Toward More Efficient NoC Arbitration: A Deep Reinforcement Learning Approach

Jieming Yin* Yasuko Eckert* Shuai Che* Mark Oskin*† Gabriel H. Loh*
*Advanced Micro Devices, Inc. †University of Washington
{jieming.yin, yasuko.eckert, shuai.che, mark.oskin, gabriel.loh}@amd.com

***Abstract*—The network on-chip (NoC) is a critical resource shared by various on-chip components. An efficient NoC arbitration policy is crucial in providing global fairness and improving system performance. In this preliminary work, we demonstrate an idea of utilizing deep reinforcement learning to guide the design of more efficient NoC arbitration policies. We relate arbitration to a self-learning decision making process. Results show that the deep reinforcement learning approach can effectively reduce packet latency and has potential for identifying interesting features that could be utilized in more practical hardware designs.**

I. Introduction

Advancements in computational speed, memory capacity, and "Big Data" have enabled a resurgence in artificial intelligence (AI). Within the broad field of AI is a subfield called machine learning, in which computers are iteratively trained on sets of data using various algorithms, and then apply what has been "learned" to, for example, make predictions or to classify data with new inputs. Deep learning is a subfield within machine learning in which the algorithms in question attempt, loosely, to mimic the structure and behavior of biological neurons in order to recognize patterns in data. Automated image categorization and natural language processing are common applications of deep learning, but there are many other emerging and future use cases for deep learning, including gaming, robotics, medicine, finance, high-performance computing, and more. Deep learning has advanced in sophistication to such a degree that it is beginning to be applied to areas in which, traditionally, human expert knowledge and experience was considered vital to decision making. Deep learning can, in some situations, augment human expert intelligence in order to enable deeper insights and provide more effective decision making than humans could accomplish without the application of these techniques.

One such application of deep learning is in computer system designs. Recent work has used machine learning for identifying patterns to aid optimizing memory-controller policies [1] and hardware prefetchers [2], [3]. We believe that these are just the tip of the iceberg. We expect more designers to utilize deep learning to discover previously unexploited patterns for optimization, creating opportunities for unique innovations at a faster rate.

In this paper, we use the design of network on-chip (NoC) arbitration policies as a case study for examining the potential benefits of deep-learning-assisted microarchitecture designs. The NoC is a shared resource that arbitrates on-chip traffic from various components, such as cores, caches, and memory controllers. Unoptimized arbitration logic can lead to suboptimal performance, but designing an efficient policy is a challenging task because the policy must be effective for dynamically variable traffic patterns with different levels of message criticality while ensuring forward progress and fairness.

We relate NoC arbitration to a reinforcement learning problem that addresses decision making. Reinforcement learning is a machine learning method in which the agent learns control policies by interacting with a stochastic environment. To be specific, we augment the NoC arbiters with a deep Q-network that observes the router state and evaluates the long-term performance impact of arbitration decisions. During execution, the arbiter learns to optimize its arbitration policy to maximize long-term performance. Instead of building an inference neural network in hardware, this preliminary work aims to demonstrate the feasibility and effectiveness of applying deep reinforcement learning to assist in microarchitecture design. Results show that the proposed deep reinforcement learning model is able to significantly reduce packet latency under synthetic traffic. Moreover, we believe the proposed approach has potential for identifying useful behaviors and exploring new features that researchers can exploit in practical, implementable NoC arbitration policies.

II. Background and Problem Formulation

A. NoC Arbitration

An NoC consists of routers that are connected with links. As numerous on-chip components (endpoints) compete for the same network resources, arbitration becomes a critical router functionality for controlling and hopefully reducing network congestion. The arbiter grants an output port to one of multiple input ports for sending a packet. Traditional round-robin arbitration provides a high degree of fairness by treating each input port fairly and guaranteeing fairness in scheduling; however, it considers only local fairness for each router, and therefore provides insufficient equality of service (i.e., link bandwidth allocation becomes more unfair the longer the routes are). Approximated age-based packet arbitration [4] provides better equality of service but has limitations regarding fairness of bandwidth allocation. Fair Queueing [5] and Virtual Clock [6] achieve fairness and high network utilization by maintaining per-flow state and queues, but they are costly to implement in an NoC.

