tables and therefore, the overall size and cost of Q tables will be extremely large. This state-action mapping table challenge of RL can be addressed by an approximate function of state, action, and reward. In this work, NN is used to approximate Q-function $Q(s, a)$ that estimates the future returns taking action a from state s. NN's function approximation removes the need for large state-action mapping table, which makes the proposed work scalable for large-scale systems. Given a state s, an NN can output a vector of approximated Q-values for each possible action a. Then, the action with the highest Q-value is chosen or a random action chosen with a small probability. This technique of combining RL and NN is called deep RL. Deep RL solutions have made many breakthrough to create something and/or to solve problems like to achieve human-level performance in AlphaGo [43] and Atari [36] games. The significance of deep RL contributions motivated us to apply deep RL to solve NoC optimization and configuration issues in multi/many-core systems.

NNs are used to approximate the state-action mapping table in an RL agent, as NNs have shown significant advantages in many domains such as image processing, speech recognition, and machine translation. NNs are used to discover the patterns in the NoC statistics (collected during current and past experiences) and predict the NoC V/F configuration actions with corresponding values (Q-values) for the next period. With NNs, we can easily extend the number of V/F-levels by just adding an additional output neuron in the output layer of NNs. An RL agent takes the V/F configuration decision with the maximum Q-value that minimize the latency and energy consumption of NoC. With a small probability, an RL agent also chooses random action (instead of best action) to avoid the local optimal solution (to achieve global optimal solution).

State-Action Mapping Table				
	a_0	a_1	a_2	a_3
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	$Q(s_0,a_3)$
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	$Q(s_1,a_3)$
...	...	...	...	...

FIGURE 2. Q-Table: State (s), Action (a), and Q-value (Q(s,a)).

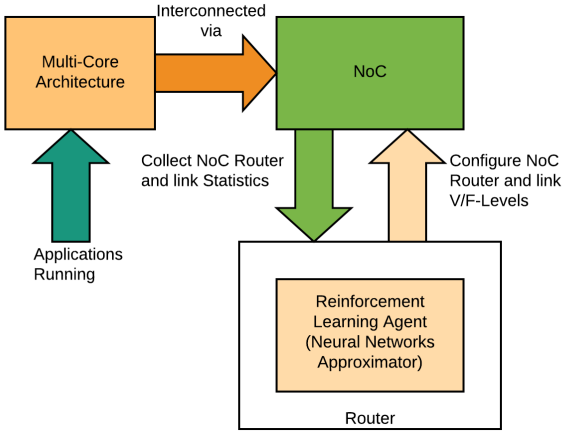

FIGURE 3. NoC configuration framework using deep RL.

The overall framework for dynamic NoC configuration using deep RL is shown in Figure 3. In the NoC configuration framework, an RL agent is integrated with each router for V/F-level configuration decision. Distributed RL agents collect NoC statistics in a fixed interval and configure the routers and links (connected to the router). Though RL agents take the local decisions, their target is to improve the global NoC performance by interacting with the NoC environment, as an RL agent checks the impact of its actions by evaluating NoC latency and power consumption. RL agents learn the best actions with time as it trains the NNs. The backpropagation algorithm [41] is used to train the NNs to learn the parameters of the NNs. Gradient descent approach is used to (back)propagate the prediction errors to learn the parameters of NNs for improving Q-value prediction decision. Upon receiving a decision from the RL agent, the DVFS controller with the help of voltage regulator selects the appropriate

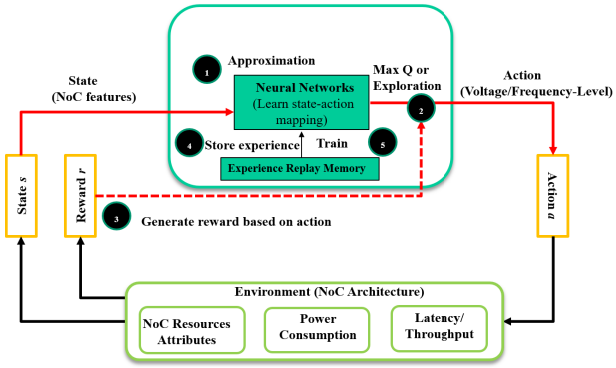

FIGURE 4. Deep RL model components and flow for self-configurable NoC.

V/F-level. A synchronization is needed between two routers as V/F-levels may be different for routers.

C. COMPONENTS OF THE DEEP RL MODEL FOR NoC CONFIGURATION

An RL model contains state, action, environment, and reward components. The deep RL model further contains NN component. The components of the proposed deep RL model for self-configurable NoC are shown in Figure 4 and are described below.

1) ENVIRONMENT (ARCHITECTURE)

The environment of the NoC configuration framework consists of routers, links, and processing cores. The environment generates the reward in terms of NoC latency and power consumption for the taken NoC V/F-level configuration decision by an RL agent (at a router) depending on the current state (which is discussed in the next section) of the router and associated links.

2) STATE (AGENT's VIEW OF THE ARCHITECTURE)

A vector of features is defined as a system state. Feature selection is very important for an RL model. If the features do not correlate to the action (NoC configuration decision), then an RL algorithm chosen to create the model will not be able to predict the outcome. Therefore, a great deal of thought are placed in the features selected for the RL model. The selected features are the utilization and capacity of the NoC resources: router and link. Flit count in the buffer of the router and link is used to calculate router and link utilization. Three features are used in a state vector, as follows: flit count, % of buffer utilization, and % of link utilization. V/F-level configuration decision is taken based on the utilization and capacity of the link and router resources. The number of features are kept to a minimum in order to decrease the amount of time needed to calculate the predicted Q-values at run-time. Also the features are selected by using the information already present at each router. We collect performance data to evaluate the performance of the model and collect energy consumption data to evaluate whether the decision from the RL model meets the global energy budget constraint.

3) ACTION (RESOURCE CONFIGURATION)

The agent takes online decision to configure V/F-levels setting for NoC routers and links based on the utilization of NoC resources. As mentioned before, the following V/F-level pairs are used in this RL model: 0.8 V/1 GHz, 0.9 V/1.5 GHz, 1.0 V/1.8 GHz, and 1.1 V/2 GHz. However, the V/F-level pairs are configurable based on the availability of V/F-levels. The agent generates vector of Q-values for each V/F-level configuration action. Figure 5 shows the Q-values for four different V/F configurations based on the router and link features at a router port. NN approximates the Q-values for different V/F configurations, and RL agent generally selects the V/F configuration with the maximum Q-value, which is the best action to maximize NoC performance. However, in this work, an RL agent selects exploration or exploitation policy with random probability ϵ to avoid the local optimal solution. ϵ parameter controls the amount of exploration vs. exploitation. In the exploitation phase, with probability $1-\epsilon$, an RL agent selects V/F-level configuration action with the maximum Q-value from the trained NNs. In the exploration phase, with probability ϵ, an RL agent takes random decision (instead of best action from trained NNs) to reduce the probability of local optimal solution. for example, for ϵ value of 0.1, an RL agent takes random decision with 10% probability or takes best Q-value actions with 90% probability. The value of ϵ can be tuned based on the performance of the agent and the demands of the application(s). ϵ value should be set to higher value (e.g., 0.5 or 50%) initially as an RL agent needs to explore more to learn the impact of its actions as the knowledge of the RL agent may not be complete. ϵ values can be reduced with time as an RL agent becomes more skilled over time in taking correct decisions by learning actions and corresponding energy and performance impact on the NoC environment. Because of the exploration capability, the proposed RL agent selects optimal action almost all the time after a certain number of training steps. In this work, we stop the training of an agent after the convergence is achieved as the agent can correctly predicts the correct actions.

