Deep Reinforcement Learning for Self-Configurable NoC

Md Farhadur Reza
Email: reza@ucmo.edu
School of Computer Science and Mathematics, University of Central Missouri, Warrensburg, MO, USA

***Abstract*—Network-on-Chips (NoCs) 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 creating thermal hotspots and exceeding chip power budget, from happening in adaptive systems. Therefore, we propose machine learning (ML) technique to provide proactive solution within an instant for both energy and performance efficiency. In this paper, we present deep reinforcement learning (deep RL) techniques to configure the voltage/frequency levels of both NoC routers and links in multicore architectures for energy-efficiency while providing high-performance NoC. We propose the use of reinforcement learning (RL) to configure the NoC resources intelligently based on system utilization and application demands. Additionally, neural networks (NNs) are used to approximate the actions of distributed RL agents in large-scale systems, to mitigate the large cost of traditional table-based RL. Simulations results for 256-core and 16-core NoC architectures under real-world benchmarks show that the proposed approach improves energy-delay product significantly (40%) when compared to traditional non-ML based solution. Furthermore, the proposed solution incurs very low energy and hardware overhead while providing self-configurable NoC to meet the real-time requirements of applications.**

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. Many-core on-chip systems have better power efficiency, interconnection, and latency compared to traditional local area network (LAN) based systems [12]. Industry and academia are working on many-core single chip solution that could replace the traditional big data-center solution [1], [4]. For example, Cerebras have developed a single chip with 400K independent cores using 1.2 trillion transistors [1]. As the number of cores increases to hundreds and thousands, network-on-chip (NoC) has been adopted as a viable solution to manage the complex on-chip communication [16]. However, as the number of cores on the chip increases, the NoC as it turns out is consuming a significant percentage of the total chip power. For example, research has shown that on-chip network can consume 10-30% of total chip power [9]. NoC power consumption has been steadily increasing with the increase in the number of cores that are integrated on the chip. Because of the gap between transistor density and transistor power efficiency in process technology (failure of Dennard Scaling [5]), many-core chip faces power budget problem. Therefore, a challenging research problem is to design energy-efficient and high-performance NoC. 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 concentrated mesh 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, where every core can be heterogeneous with different computational capacities. Each processor 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.

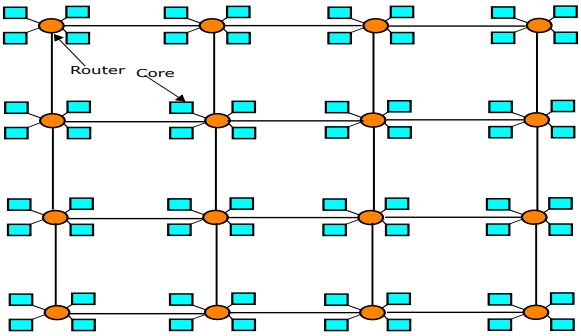

Fig. 1: 64-core architecture connected through 4×4 concentrated 2D-Mesh NoC.

Because of the diverse traffic patterns of application(s), different routers and links in NoC will carry different amounts of data by merging traffic flows from various applications. To cope with the heterogeneity of traffic workloads, NoC can be configured heterogeneously for energy efficiency. Prior research has shown that dynamic voltage and frequency scaling (DVFS) can reduce dynamic energy consumption of NoC routers and links at run-time [17]. 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 (voltage/frequency-levels (V/F-levels), heterogeneity of cores and routers, multicores, etc.), reactive solutions using heuristic or linear optimizer is not effective at run-time systems due to high time complexity for exploring many designs for optimal solution. Therefore, static or ad-hoc resource configuration may not be an effective technique to reduce hotspots. ML can help by predicting NoC resource requirements 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 explore the use of 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), state transition, and evaluate the reward of the NoC configuration action. RL agent reinforces the positive or negative rewards of the taken decision, which helps the system to learn and take proper actions and achieves the optimization objective. In this work, we also take advantage of neural networks (NNs) to learn the patterns in application demand, and approximate NoC configuration decision accordingly.

The major contributions of this work are outlined below:

- *Dynamic NoC Configuration using Deep Reinforcement Learning:* Voltage-frequency levels of the NoC routers and links are dynamically configured at run-time based on the application demand using RL and NNs.
- *Evaluation on Real Benchmarks:* Our proposed approach is evaluated using both large-scale (256-core) and small-scale (16-core) architectures on a real system simulator. Simulation results under real (COSMIC and E3S) benchmarks show that the proposed approach improves the energy-delay product significantly (40%) compared to non-ML based solution.