B. Reinforcement Learning

In reinforcement learning, the agent attempts to learns the optimal actions that lead to maximum long-term reward by interacting with the environment, and the environment returns a numerical reward to the agent for each action it takes. As one well-known branch of reinforcement learning, value-based learning (e.g., Q-learning) uses a Q-value to represent the quality of taking a particular action a when an agent is in state s [7]. In its simplest form, for each state s, there may be several possible actions to take. An agent can either choose an action a that has the highest (currently estimated) Q-value among all possible actions, or a random action to explore new trajectories. After taking the action, the agent transitions to a new state s' while in the meantime the environment provides a reward r. With the tuple $\langle s,a,r,s'\rangle$, the algorithm uses the following Bellman Equation as an iterative update, which is used to maximize the expected cumulative reward achievable from a given state-action pair:

$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$

where α is the learning rate and γ is a discounting factor. To perform well in the long-term, both the immediate reward and the future rewards need to be accounted for. γ determines how much weight is given to future rewards.

Traditional Q-learning uses a Q-table to store the Q-value for each state-action pair. However, for many real-world problems, the state-action space can be extremely large. For example, when using reinforcement learning to play Atari® games, a Q-value for each image frame and possible actions must be tracked, requiring an impractical amount of storage space [8]. One possible solution is to use a neural network to approximate the Q-function. Given a state s, the neural network can output a vector of approximated Q-values for each possible action. Then, the action with the highest Q-value is chosen. This technique is called deep Q-learning (DQL) [8]. In this paper, we adopt DQL and use a multi-layer perceptron neural network to approximate the Q-function.

C. NoC Arbitration as a Reinforcement Learning Problem

Reinforcement learning can be applied to the problem of NoC arbitration to learn an efficient arbitration policy for a given NoC topology. We can train the *agent* (a recommendation system for the arbiter) such that for a given *state* (a collection of input buffers at a router all competing for the same output port) in the *environment* (NoC), the agent recommends an *action* (one input buffer) that would maximize an accumulated *reward* (e.g., network throughput). In our preliminary exploration, all routers in the system share the same agent for faster training.

III. Design

In this section, we first provide an overview of the DQL-based NoC arbitration model. Then we describe the DQL model in detail, explain how arbitration decisions are made, and how to train the model for better prediction. While building a neural network in hardware is costly, our goal is to propose a framework that assists designers in discovering useful features.

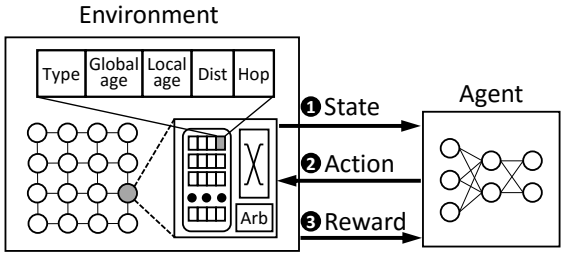


Fig. 1: Conceptual diagram of the proposed DRL-based NoC arbitration model.

A. DQL-based NoC Arbitration Model

Figure 1 shows a conceptual diagram of the proposed DQL-based NoC arbitration model. Each cycle, each router interacts with the agent by sending its own router state(s). The agent then evaluates the Q-values for all possible actions and returns the values to the router. Based on the Q-values, the router selects one input buffer and grants the output port. Meanwhile, a reward is sent to the agent for training and further evaluation.

Our proposed DQL model consists of the following major components:

1) Environment: The environment is the modeled NoC consisting of routers, links, as well as processing elements that generate and consume packets.

2) Agent: The agent takes the router state (see below) as an input and computes the Q-value for each action. In traditional table-based Q-learning, the number of table entries grows exponentially with the number of features used in state representation. It is impractical to maintain a table for the arbitration problem we are solving. As a result, we use a neural network to approximate the Q-values.

3) State: Each router has one state for each output port. Each state is represented with a feature vector for packets from all input buffers. Consider a router with n input ports and m output ports, and each input port has k input buffers (e.g., for multiple message classes, virtual channels). In each cycle, there could be multiple packets with different features from different input buffers competing for the same output port. Thus, we use m state vectors per router, with one vector per output port. A state vector consists of a list of features from all packets that compete for the same output port, as well as zero inputs for empty input buffers and irrelevant packets that request other output ports. To be more specific, assuming each packet has p features, the total length of the state vector is $n \times k \times p$. Because all routers share the same DQL agent, the input layer width of the neural network used by the agent must match the width of the largest router's state vector. In other words, all state vectors from different routers must have the same width, and smaller routers must zero pad their state vectors as needed. The agent considers one state vector at a time.