4) REWARD FUNCTION

An RL agent gets feedback for its action from the NoC environment in terms of positive or negative reward. Depending on the V/F-level configuration action in the current state, the environment generates a reward, based on the optimization target, and sends it to the agent. The reward determines how good the taken action is, and Q-learning algorithm is designed to maximize the long-term reward. In this run-time configuration context, a reward function is used to maximize NoC performance and energy efficiency. The product of latency and power is used in the reward function to minimize both latency and power consumption of NoC, as shown in equation 2. The negative reward is used to reduce latency and power consumption. Lower negative values of latency and

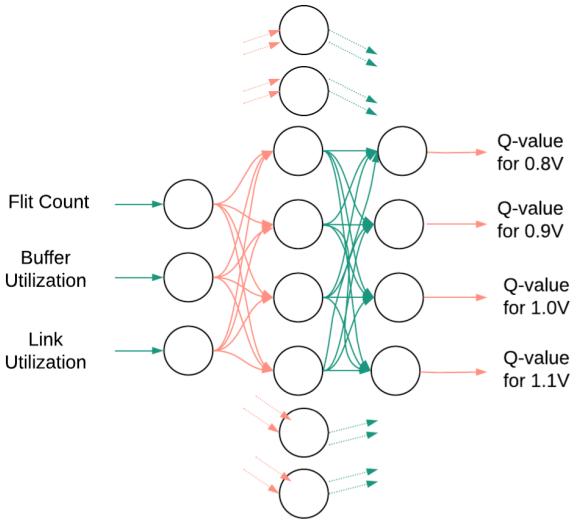

FIGURE 5. NoC features and actions for voltage-level configuration using a neural network.

power consumption means higher performance and energy efficient, respectively, NoC. Higher value of latency and/or power make the reward value more negative, and an RL agent tries to achieve lower negative value of reward.

$$reward = -latency * power \tag{2}$$

where, the unit of reward is joules, where the units of power and latency are watt and (nano) seconds, respectively.

5) NEURAL NETWORK FOR FUNCTION APPROXIMATION

Since state attributes (e.g., buffer and link utilization) are continuous numbers, number of states can be infinite that leads to large state-action mapping tables and high convergence time for Q-learning. NNs are used for action-function approximation in RL agents to solve the the problems (large state-action mapping tables and high convergence time) associated with traditional Q-learning solution. An NN provides a mechanism for generalizing training experience across states. Therefore, it is no longer necessary to keep the state-action mapping table for RL agent's decisions. An NN approximates the vector of Q-value scores for possible V/F-level configuration actions, and the RL agent selects the action with the maximum score (or random action sometimes) for maximizing the NoC performance and energy efficiency. The proposed deep RL approach uses fully connected NNs with three layers: input layer with three neurons (for three features), hidden layer with 8 neurons and an output layer with 4 neurons. Three layers for an NN and 8 neurons per hidden layer are chosen empirically as it provides sufficient accuracy with lower hardware overhead. *Sigmoid* function is used at the neurons of the hidden layers, where the activation value ranges from 0 to 1. *ReLU* activation function is used at the output layer neurons as Q-values can be greater than 1 (for the corresponding V/F-levels). The layers of the NNs are computed in sequential order, for example, first layer must gather features and compute its values before the next layer can be computed. In NNs, it is possible to parallelize all units and operations within each layer. Because of the parallel nature of operations, the computation time reduces significantly in NNs and increases the speed of prediction.

NNs are trained to approximate the Q-values for actions (of RL agents) based on the current state (features), past learning experience, and target optimization objective (as reflected in reward function) for NoC configuration. θ parameters (weights) decides a prediction of an NN using the input parameters from NoC statistics. An NN is trained with input state (old state) and target Q-value data collected through experience to learn θ parameters. The backpropagation algorithm [41] using gradient descent method is used to learn the θ parameters. The target Q-values are calculated using Q-learning equation (as shown in equation 1). The backpropagation algorithm works by computing the gradient of the loss function, as shown in equation 3. The loss function is the squared error difference between predicted Q-value Q and target Q-value Q'.

$$loss_Q = (r + \gamma \cdot \max_{a_{t+1}} Q'(s_{t+1}, a_{t+1}) - Q(s_t, a_t))^2 \tag{3}$$

In each training step, the backpropagation algorithm computes the gradient with respect to θ parameters of the prediction network. The backpropagation algorithm computes gradient one layer at a time and iterates backward from the last layer (based on the feedback from target Q-value) to update and to learn θ parameters of NNs. The goal of learning θ parameters is to correctly approximate Q-value. The use of NNs (instead of state-action mapping table) makes the RL agents robust to different demands of the applications because of the generalization property of NNs.

D. STABILITY OF REINFORCEMENT LEARNING AGENT

RL suffers instability when the Q-value function is approximated with a non-linear supervised learning, like logistic regressions and NNs [36]. The following techniques are adapted to improve the stability (reduce the fluctuation) of predictions from deep RL.

1) EXPERIENCE REPLAY FOR DEEP RL STABILITY

Similarity of subsequent training samples can lead an NN generalization into local minima because of the correlation between the current approximate action-value and the target action value. Experience replay breaks up the correlation in data to stabilize deep RL technique [34]. Instead of training RL agent on state-action pairs as they occur during simulation or actual experience, the system stores the data discovered (state, action, reward, next state) during state transitions, in a table with limited entries (to reduce hardware overhead). The system then randomly selects the experience data from the table to provide decorrelated data to train the NNs. Experience replay also allows the model to learn past experience multiple times, as it can randomly select same data. This learning strategy leads to faster convergence. In the

experience replay table, the oldest data is overwritten by a new data (as the size of the table is kept limited to reduce hardware overhead).

2) SEPARATE TARGET NETWORK FOR DEEP RL STABILITY

The second modification to online deep RL learning is to use a separate NN for generating the target Q-values in the Q-learning update. This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases current state Q-value also increases the target Q-value, possibly leading to oscillations of the policy. Target and predictor NNs are trained at different intervals. The predictor network parameters are updated in each training step, while the target network parameters are updated periodically. For example, we set the training intervals for predictor and target NNs to 50 cycles and 100 cycles (in simulation), respectively. This policy adds a delay between target and predictor network parameters updates, and this delay helps to reduce the impact of predictor network on target network parameters, which results in the reduction of divergence in approximation (using NNs). Generating the targets using an older set of parameters adds a delay between the time an update to predict Q-value is made (each training step) and the time the update affects the target Q-value (periodically), making divergence or oscillations much more unlikely.

E. DEEP RL MODEL TRAINING AND LEARNING ALGORITHM FOR NoC CONFIGURATION

The simplified pseudocode of the deep RL algorithm for NoC configuration is shown in Algorithm 1. Firstly, the algorithm initializes both NN predictor and target NN using random weights so that predictor can take action in its initial periods. Based on the current state of a router (from the router statistics), RL agent takes V/F-level configuration decision for the router and associated links for the next period. RL agent uses exploration or exploitation phase to take the decision depending on the random probability, ϵ. In the exploration phase, RL agent takes random decision to reduce the probability of local optimal solution (to get global optimal solution). In the exploitation phase, RL agent uses the trained NNs to select the V/F-level configuration action with the maximum Q-value. RL agent considers router and all the associated links of a router at a time for V/F-level configurations.