II. Related Work

ML techniques, such as regressions and NNs, are being explored for optimization in on-chip systems. [13] proposes to use 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 that are holistically optimized across the entire design stack. [25] 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. [14] proposed an imitation learning (IL) based methodology for dynamic voltage-frequency island (cluster of nodes/links) control in manycore systems. They showed that IL is able to obtain higher quality policies than RL with significantly less computation time and hardware area overheads. [11] presented an NN-based intelligent hotspot prediction mechanism that was used with a congestion-control mechanism to handle hotspot formations efficiently. Here the NN uses online statistical data to dynamically monitor the NoC interconnect fabric, but reactively predicts the location of hotspot(s). [20] 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 many-core NoC. [15] introduced a multi-layer NN based predictive routing algorithm for on-chip networks which uses network state and congestion information to estimate routing costs and perform low-latency routing of traffic. [19] developed a systematic ML framework that uses the kernel-based vector regression method to predict the channel average waiting time and the traffic flow latency for NoC performance improvement. In [10], the authors proposed a framework for modeling on-chip router power, performance, and area using ML-based regression methods. The authors showed that non-parametric regression techniques can capture the impact of both underlying architectural and implementation parameters on on-chip router power, performance and area. [7] proposed a RL based I/O management for energy-efficient communication between many-core processor and memory, instead of transmitting data under a fixed large voltage-swing. However, unlike previous works that focus mostly on small-scale NoCs, this work focuses to configure large-scale NoC 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 mapped to multicores. Besides performance improvement, the main advantage of RL is its self-adaptive nature to configure NoC at an instant to meet the real time requirements of application, where supervised ML technique need complete retraining to adapt to changes in systems/applications and traditional non-ML based algorithm fails to provide a solution within a reasonable time (because of high time complexity).

III. Dynamic NoC Configuration using Deep Reinforcement Learning

In this work, when applications are running in the NoC-based multicore systems, RL is used to configure the NoC voltage-levels dynamically based on the communication demands of the tasks. 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. However, we chose enough voltage-levels for energy savings. With each voltage-level, a frequency is also selected to form a voltage-frequency pair: 0.8 V/1 GHz, 0.9 V/1.5 GHz, 1.0 V/2 GHz, 1.1V/2.5 GHz.

RL [23] is very much suitable for intelligent decisions in autonomic and dynamic systems because of following reasons: (I) firstly, autonomic system can learn (using RL) what actions to take to maximize the long-term rewards from a specific state (Ii) secondly, RL can 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 properties 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. Therefore, RL is adapted in this work to dynamically configure voltage-level depending on the application demands to maximize the performance 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. Q-learning is used as a RL technique for calculating voltage-level configuration action value. As discussed in the Section I, NN is used to approximate the state-action mapping in the

RL agent. NN is 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. RL agent takes the V/F configuration decision with the maximum Q-value that maximize the latency and throughput of NoC.

A. Components of Deep RL model for NoC Configuration

The components of the proposed deep RL model for NoC configuration are described below, and shown in Figure 2.

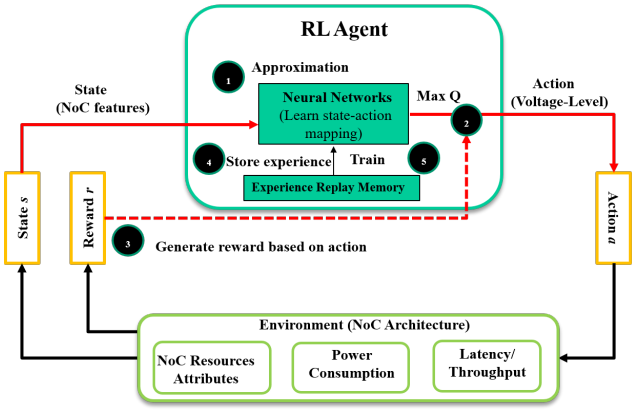

Fig. 2: Deep RL model components and flow for NoC configuration.