Figure 2 shows an example for a router with three input ports and two output ports, where each input port has two

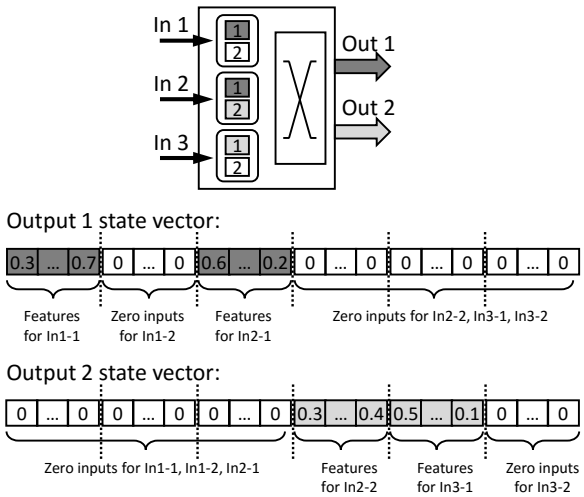


Fig. 2: Router state example.

buffers. For simplicity, this example assumes all packets are one-flit wide, and each packet has k features. Packets from the first buffer of Input 1 ($In1\_1$) and the first buffer of Input 2 ($In2\_1$) compete for Output 1, while the packets from the second buffer of Input 2 ($In2\_2$) and the first buffer of Input 3 ($In3\_1$) compete for Output 2. According to our definition of state, this router has two state vectors. The state vector for Output 1 starts with k features for $In1\_1$, followed by k zeros because $In1\_2$ does not have a packet. The next k elements are features for $In2\_1$, followed by $3k$ zeros for packets that request Output 2 and for empty buffers. Similarly, the state vector for Output 2 starts with $3k$ zeros, followed by features for $In2\_2$ and $In3\_1$, and zero inputs for $In3\_2$.

In this work, we consider the packet features listed below. Each feature is normalized such that all elements in the state vector are between 0 and 1.

- Type: an identifier specifying whether a packet is a request, response, or coherence probe message.
- Global age: number of network cycles spent from the point when the packet is generated.
- Local age: number of network cycles spent from the point when the packet arrived at the local router.
- Distance: number of total hops from the packet's source to destination.
- Hop: number of hops the packet has traversed so far.

4) Action: The agent in our model generates a vector of Q-values. Each element of the vector corresponds to an input buffer. Again, because all routers share the same agent, the neural network's output layer width must match the number of input buffers of the largest router. The Q-value vector is sent back to the router, which is used for making an arbitration decision (described in Section III-B).

5) Reward: After taking an action, the environment generates a reward 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 the context of NoC arbitration, several different metrics are possible for defining a reward. Examples include packet latency, network throughput, and fairness. In this work, we consider the following reward options:

- Reciprocal of average packet latency. For every fixed period of time (e.g., 100 cycles), we calculate the average latency L of all packets sent through the NoC. Because lower latencies indicate better performance, the reciprocal of average packet latency (i.e., $1/L$) is used as a reward. This value is used for all actions in the next period regardless of the actual actions taken.
- Fixed reward of 1 for each action. The intuition behind using a fixed reward is to maximize the total number of arbitration actions. The more arbitration is performed, the more packets are sent in the NoC, which leads to a higher overall network throughput.
- Link utilization in the previous cycle. The utilization is calculated as the number of active links in the previous cycle divided by the total number of network links. The goal is to maximize link utilization which in turn improves the network throughput.

B. Decision Making

Depending on input status and output port availability, a router queries the agent up to m times every cycle, where m is the number of output ports. If none of the input packets request an output port, or an output port is not available (e.g., in the middle of transmitting a multi-flit packet), the router does not issue a query for that particular output. If an output port is requested by only one input packet, the output port is directly granted to the input buffer without querying the agent.

For each issued query, the agent generates a vector of Q-values and sends it back to the router. After receiving the Q-value vector, the router grants the output port in question to the input buffer that has the largest Q-value. There are a few cases in which the router selects an input buffer with a lower Q-value. The largest Q-value element might correspond to an empty buffer, especially during the initial training phase, when there are other buffers with pending packets. Another case involves an input-port conflict. Because an input port can route at most one packet per cycle, no more than one output port should be granted to the same input port in the same cycle. In these cases, the router selects an input buffer with the next largest Q-value.