Because of the cost of training in terms of computation and time, we train the predictor NN and target NN in fixed intervals instead of per cycle training. As mentioned before, training intervals are different for predictor and target networks to improve the deep RL stability. We set the training intervals for predictor and target NNs to 50 cycles and 100 cycles, respectively. Q-values for current state, $Q_{current\_predictors}$, are predicted by the predictor NN. Q-values targets, $y_{current\_targets}$, are set by using the predicted Q-values from target Q-network and the reward from the environment. Then training for predictor network to update connection weights is done by using the square loss of the calculated Q-values from predictor and target Q-networks, $(y_{current\_targets} - Q_{current\_predictors})^2$. For training, we sample random records from the experience replay table to form mini-batches and then use gradient descent approach to back propagate this error to the hidden layer to tune their weights. The mini-batch size is set to 50 in

Algorithm 1: Deep RL Algorithm for NoC Configuration

input : NoC Features: Flit count, Buffer Utilization, Link Utilization
output: NoC V/F-Levels Configuration
Initialize NN predictor and target NN using random weights;
Initialize Experience Replay memory buffer;
while *simulation cycle_count* $\leq$ *maximum number of cycles or packet_count* $\leq$ *maximum number of packets per node threshold* **do**
  Step 1: Monitor the traffic in router and links of NoC (as state);
  Step 2: With probability ϵ, select a random action configuration or with probability $1 - \epsilon$ predict the best action configuration for current NoC state using NNs;
  Step 3: Configure V/F-levels of NoC router and links using action in previous step;
  Step 4:
  Observe reward (-latency*power) of current action and find next state (NoC utilization);
  Store experience (state, action, reward, next state) into the Experience Replay buffer;
  Sample the random records from the replay memory to form mini-batches;
  Step 5:
  Calculate the predicted Q-values using predictor Q-network for different V/F actions in the current state, $Q_{current\_predictors}$;
  Calculate the target Q-values using target Q-network for different V/F actions (a') in the next state, Q_{next_state};
  Set Q-values targets as $y_{current\_targets} = reward + \gamma \max_{a'} Q_{next\_state}$;
  Train and update the connection weights of the predictor Q-network using the square loss of calculated predicted Q-values (from predictor Q-network) and target Q-values (from target Q-network) using sample data from replay memory, $(y_{current\_targets} - Q_{current\_predictors})^2$;
  Step 6:
  if *cycle_count* == *target_interval* **then**
    Update the NN connection weights of the target Q-network as same weights as the predictor Q-network for the next period;
  end
  cycle_count = *cycle_count* + *predictor_interval*;
end

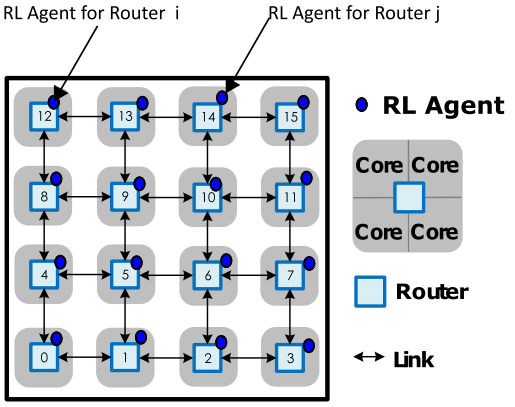

FIGURE 6. Distributed RL agents based optimization in a 4 × 4 CMesh NoC based 64-Core system.

this work. θ parameters (connection weights of NNs) of the target Q-network are set to the same values as θ parameters of the predictor Q-network after a fixed interval (set as target Q-network training interval). The algorithm runs until the maximum number of simulation cycles reached or packet count per node exceeds the maximum number of packets per node threshold.

F. DISTRIBUTED REINFORCEMENT LEARNING FOR LARGE-SCALE NoC

In a NoC based manycore archiecture, it is not feasible to implement centralized RL (or other ML) agent monitoring and decision framework for all the NoC resources due to exponential increase in communication delay with the distance of the nodes from the centralized controller. Decision (dynamic configuration) delay increases as every node has to communicate with the centralized RL agent for local NoC router and links configuration decisions. As configuration decision delay increases, that decision could be too late for its effective application in real-time systems, which could results in opportunity loss to reduce energy consumption and/or to improve performance of NoC. Furthermore, centralized decision maker can become a communication bottleneck in the manycore NoC as many communications are going through that one controller node. Hotspot can form in the centralized controller and packets passing through the hotspot area will be delayed as the NoC resources (e.g., buffers and links) are already occupied by the packets in the hotspot region. This contention/delay problem in the centralized controller region can propagate throughout the NoC. Distributed RL technique is needed in manycore NoC to reduce the above mentioned problems (high communication delay and hotspots) in centralized RL technique. A distributed per-router RL agent based framework in a 4 × 4 CMesh NoC is presented in Figure 6. In the CMesh NoC architecture, a router is connected with four cores, which results in total 64 cores as shown in Figure 6. At a router, an RL agent can control the configuration decisions of the router itself and associated all the NoC outgoing links to meet the communication demands through that node. RL agent computes the best voltage-level configuration decisions based on experience from previous historical data. Initially, RL agents could provide local optimal solution. With more training, RL agents have better understanding of the impact of their actions in overall NoC, as reward value is changed based on latency and energy consumption of the whole NoC. Because all features are local to the router, even if more routers/cores were added to the network, the proposed deep RL-enabled model could be easily scaled to the new architecture with no change in the algorithm.

G. HARDWARE OVERHEAD FOR DEEP REINFORCEMENT LEARNING

Each new feature for deep RL increases the arithmetic overhead. That's why the number of features for deep RL is kept to a small value to reduce hardware overhead. The proposed deep RL uses fully connected NNs with three layers. As the proposed approach uses three features for configuration decisions, it needs three (3) neurons in the input layer. Similarly, it needs four (4) output neurons in the output layer as the proposed approach uses 4 V/F-levels as Q-values. We have empirically selected eight (8) hidden neurons as 8-neuron provides good performance. For input layer to hidden layer connections, NNs have a total of 24 multiplies, 16 additions, and 8 comparisons. For hidden layer to output layer, NNs have a total of 32 multiplies, 28 additions, and 4 comparisons. This equates to a total of 56 multiplies, 44 additions, and 12 comparisons to gather the features, compute state-action values, and to select the voltage-level with the largest action value. In 45nm, the energy cost of a single 16-bit floating point add is estimated to be 0.4 pJ and the area cost is 1360 um^2 [23]. The energy cost of a multiply is estimated to be 1.1 pJ and the area cost is 1640 um^2 in 45nm [23]. Therefore, for a single RL agent, the total energy overhead is 75.6 pJ and the total area overhead is 0.168 mm^2. As we are using separate target network for deep RL stability, overall overhead is multiplied by two. So the total energy overhead is 151.2 pJ and the total area overhead is 0.3366 mm^2. Additionally, an RL agent needs a buffer to hold the historical data for experience replay. Because of the limited entries (e.g., 200 entries) in the buffer, the overhead of the buffer is not significant enough. In the proposed distributed RL algorithm, per router RL agent decides the V/F action values based on the feature values (state) of the associated router and links of NoC. Though the energy and area cost of an RL agent is not significant enough, for a large scale NoC, such as 25 × 25 NoC, the overhead of many RL agents can be high. To address that, we propose CMesh NoC, where a single RL agent at a router supports multiple cores in the NoC. The number of concentrated cores per router can be configured based on NoC size, performance, and allowed hardware overhead. In this work, four cores are connected to a router for large-scale NoCs (64-core and 256-core). This CMesh NoC approach makes the overhead (both energy and area) of distributed RL implementation feasible for large-scale computing systems.

IV. SIMULATION AND RESULTS

Real system experiments are carried out using gem5 [7] and Garnet [3] platforms to evaluate the NoC performance and energy efficiency. The following four different V/F-levels are used for NoC configurations: 0.8 V/1 GHz, 0.9 V/1.5 GHz, 1.0 V/2 GHz, 1.1 V/2.5 GHz. The DSENT [44] tool is integrated with gem5 to evaluate the energy/power consumption. The gem5 and DSENT configurations are shown in Tables. 1 and 2, respectively. The proposed deep RL algorithm is implemented and integrated in Garnet for dynamic configuration of NoC. An RL agent module is added with each router in the Garnet. The NN function approximator implementation consists of one input layer, one hidden layer, and one output layer. *Sigmoid* and *ReLU* activation functions are used for hidden and output layers, respectively. We use three features at the router for V/F-level configuration prediction: flit count, % of buffer utilization, and % of link utilization. The buffer utilization is calculated as the number of active buffers in the previous interval divided by the total number of buffers at a router. Similarly, the link utilization is calculated as the number of active links in the previous interval divided by the total number of NoC links. A reward function as the negative product of latency and power is used to evaluate the V/F-level configuration actions to maximize performance and energy efficiency of NoC. ϵ value is set to 0.1, which means an RL agent takes a random action instead of the best action in 10% cases. ϵ value is not changed in this work as it (change in ϵ) did not significantly impact our results as the proposed RL agent selects optimal action almost all the time after a certain number of training steps for ϵ value of 0.1. Learning rate α is set to 0.5.