1) Environment (Architecture): The environment of the NoC configuration framework is consisting of routers, links, and processing cores. The environment generates the reward in terms of NoC latency and throughput for the taken NoC voltage-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. The 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. Furthermore, we collect energy consumption and performance data to evaluate the energy-efficiency and high-performance of the proposed technique.

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. The agent generates vector of Q-values for each V/F-level configuration action: 0.8 V/1 GHz, 0.9 V/1.5 GHz, 1.0 V/2 GHz, 1.1 V/2.5 GHz. Figure 3 shows the Q-values for four different V/F configuration based on the router and link features at a

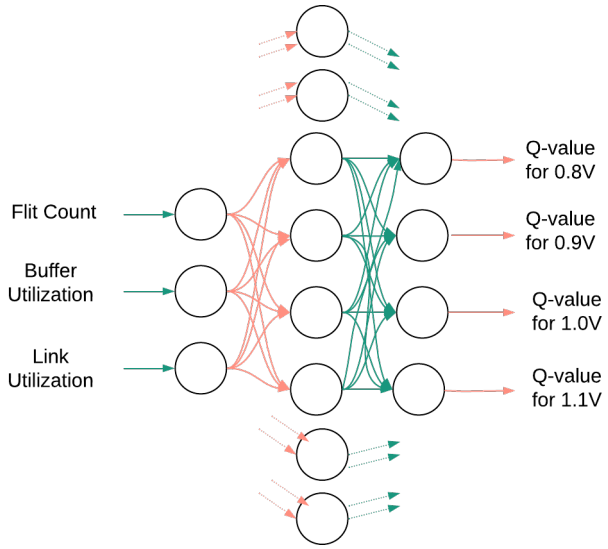

Fig. 3: NoC features and actions for voltage-level configuration using neural networks.

router port. NNs approximate the Q-values for different V/F configurations, and RL agent selects the V/F configuration with the maximum Q-value, which is the best action to maximize NoC performance.

4) Reward Function: 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 the Q-learning algorithm is designed to maximize the long-term reward. In this run-time configuration context, reward function is used to maximize the performance: reward = latency/throughput. The product of throughput and inverse of latency is used in reward function to maximize both latency and throughput performance of NoC.

5) Neural Networks for Function Approximation: NNs are used for action-function approximation in RL to solve the large state-action mapping table problem for large-scale systems. An NN provides a mechanism for generalizing training experience across states, so that it is no longer necessary to keep the state-action mapping table for RL agent's decisions. An NN approximates the vector of Q-values scores for each possible V/F-level configuration actions, and the RL agent selects the action with the maximum score for maximizing the NoC performance. NNs with three layers (input-hidden-output layers) are trained to approximate the Q-values for actions based on the current state (features), past learning experience, and target optimization objective (as reflected in reward function) for NoC configuration.

6) RL Stability Components: RL suffers instability when the Q-value function is approximated with a non-linear supervised learning, like NNs [18]. Similarity of subsequent training samples can lead the 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. 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. The system then randomly selects the experience data from the table to train the NNs. In the table, the oldest data is overwritten by a new data.

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. 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 networks on target networks parameters, which results in the reduction of divergence in approximation (using NNs).

B. Distributed RL for Large-Scale NoC

In the large-scale NoC, it is not feasible to implement centralized RL 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) in the NoC. Decision delay increases as every node has to communicate with the centralized 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. Furthermore, centralized decision maker can become a bottleneck (e.g., hotspot) in the many-core NoC as many communications are going through that one controller node. Distributed RL technique is needed in large-scale 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 concentrated-Mesh NoC is presented in Fig. 4. In 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.

C. Hardware Overhead for ML

Each new feature for deep RL increases the arithmetic overhead. That's why the number of features for deep RL are kept to a small number of values to reduce the hardware overhead. The proposed deep RL approach uses NNs with three layers: input layer with three neurons (for three features), hidden layer with 8 neurons and an output layer with 4 nodes. The layers of the NNs are computed in sequential order, for example, first layer must gather features and compute