C. Model Training

Certain reinforcement learning algorithms present instability when the Q-value function is approximated with a nonlinear function such as a neural network. DQL stabilizes the training using experience replay [9] with a separate target network [8]. In this section, we explain how the parameters θ (weights and biases) of the neural network are updated, and how we stabilize the training.

Updating Neural Network Parameters. There are two neural networks in our model: a prediction network Q with parameters θ, and a target network $\hat{Q}$ with parameters $\hat{\theta}$. The prediction network is used to compute the Q-value vector that the router uses to make arbitration decisions. The target

network is used to compute the target Q-values that are used to update parameters. In each training step, a gradient descent on

$$\frac{1}{2}[r+\gamma max_{a'}\hat{Q}(s',a')-Q(s,a)]^2$$

is performed with regard to the prediction network's parameters θ. To stabilize the training, we set $\hat{\theta}=\theta$ for every fixed number of steps (100 steps in our model). In other words, the prediction network parameters are updated in each training step, while the target network parameters are updated periodically.

Experience Replay. Each time the agent interacts with the environment, an experience record $\langle s,a,r,s'\rangle$ is generated and stored in a replay memory. The replay memory is a circular buffer with a limited number of entries (200 in our model) and the oldest record is overwritten by a new record. When training the model, instead of using the most recent record, records are randomly sampled from the replay memory to form minibatches. Based on the current state s, the agent only provides the router with a vector of Q-values, and the router decides which input buffer to select. Therefore, the router needs to inform the agent of the next state s' after arbitration is performed. Experience replay breaks the similarity of subsequent training samples, which in turn reduces the likelihood of the neural network from being directed into local minima. In addition, experience replay allows the models to learn the past experience multiple times, leading to faster model convergence.

IV. Evaluation

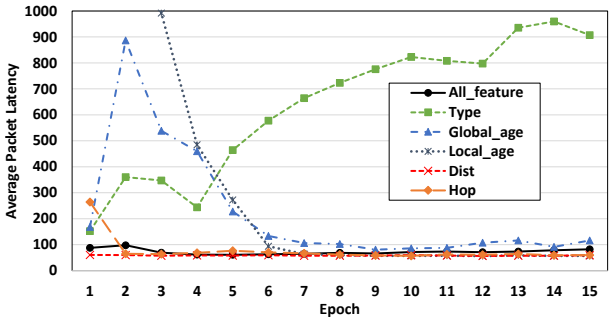
In this section, we describe our evaluation environment. Then we present the experimental results and our findings.

A. Experimental Methodology

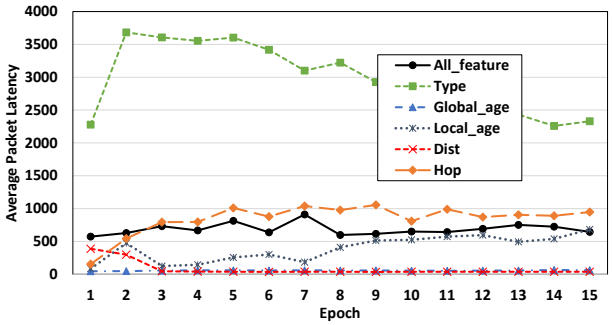
We use Garnet [10] in our evaluation. We primarily study a 4×4 mesh network with 2-stage routers. Each router is attached to a processing element that generates and consumes network packets. The default routing algorithm is XY routing, and the baseline arbitration policy is round-robin. Without loss of generality, we generate three types of messages: 1) request messages, which are 1 flit in length; 2) forward messages, which are also 1 flit in length; and 3) response messages, which are 5 flits in length (one header flit plus four more data flits for a cacheline). There are separate input buffers for different message classes, and each message class has one virtual channel (VC) with 4-flit buffers. We also evaluate and present results for a larger network (8×8 mesh) and varying numbers of virtual channels, but we primarily focus our discussion on the 4×4 mesh 1-VC configuration.

We evaluate the proposed DQL-based arbitration model with three traffic patterns: 1) Uniform Random: destinations are randomly selected with a uniform distribution; 2) Bit-complement: each node sends messages only to the node corresponding to the 1's complement of its own address; and 3) Transpose: node (x,y) sends messages only to (y,x).