TABLE 1. gem5 configurations for 256-core CMesh NoC.

Instruction Set Architecture (ISA)	ALPHA
CPU Frequency	2GHz
Interconnection Networks	GARNET
No. of Virtual Networks(VN)	3
No. of Virtual channels(VC) per VN	4
No. of Buffers per VC	4
Network Size	8×8 Mesh
No. of Cores	256
No. of Routers	64
No. of Cores per Router	4
Routing	XY-Routing
Flit Width	128-bit
Max. Packets per Node	50000
Simulation Cycle	1000

We evaluate the proposed deep RL based NoC configuration model using communication-observant schedulable memory-inclusive computation (COSMIC) benchmark suite [50] and embedded system synthesis benchmarks suite (E3S) [18]. For large-scale problems, we have simulated 256-core connected through 8×8 CMesh architecture, where a router is connected to four computing nodes (cores), and simulated large applications in the COSMIC benchmark suite. The COSMIC benchmark suite is used for large-scale problem simulation because it (COSMIC) is based on real applications with a large number of tasks per application. The following five applications from the COSMIC benchmark suite are mapped in our simulations: face recognition, cifar, ultrasound imaging, reed-solomon code decoder (RS-Decoder), and reed-solomon code encoder (RS-Encoder). The face recognition (face), cifar (cfr), ultrasound (ultra), RS-Decoder (RSD), and RS-Encoder (RSE) application have 33,33, 526, 527, and 141 tasks, respectively. For small-scale problems, we have simulated 16-core connected through 4×4 2D-Mesh NoC, and simulated applications in the E3S benchmark suite. The E3S benchmark suite comprises applications in consumer (C), autoindustry (A), networking (N), telecom (T) and office-automation (O). E3S applications also have several sub-applications within themselves, for example, telecom application has 8 sub-applications. In overall, C, A, N, T, and O applications have 12, 24, 13, 28, and 5 tasks, respectively. We also simulate complex multi-application domain systems by mixing multiple applications in the COSMIC and E3S benchmarks, e.g., FaceCfr for face recognition and cifar applications and CN for consumer and networking applications. Furthermore, we evaluate the proposed deep RL based NoC configuration model using the following eight synthetic traffic patterns: (i) Uniform Random (UniR): destinations are randomly selected with a uniform distribution; (ii) Bit-complement (BitC): each node sends messages only to the node corresponding to the 1's complement of its own address; (iii) Tornado: every node i sends a packet to node $(i+3) mod 8$; (iv) Bit Rotation (BitRt): destination is found by circular shifting of the bits of the source; (v) Neighbor (NBor): node sends messages to only its neighbors; (vi) Shuffle (Shuf): destination is calculated by using source address and number of destinations; (vii) Bit Reverse (BitRv): destination is found by reversing the bits of the source; (viii) Transpose (Trans): node (x,y) sends messages only to (y,x).

TABLE 2. DSENT configurations for 256-core CMesh NoC.

Technology	22nm
Operating Frequency	1 GHz
No. of Virtual Networks(VN)	3
No. of Virtual channels(VC) per VN	4
No. of Buffers per VC	4
Network Type	CMesh
No. of Cores	256
No. of Routers	64
No. of Cores per Router	4
No. of Input Ports	5
No. of Output Ports	5
Flit Width	128-bit
Buffer Model	128-bit
Crossbar Model	Multiplexer
Switch Allocator Model	MatrixArbiter

We compared our solution with a non-ML based NoC configuration solution in [40]. In this work [40], the system managers using heuristic (non-ML) increase or decrease the

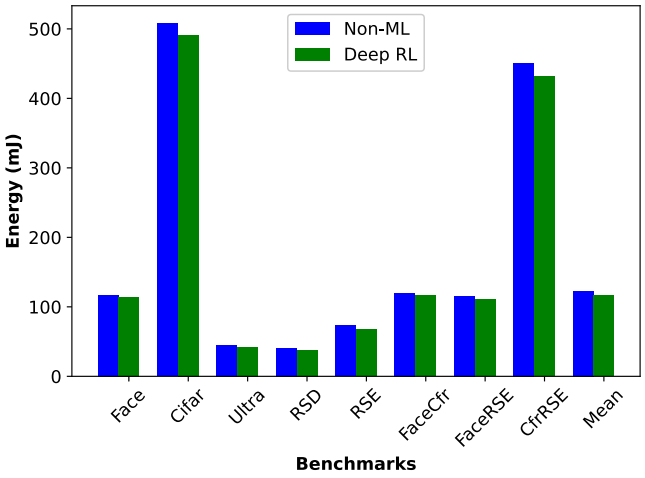

FIGURE 7. Energy comparison under COSMIC benchmarks in 256-core NoC.

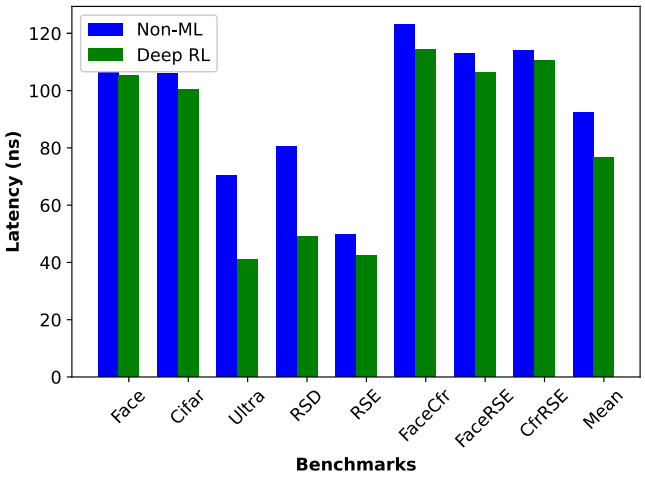

FIGURE 8. Latency comparison under COSMIC benchmarks in 256-core NoC.

V/F-levels of the NoC resources depending on the change in application traffic. We have evaluated our NoC configuration framework using latency, throughput, and energy metrics. For the COSMIC and E3S benchmarks, applications are initially mapped to NoC based system using the (same) load-balanced mapping solution in [40] for both ML and non-ML solutions for fair comparisons. The total (global) energy budget constraint is configured depending on the demands of the applications in a benchmark. The global energy budget constraint is set to 1.5 joule (J) for COSMIC benchmarks, and it (global energy budget constraint) is set to 50 mJ (milliJoule) for both E3S benchmarks and synthetic traffic patterns. Higher energy budget constraint is used for COSMIC benchmarks (compared to other benchmarks) because of the larger number of tasks and demands of the applications.

A. PERFORMANCE UNDER LARGE-SCALE NoC AND APPLICATION SETTINGS

Figure 7 shows the energy consumption for various applications in COSMIC benchmarks for all techniques.

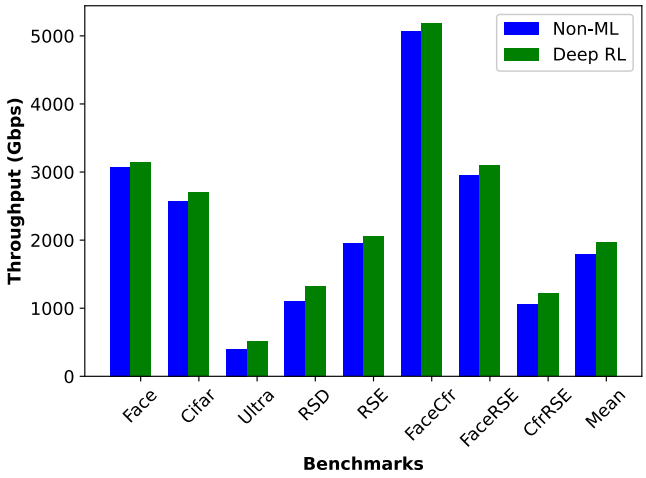

FIGURE 9. Throughput comparison under COSMIC benchmarks in 256-core NoC.