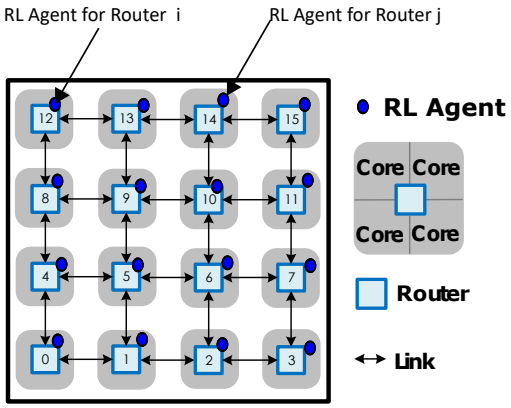

Fig. 4: Distributed RL-Agents based Optimization in a 4×4 NoC based 64-Core System.

its values before the next layer can be computed. Because of the parallel nature of operations, the computation time reduces significantly in NNs. For input layer to hidden layer connections, we have a total of 24 multiplies, 16 additions, and 8 comparisons. For hidden layer to output layer, we 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 [8]. The energy cost of a multiply is estimated to be 1.1 pJ and the area cost is 1640 um^2 in $45nm$ [8]. 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: 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. For the proposed distributed RL algorithm, this work needs an RL agent per router to decide the V/F action values based on the features. Though the energy and area cost of an RL agent is not significant enough, for a very large scale NoC, such as 25×25 NoC, the overhead of many RL agents can be high. To address that, we propose concentrated mesh NoC, where a single RL agent supports multiple cores in the NoC. This approach makes the overhead of ML feasible for very large-scale NoC based systems.

IV. Simulation and Results

Real system experiments are carried out using gem5 [3] and Garnet [2] platforms to evaluate the NoC performance in terms of end-to-end latency and network throughput. Four different voltage-levels are used for NoC configurations: 0.8V, 0.9V, 1.0V, and 1.1V. DSENT [22] tool is integrated with gem5 to evaluate the power consumption. gem5 configurations are shown in Table. I. 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. $ReLU$ function is used in the output layer as Q-values can be greater than 1 (for the corresponding V/F-levels). We use three features at the router for V/F-level configuration prediction: flit count, % of buffer utilization, and % of link utilization. a reward function, product of throughput and inverse latency, to evaluate the V/F-level configuration action in NoC.

TABLE I: gem5 configurations for 256-core CMesh NoC.

Instruction Set Architecture (ISA)	ALPHA
CPU Frequency	2GHz
Interconnect Frequency	1GHz
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 COSMIC benchmark suite [24] and embedded system synthesis benchmarks suite (E3S) [6]. For large-scale problems, we have simulated 256-core connected through 8×8 concentrated mesh (CMesh) architecture and simulated large applications in COSMIC. COSMIC benchmark suite is used here because it is based on real applications with a large number of tasks per application. Five applications from COSMIC are mapped in our simulations: face recognition, cifar, ultrasound imaging, reed-solomon code decoder (RS-Decoder), and reed-solomon code encoder (RS-Encoder). For small-scale problems, we have simulated 16-core connected through 4×4 2D-Mesh NoC, and simulated applications in E3S. E3S 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. We also simulate complex multi-application domain systems by mixing multiple applications in COSMIC and E3S, e.g., FaceCfr for face recognition and cifar applications and CN for consumer and networking applications.

We compared our solution with an non-ML based load-balanced mapping solution in [21]. We have evaluated our mapping and configuration framework using throughput and energy-delay product (EDP). The EDP is used to compare the average energy spent with the average time packet spends within the network.

Figure 5 shows the energy-delay product for various COSMIC benchmarks for all techniques. Simulation results for COSMIC benchmarks suggest that energy-delay product improvement is significant in the proposed dynamic NoC configurable solution. The proposed approach improves EDP by up to 45% (20% on average) compared to non-ML based configuration solution. In our proposed approach, we configure the NoC V/F-levels after application mapping which tailors NoC when the application has mapped. In other approaches, the NoC V/F-levels are static and there are no significant changes once mapping is complete. EDP decreases due to both lower energy consumption and lower delay (latency) in the NoC because of the proactive V/F-levels configuration using RL. RL assigns the required V/F-levels proactively (depending on the historical data) to reduce the energy consumption. As RL agents learn the policy to maximize the reward (product of throughput and inverse of latency), communication latency in NoC decreases because of the lower queuing latency at NoC routers. Because of the reduction of latency, throughput in the proposed solution improves by 15% (on average) compared to non-ML solution, as shown in Figure 6.