Simulation results for applications in COSMIC benchmarks suggest that latency improvement is significant (throughput also improves) in the proposed self-configurable NoC solution, while minimizing the energy consumption in the system. The proposed approach improves (reduces) energy consumption by up to 9% (by 6% on average) compared to the non-ML based configuration solution. Energy consumption decreases due to lower energy consumption in the NoC because of the proactive V/F-levels configuration using RL to maximize reward, where reward is configured to minimize energy (and latency). RL assigns the required V/F-levels by learning from the historical data. As RL agents learn the policy to maximize the reward (product of latency and power), communication latency in NoC improves significantly because of the lower queuing latency at NoC routers. The proposed approach improves latency by up to 70% (by 25% on average) compared to the non-ML based configuration solution, as shown in Figure 8. Because of the reduction of latency, throughput in the proposed solution improves by 10% (on average) compared to the non-ML solution, as shown in Figure 9. Furthermore, we observe that the proposed approach also improves energy, latency, and throughput more for all the multiple application mixes running in the system. This further indicates the effectiveness of the proposed approach for running multiple applications in manycore architectures.

B. PERFORMANCE UNDER SMALL-SCALE NoC AND APPLICATION SETTINGS

E3S applications are simulated under 16-core architecture connected through 2D-Mesh NoC. Like the COSMIC benchmarks results, simulation results for the E3S benchmark suggest that latency improvement is significant (throughput also improves) in the proposed self-configurable NoC solution, while minimizing the energy consumption in the system. As shown in Figure 10, the proposed configurable solution improves energy consumption by up to 14% (by 5% on average) compared to the non-ML solution. Energy improvement

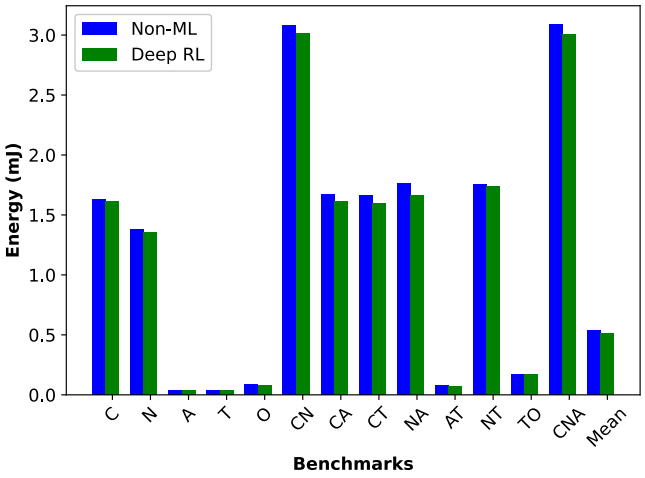

FIGURE 10. Energy comparison under E3S benchmarks in 16-core NoC.

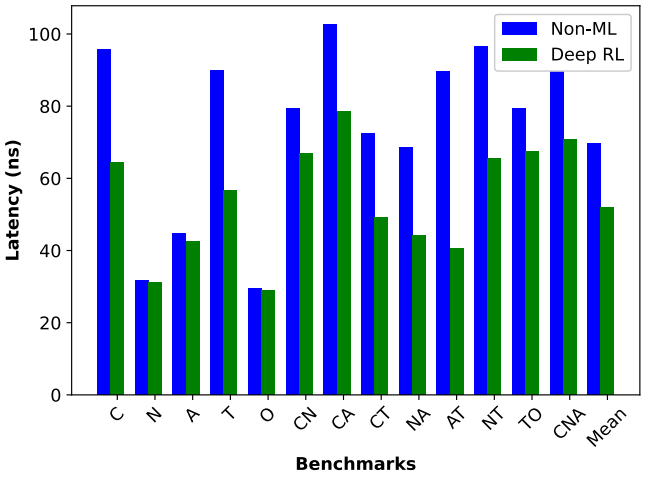

FIGURE 11. Latency comparison under E3S benchmarks in 16-core NoC.

is higher because of the dynamic heterogeneous assignment of required V/F-levels to NoC routers and links using online RL solution (instead of assigning homogeneous maximum voltages). As RL agents learn the policy to minimize latency, the proposed approach improves latency by up to 120% (by 34% on average) compared to the non-ML based configuration solution, as shown in Figure 11. Because of the reduction in flit transmission delay, throughput in the proposed solution improves by 7% on average compared to the non-ML solution, as shown in Figure 12.

Furthermore, the proposed RL-enabled approach performs better for mixture of applications in the E3S benchmark suite for all the metrics (energy, latency, and throughput). For example, the proposed approach improves latency by 120% for combined autoindustry (A) and telecom (T) applications, while the improvement is 30%, on average, for individual applications running separately. This indicates the feasibility of the proposed approach for manycore systems running multiple applications. For two single applications, networking (N) and office-automation (O), the proposed approach did not significantly improve (degrades in one case) energy, latency, and throughput mainly due to the presence of lower number of tasks and communication traffic (and so less opportunity of improvement) in those applications.

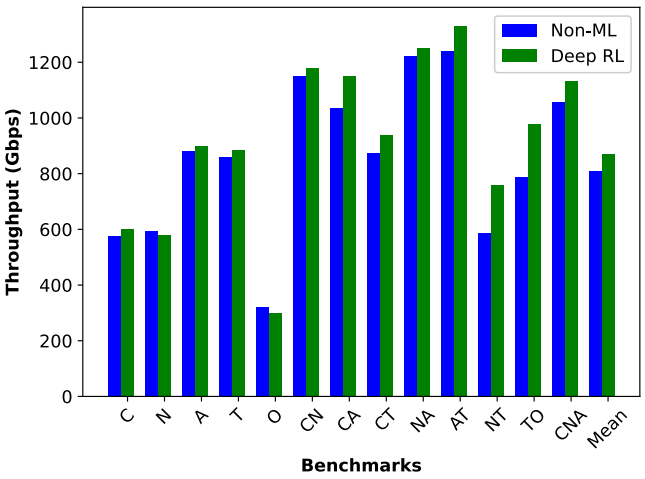

FIGURE 12. Throughput comparison under E3S benchmarks in 16-core NoC.

C. PERFORMANCE UNDER SYNTHETIC TRAFFIC

For synthetic traffic patterns, we compare the proposed deep RL based run-time heterogeneous NoC configuration with static homogeneous NoC configuration solution in 16-core architecture. We analyze whether RL can improve the energy, latency, and throughput even in regular NoC (without the help of mapping). As shown in Figure 13, the proposed configurable NoC solution improves energy by up to 17% (by 8% on average) compared to the homogeneous static NoC solution. Energy consumption decreases because of the proactive resource configuration of the routers and links using RL based on the past historical traffic requirements and system utilization (e.g., link and buffer utilization). As RL agents

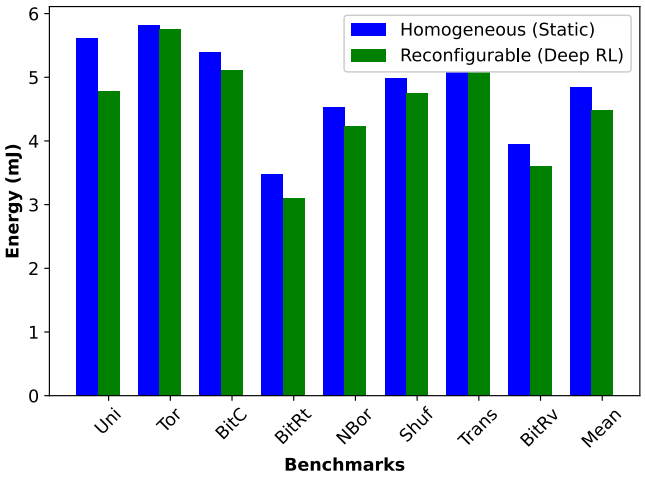

FIGURE 13. Energy comparison under synthetic traffic in 16-core NoC.

learn the policy to maximize the reward under constrained energy consumption, the proposed approach, on average, improves latency by 25% compared to the non-ML based

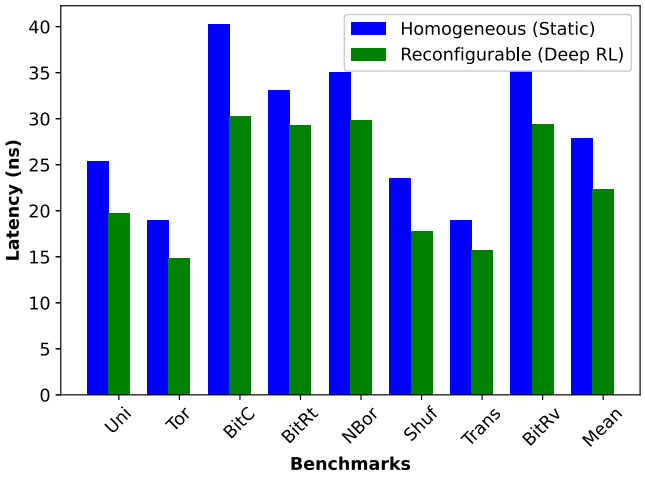

FIGURE 14. Latency comparison under synthetic traffic in 16-core NoC.

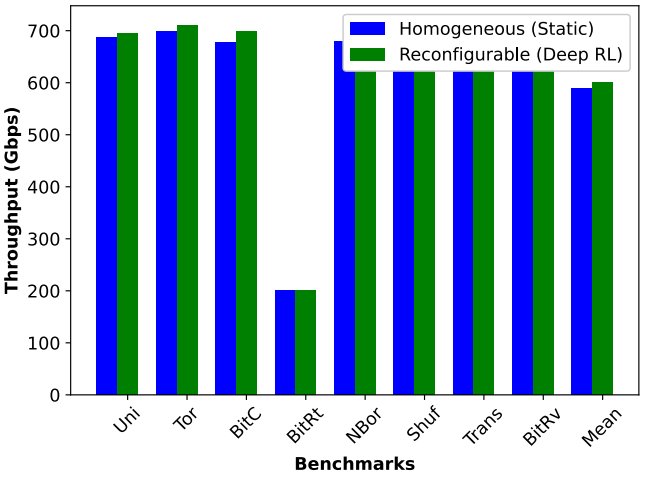

FIGURE 15. Throughput comparison under synthetic traffic in 16-core NoC.

configuration solution, as shown in Figure 14. Throughput remains almost same or slightly improved (by 2%) in the configurable deep RL solution, as shown in Figure 15.

D. HOTSPOT REDUCTION

This proposed work uses a load-balanced mapping solution, which was adopted from [40], to evenly distribute computation and communication demands of the tasks among nodes (cores/routers) in NoC to prevent hotspots. Hotspots increase the probability of several problems, including electromigration, burning chip, and fault. This work only focuses only on communication energy hotspots in NoC. The energy consumption at a router is calculated by summing up the energy consumption on the router and on the associated communication links (to adjacent routers). Figure 16 shows router-wise energy distribution in 16-core NoC for a mixture of applications (consumer (C), networking(N), and auto-industry(A)) under E3S benchmarks. Router-wise energy distribution results show that energy is almost uniformly distributed among the routers and the links except the edge/corner routers/links in 2D-Mesh NoC, for example, routers 0, 4, and 12 in Figure 16. The balanced energy distribution reduces the possibility of hotspots in NoC.

E. SCALABILITY

Latency and throughput results are calculated for E3S benchmarks with the number of cores ranging from 16 to 256 (16, 64, and 256) to test the scalability of the proposed approach. Figure 17 shows that latency decreases with the increase in NoC resources as packets face less contention (and congestion) in NoC links and routers. Because of lower congestion, NoC throughput increases with the addition of more NoC resources, as shown in Figure 18. The latency and throughput performance results demonstrated that the proposed deep RL-enabled self-configurable NoC approach is scalable.

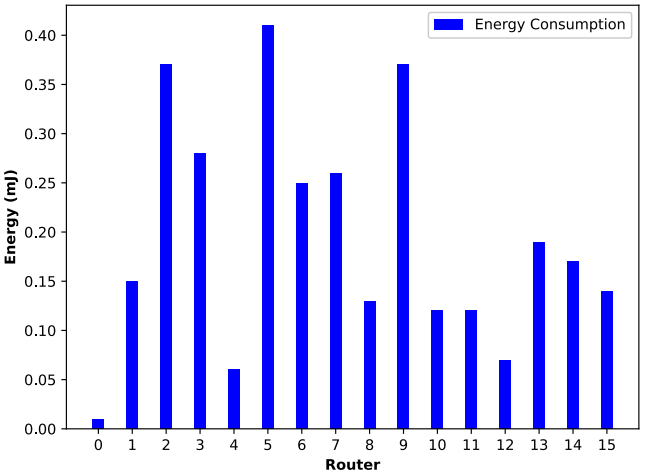

FIGURE 16. Router-wise energy distribution in 16-core NoC for CNA applications under E3S benchmarks.

F. REWARDS

The reward for an RL agent is calculated by considering both latency (delay) and energy consumption to maximize both performance and energy-efficiency. The proposed deep RL framework has distributed RL agents, and the Figure 19 shows the reward values with time (cycles) for a sample distributed RL agent. The reward value increases with time, as shown in Figure 19, which means that the RL agent is learning and becoming skillful in time. After around 500 cycles, the reward value saturates and it (reward) does not change, which means that the RL agent becomes expert in making decisions. Similar patterns of change in reward values with time have been observed in the other distributed RL agents.

G. COMPARISON WITH AN RL WORK

The proposed work has been compared with another ML work that configures the link bandwidths dynamically using RL for energy savings [38]. [38] changes the link width by

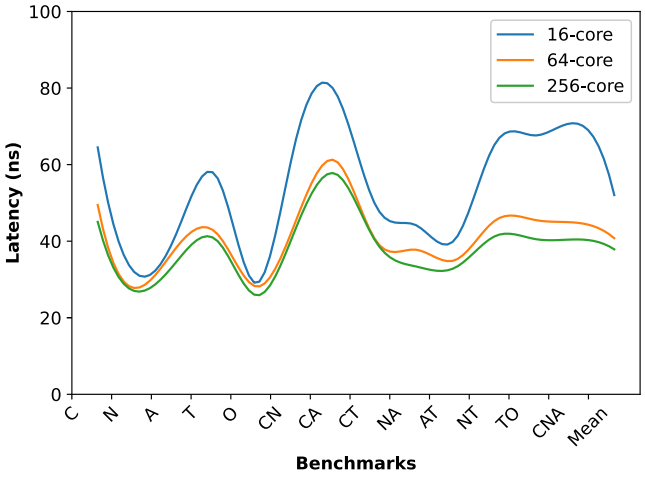

FIGURE 17. Latency for different NoC sizes under E3S benchmarks.

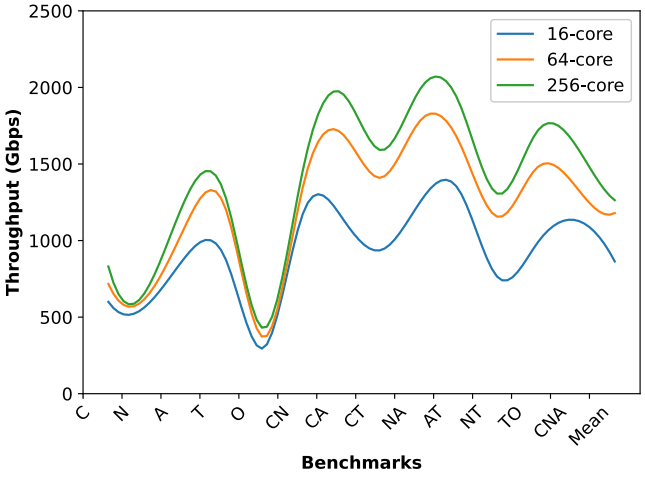

FIGURE 18. Throughput for different NoC sizes under E3S benchmarks.