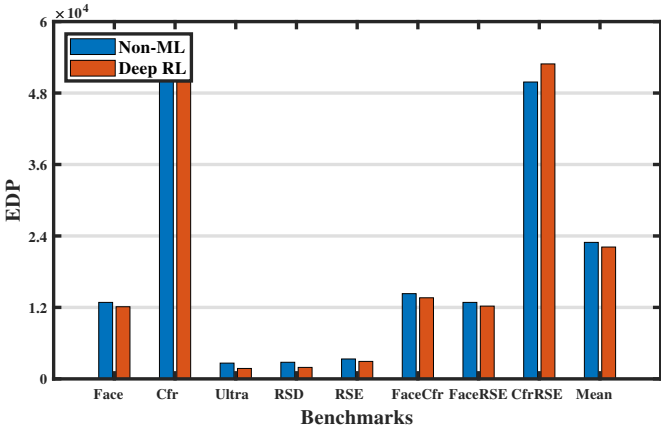

Fig. 5: EDP comparison under COSMIC Benchmarks in 256-core NoC.

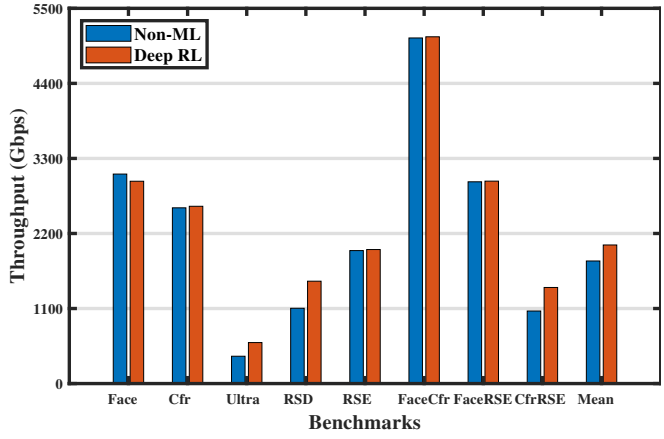

Fig. 6: Throughput comparison under COSMIC Benchmarks in 256-core NoC.

Simulation results for E3S benchmarks suggest that latency improvement is significant (throughput also improves) in the

proposed solution, while minimizing the hotspots in the system. As seen in Figure 7, the proposed solution improves EDP by up to 110% (30-50% on average) compared to non-ML solution. EDP improvement is significantly higher mainly because of the minimization of energy consumption. Energy is minimized because of the dynamic assignment of required V/F-levels using online ML solution. As RL agents learn the policy to maximize the reward (throughput/latency), throughput in the proposed solution improves by 10% on average compared to non-ML solution, as shown in Figure 8.

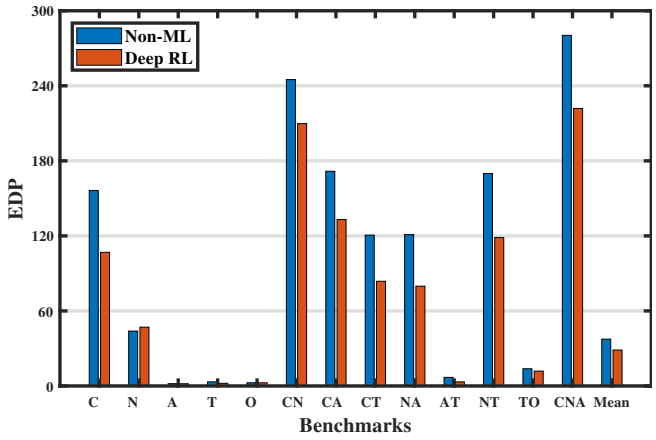

Fig. 7: EDP comparison under E3S Benchmarks in 16-core NoC.

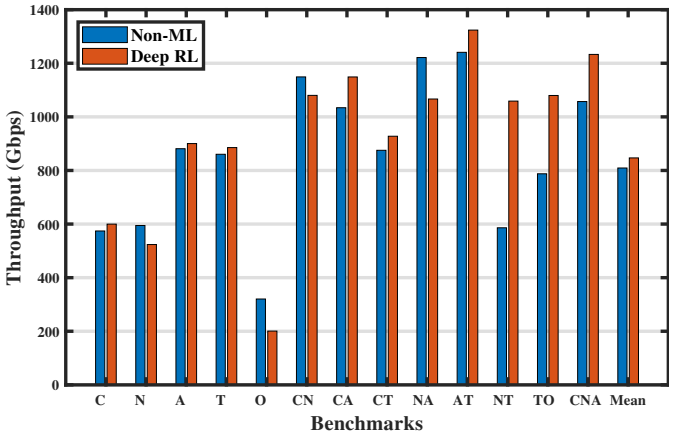

Fig. 8: Throughput comparison under E3S Benchmarks in 16-core NoC.