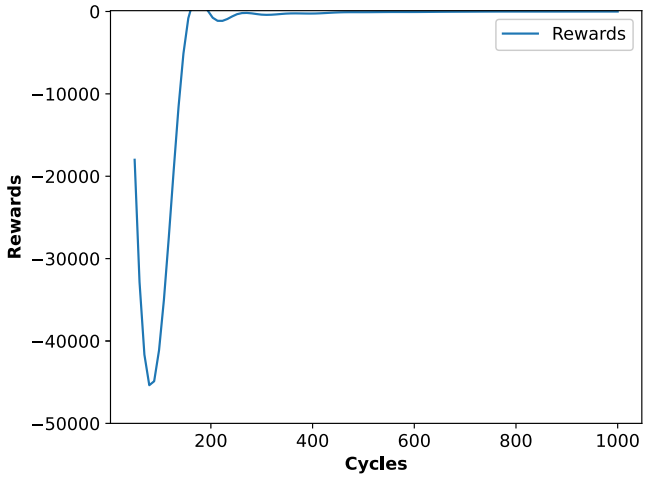

FIGURE 19. Rewards for a distributed RL agent.

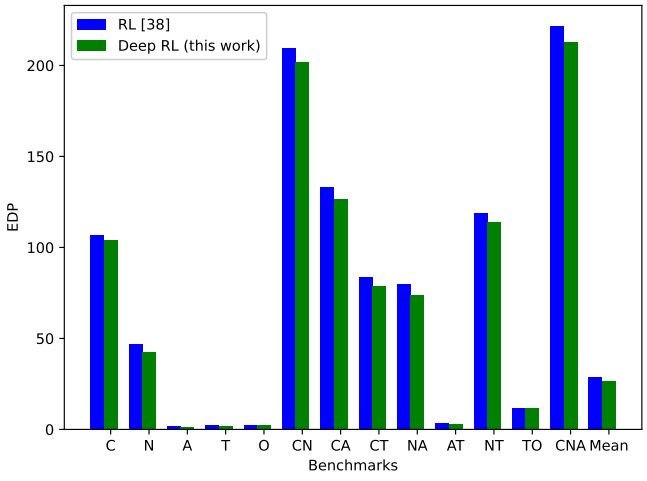

FIGURE 20. EDP comparison with an RL approach [38].

activating or deactivating wires on links using distributed RL agents, where an agent stores Q-values for state-action mapping in a table, namely Q-table. [38] is selected for comparison as the goal of [38] is to achieve energy-efficient NoC like this proposed work (proposed work also targets to achieve high performance). We found that the proposed neural networks-enabled deep RL approach performs better than the RL approach in [38] for NoC configuration in terms of both energy-efficiency and latency (delay). As shown in Figure 20, EDP in this proposed work is 8% (up to 17%) lower compared to that of [38]. And the proposed work performs better than [38] for all the applications under E3S benchmarks. The major reason for better EDP in this work is that the proposed solution focuses to address both performance (latency) and energy efficiency, where [38] focused mainly on energy-efficiency.

H. NoC AREA OVERHEAD

The area overhead of NoC is calculated using DSENT tool. In the concentrated mesh implementation with 4-cores per router, a router has 4 local ports. For connection with adjacent routers, a router has additional 2 ports (corner nodes) or 3 ports (edge nodes) or 4 ports based on the position of the router in the mesh architecture. The NoC area overhead is calculated using the number of buffers and ports at routers, the number of routers, and the bandwidths of the links. The hardware overhead for an RL agent is calculated in Sec. III-G. The NoC area overheads (excluding RL agents) for 16-core, 64-core, and 256-core NoCs are 4.51 mm^2, 18.02 mm^2, and 72.08 mm^2. The hardware overheads for RL agents for 16-core, 64-core, and 256-core NoCs are 1.34 mm^2, 5.38 mm^2, and 21.50 mm^2. So, the total NoC area overheads (including RL agents) for 16-core, 64-core, and 256-core NoCs are 5.85 mm^2, 23.40 mm^2, and 93.59 mm^2. Therefore, the hardware overhead for RL agents is 23% of the total NoC area overhead.

I. RESULTS SUMMARY

The uniqueness of the proposed self-configurable NoC solution is that the solution maximizes both performance and energy efficiency while it configures the NoC instantly

with the help of RL-based distributed intelligent agents. The proposed solution addresses the challenges of reactive and slow solutions from heuristics (as in [40]) and optimization solvers (e.g., IBM CPLEX [2]). Because of the reactive/slow solutions, non-ML based approach [40] cannot increase or decrease V/F-levels efficiently to provide both high-performance and energy-efficient solution. Furthermore, as shown in the simulation results, the proposed solution does not try to maximize the NoC performance by assigning maximum V/F-levels as high power consumption and corresponding high temperature, even in only one region of NoC, can create problems, including failure of electrical components [42], in systems. The proposed solution is scalable, as demonstrated for NoC with cores ranging from 16 to 256, and reduces energy hotspots in NoC. The proposed solution improves EDP by 27%, by 40%, and by 35% for COSMIC, E3S, and synthetic traffic benchmarks, respectively, compared to corresponding non-ML based solution [40] and improves EDP by 8% for E3S benchmarks compared to an RL-based solution [38]. Therefore, the proposed self-configurable NoC solution maximizes performance with limited power consumption to quickly provide energy-efficient and high-performance solution.

V. CONCLUSION

We have proposed dynamic configuration of on-chip networks on multi/many-core computing systems using machine learning techniques for energy-efficient and high-performance NoC. NoC is configured proactively based on the historical data using deep reinforcement learning, where distributed reinforcement learning agents take the voltage/frequency-level configuration actions for NoC routers and links using neural networks function approximators. The use of neural network in a reinforcement learning agent and distributed per-router reinforcement learning agents in NoC make the proposed approach feasible for large-scale systems. Simulation results under real and synthetic traffic demonstrate that the proposed machine learning-enabled self-configurable NoC solution improves NoC performance significantly while maximizing energy efficiency. The scalability and area overhead of the proposed approach have been demonstrated with 16-core, 64-core, and 256-core NoC architectures. The proposed solution incurs low hardware overhead for machine learning while providing self-configurable NoC to meet the real-time requirements of multiple applications.

REFERENCES

[1] *Cerebras Wafer Scale Engine*. Accessed: Jun. 16, 2022. [Online]. Available: https://www.cerebras.net/product/

[2] *IBM CPLEX Optimization Studio*. Accessed: Jun. 16, 2022. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio?mhsrc=ibmsearch_a&mhq=CPLEX

[3] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2009, pp. 33–42.

[4] Y. Bai, V. W. Lee, and E. Ipek, "Voltage regulator efficiency aware power management," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2017, pp. 825–838.

[5] L. Benini and G. De Micheli, "Networks on chip: A new paradigm for systems on chip design," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Nov. 2002, pp. 418–419.

[6] K. Bergman *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," Tech. Rep. TR-2008-13, 2008.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, and J. Hestness, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[8] B. Bohnenstiehl, A. Stillmaker, J. J. Pimentel, T. Andreas, B. Liu, A. T. Tran, E. Adeagbo, and B. M. Baas, "KiloCore: A 32-nm 1000-processor computational array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 891–902, Apr. 2017.

[9] S. Borkar, "Exascale computing—A fact or a fiction?" in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process. (IPDPS)*, May 2013, pp. 1–3.

[10] Y. Cai, C. Yang, W. Ma, and Y. Ao, "Extreme-scale realistic stencil computations on Sunway TaihuLight with ten million cores," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2018, pp. 566–571.

[11] X. Chen, Z. Xu, H. Kim, P. V. Gratz, J. Hu, M. Kishinevsky, U. Ogras, and R. Ayoub, "Dynamic voltage and frequency scaling for shared resources in multicore processor designs," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, May/Jun. 2013, pp. 1–7.

[12] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proc. IEEE/ACM Design, Autom., Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 1521–1526.

[13] M. Clark, R. Bunescu, A. Kodi, and A. Louri, "LEAD: Learning-enabled energy-aware dynamic voltage/frequency scaling in NoCs," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[14] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. 38th Design Autom. Conf.*, Jun. 2001, pp. 684–689.

[15] 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 *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 705–712.

[16] G. D. Micheli, C. Seiculescu, S. Murali, L. Benini, F. Angiolini, and A. Pullini, "Networks on chips: From research to products," in *Proc. ACM/IEEE DAC Design Autom. Conf.*, Jun. 2010, pp. 300–305.

[17] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE J. Solid-State Circuits*, vol. JSSC-9, no. 5, pp. 256–268, Oct. 1974.

[18] R. Dick. *Embedded System Synthesis Benchmarks Suites (E3S)*. [Online]. Available: http://robertdick.org/tools.html

[19] D. DiTomaso, T. Boraten, A. Kodi, and A. Louri, "Dynamic error mitigation in NoCs using intelligent prediction techniques," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[20] 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 Trans. Comput.*, vol. 68, no. 3, pp. 375–389, Mar. 2019.

[21] H. Hantao, P. D. S. Manoj, D. Xu, H. Yu, and Z. Hao, "Reinforcement learning based self-adaptive voltage-swing adjustment of 2.5 DI/Os for many-core microprocessor and memory communication," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2014, pp. 224–229.

[22] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, Aug. 2007, pp. 38–43.

[23] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[24] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sep. 2007.

[25] R. Jain, P. R. Panda, and S. Subramoney, "Machine learned machines: Adaptive co-optimization of caches, cores, and on-chip network," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 253–256.

[26] D.-C. Juan, H. Zhou, D. Marculescu, and X. Li, "A learning-based autoregressive model for fast transient thermal analysis of chip-multiprocessors," in *Proc. 17th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2012, pp. 597–602.

[27] E. Kakoulli, V. Soteriou, and T. Theocharides, "Intelligent hotspot prediction for network-on-chip-based multicore systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 3, pp. 418–431, Mar. 2012.
[28] M. Kas, "Toward on-chip datacenters: A perspective on general trends and on-chip particulars," *J. Supercomput.*, vol. 62, no. 1, pp. 214–226, Oct. 2012.
[29] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie, "Quantifying the energy cost of data movement in scientific applications," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Sep. 2013, pp. 56–65.
[30] R. G. Kim, W. Choi, Z. Chen, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Imitation learning for dynamic VFI control in large-scale manycore systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2458–2471, Sep. 2017.
[31] R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Machine learning and manycore systems design: A serendipitous symbiosis," *Computer*, vol. 51, no. 7, pp. 66–77, Jul. 2018.
[32] M. A. Kinsy, S. Khadka, and M. Isakov, "PreNoc: Neural network based predictive routing for network-on-chip architectures," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, May 2017, pp. 65–70.
[33] L. Shang, L. Peh, and N. K. Jha, "Power-efficient interconnection networks: Dynamic voltage scaling with links," *IEEE Comput. Archit. Lett.*, vol. 1, no. 1, pp. 1–6, Jan. 2002.
[34] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 293–321, May 1992.
[35] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, "A case for dynamic frequency tuning in on-chip networks," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2009, pp. 292–303.
[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, and A. Graves, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
[37] M. F. Reza, "Deep reinforcement learning for self-configurable NoC," in *Proc. IEEE 33rd Int. Syst.-Chip Conf. (SOCC)*, Sep. 2020, pp. 185–190.
[38] M. F. Reza, "Reinforcement learning based dynamic link configuration for energy-efficient NoC," in *Proc. IEEE 63rd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2020, pp. 468–473.
[39] M. F. Reza, T. T. Le, B. De, M. Bayoumi, and D. Zhao, "Neuro-NoC: Energy optimization in heterogeneous many-core NoC using neural networks in dark silicon era," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.
[40] M. F. Reza, D. Zhao, and M. Bayoumi, "Power-thermal aware balanced task-resource co-allocation in heterogeneous many CPU-GPU cores NoC in dark silicon era," in *Proc. 31st IEEE Int. Syst.-Chip Conf. (SOCC)*, Sep. 2018, pp. 260–265.
[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," in *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, pp. 696–699.
[42] L. Shang and R. P. Dick, "Thermal crisis: Challenges and potential solutions," *IEEE Potentials*, vol. 25, no. 5, pp. 31–35, Sep. 2006.
[43] 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," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
[44] C. Sun, C.-H.-O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "DSENT—A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proc. IEEE/ACM 6th Int. Symp. Netw.-Chip (NOCS)*, May 2012, pp. 201–210.
[45] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.
[46] Y. Tan, W. Liu, and Q. Qiu, "Adaptive power management using reinforcement learning," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, New York, NY, USA, 2009, pp. 461–467.
[47] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, Sep. 2013.
[48] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, Mar. 2002.
[49] 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 *Proc. 46th Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 589–600.
[50] Z. Wang, W. Liu, J. Xu, B. Li, R. Iyer, R. Illikkal, X. Wu, W. H. Mow, and W. Ye, "A case study on the communication and computation behaviors of real applications in NoC-based MPSoCs," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2014, pp. 480–485.
[51] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *Proc. IEEE 14th Int. Symp. High Perform. Comput. Archit.*, Feb. 2008, pp. 123–134.
[52] J. Yin, Y. Eckert, S. Che, M. Oskin, and G. H. Loh, "Toward more efficient NoC arbitration: A deep reinforcement learning approach," in *Proc. 1st Int. Workshop AI-Assist. Design Archit. (AIDArc)*, Jun. 2018, pp. 1–18.
[53] Y. Zhang, X. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2002, pp. 183–188.

MD FARHADUR REZA (Member, IEEE) received the B.Sc. degree in computer science and engineering from the Bangladesh University of Engineering and Technology, in 2005, the M.B.A. degree from the Institute of Business Administration (IBA), University of Dhaka, in 2011, and the M.Sc. and Ph.D. degrees in computer science from the University of Louisiana, Lafayette, in 2014 and 2017, respectively. He was an Assistant Professor of computer science at the University of Central Missouri. He was a Postdoctoral Researcher at Virginia Tech and at George Washington University, in 2019 and 2018, respectively. He has seven years of working experience in telecom and software industries. He is an Assistant Professor with the Department of Mathematics and Computer Science, Eastern Illinois University (EIU). He published papers in peer-reviewed journals and conferences, including ACM/IEEE The International Symposium on Networks-on-Chip (NOCS), ACM GLSVLSI, and Elsevier MICPRO. His research interests include resource management, networks-on-chip, multicore architectures, and machine learning/artificial intelligence. He is an ACM member. He received A. Richard Newton Young Fellow Award from ACM/IEEE Design Automation Conference (DAC), in 2014. He has been serving as a Publication Co-Chair for IEEE International System-on-Chip Conference (SOCC), since 2020. He is a Publicity Co-Chair for IEEE International Conference on Omni Layer Intelligent Systems (COINS) 2022. He is with the Technical Program Committee of several conferences, including IEEE International System-on-Chip Conference (SOCC) 2020-2022, IEEE International Conference on Omni Layer Intelligent Systems (COINS) 2020-2022, and IEEE/ACM International Workshop on Network on Chip Architectures (NoCArc) 2021. He organized a special session in IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS) 2022. He is serving as a Track Chair for the Internet of Things (IoT) Track in IEEE International Conference on Omni Layer Intelligent Systems (COINS). He served as a Session Chair for IEEE International System-on-Chip Conference (SOCC) and IEEE International Midwest Symposium on Circuits and Systems (MWSCAS) conferences. He serves in the Review Committee members for IEEE International Symposium on Circuits and Systems (ISCAS) and IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS) conferences.