V. Conclusions

We have proposed dynamic NoC configuration on many-core architecture using machine learning techniques. NoC is configured proactively based on the historical data using deep reinforcement learning, where reinforcement learning agent takes the voltage/frequency-level configuration action using neural network function approximator. The use of neural networks in reinforcement learning and distributed per-router reinforcement learning agent in NoC make the proposed approach feasible for large-scale systems. Simulation results demonstrate that the proposed dynamic configurable solution improves NoC performance significantly in terms of latency, throughput and energy consumption while consuming very low energy and area overhead.

References

[1] Cerebras Wafer Scale Engine, https://www.cerebras.net/product/.

[2] 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. ISPASS*, pages 33–42, April 2009.

[3] N. Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[4] B. Bohnenstiehl et al. Kilocore: A 32-nm 1000-processor computational array. *IEEE Journal of Solid-State Circuits*, 52(4):891–902, April 2017.

[5] R. H. Dennard et al. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct. 1974.

[6] R. Dick. Embedded system synthesis benchmarks suites (e3s), http://robertdick.org/tools.html.

[7] H. Hantao, P. D. S. Manoj, D. Xu, H. Yu, and Z. Hao. Reinforcement learning based self-adaptive voltage-swing adjustment of 2.5d i/os for many-core microprocessor and memory communication. In *Proc. ICCAD*, pages 224–229, Nov 2014.

[8] M. Horowitz. 1.1 computing's energy problem (and what we can do about it). In *Proc. ISSCC*, pages 10–14, Feb 2014.

[9] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61, Sept. 2007.

[10] K. Jeong, A. B. Kahng, B. Lin, and K. Samadi. Accurate machine-learning-based on-chip router modeling. *IEEE Embedded Systems Letters*, 2(3):62–66, Sept 2010.

[11] E. Kakoulli, V. Soteriou, and T. Theocharides. Intelligent hotspot prediction for network-on-chip-based multicore systems. *IEEE Trans. TCAD*, 31(3):418–431, March 2012.

[12] M. Kas. Toward on-chip datacenters: A perspective on general trends and on-chip particulars. *J. Supercomput.*, 62(1):214–226, Oct. 2012.

[13] R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu. Machine learning and manycore systems design: A serendipitous symbiosis. *Computer*, 51(7):66–77, July 2018.

[14] R. G. Kim et al. Imitation learning for dynamic vfi control in large-scale manycore systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(9):2458–2471, Sep. 2017.

[15] M. A. Kinsy, S. Khadka, and M. Isakov. Prenoc: Neural network based predictive routing for network-on-chip architectures. In *Proc. GLSVLSI*, pages 65–70, 2017.

[16] G. D. Micheli et al. Networks on chips: From research to products. In *Proc. DAC*, pages 300–305, June 2010.

[17] A. K. Mishra et al. A case for dynamic frequency tuning in on-chip networks. In *Proc. MICRO*, pages 292–303, Dec 2009.

[18] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

[19] Z. Qian et al. SVR-NOC: A performance analysis tool for network-on-chips using learning-based support vector regression model. In *Proc. DATE*, pages 354–357, March 2013.

[20] M. F. Reza, T. Le, B. Dey, M. Bayoumi, and D. Zhao. Neuro-NoC: Energy optimization in heterogeneous many-core noc using neural networks in dark silicon era. In *Proc. ISCAS*, 2018.

[21] 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. SOCC*, 2018.

[22] C. Sun et al. DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proc. NOCS*, pages 201–210, May 2012.

[23] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[24] Z. Wang et al. A case study on the communication and computation behaviors of real applications in noc-based mpsocs. In *Proc. ISVLSI*, pages 480–485, July 2014.

[25] J. Yin, Y. Eckert, S. Che, M. Oskin, and G. H. Loh. Toward more efficient noc arbitration: A deep reinforcement learning approach. In *Proc. AIDArc, held in conjunction with ISCA*, June 2018.