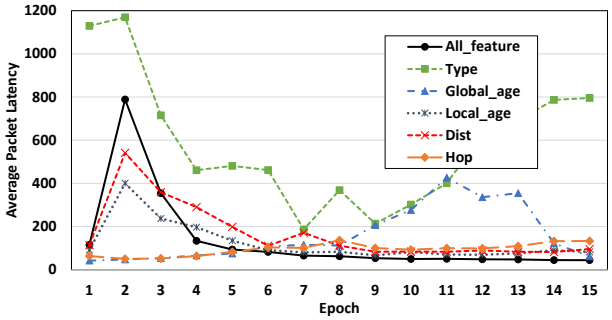
The DQL framework described in Section III is implemented and integrated into Garnet. The agent neural network consists of one input layer, one hidden layer, and one output layer. The activation functions we use for the hidden and the output layer are *Sigmoid* and *ReLU*, respectively. We use five features from each message (type, global age, local age, source-to-destination distance, and hop count), and three reward criteria (reciprocal of average latency, fixed reward, and link utilization). To add an additional comparison point, we provide an ideal implementation of age-based arbitration where each packet is timestamped at injection and the age continues to increase every cycle. Age-based arbitration is costly to implement in real hardware but provides global fairness [11].



(a) Uniform Random (0.23 packet/cycle/node).



(b) Bit-complement (0.17 packet/cycle/node).



(c) Transpose (0.15 packet/cycle/node).

Fig. 3: Training results comparison for agent neural networks with different input features across three different traffic patterns. Each figure shows only 15 epochs.

B. Experimental Results

1) Feature Studies: Figure 3 shows the training results for the proposed model with different input features. For each traffic pattern, we first sweep the network injection rate using the baseline RR arbitration routers to find the saturation point

at which the packet latency increases dramatically. We then train the DQL-based model using the same injection rate. The model is trained for 30 epochs, and each epoch is three million cycles long. The parameters of the neural network are randomly initialized for the first epoch. At the end of each epoch, the updated parameters are saved; in the next epoch, the saved parameters are used to initialize the neural network. In this figure, *All_feature* means that the router input states include all five features described in Section III, while the rest only include a single feature. We notice that all features except for *Type* converge after a few epochs of training. Features such as *Global_age* and *Dist* perform well across all traffic patterns while other features, namely *Local_age* and *Hop*, significantly reduce packet latency under Uniform Random and Transpose traffic but are less effective under Bit-complement traffic. In addition, the configuration with all features combined does not necessarily lead to the lowest latency under Uniform Random and Bit-complement traffic.

TABLE I: Average packet latency comparison. Feature(s) that lead to the best performance improvement are noted in the parenthesis.

	RR	Age	DQL
Uniform Random	4855.8	28.7	56.1 (Local_age, Dist)
Bit-complement	5198.6	24.7	36.9 (Dist)
Transpose	3600.8	19.8	41.8 (All_feature)

To demonstrate the effectiveness of the proposed model, Table I shows a comparison of packet latency across round-robin, age-based, and DQL-based arbitration policies. Again, all of the networks operate at the injection rate that saturates the RR-based network. Age-based arbitration prioritizes the packet with the oldest age, thereby providing global fairness and reducing the variance in packet transit time. It also reduces the average packet latency in the evaluated system. Our proposed DQL-based arbitration model is also effective in reducing packet latency. At the RR-based network's saturation point, 98.8%, 99.2%, 98.8% of latency reduction is achieved for uniform random, bit-complement, and transpose traffic, respectively. In terms of throughput, as shown in Figure 4, the DQL-based arbitration policy improves the network throughput over RR-based policy by 4.5%, 6.2%, and 7.1% under uniform random, bit-complement, and transpose traffic, respectively.

Although not all features lead to promising latency reductions, designers can use the proposed model to filter out uninteresting features and focus on potentially useful features. For example, *Dist* and *Hop* are shown to reduce latency, and they have been used to approximate the global age in prior work [4], [12]. Although we only evaluate a small number of features in this work, we believe our proposed model can potentially be used to explore a wide range of new features, enabling architects to design new arbitration policies and/or improving existing policies.

2) Study on Reward Criteria: Figure 5 shows a comparison among our three reward criteria: reciprocal of average packet latency, fixed reward for each action, and link utilization in the previous cycle. Overall, all three criteria lead to significant reduction in network latency compared to the RR policy. As training proceeds, *Avg_latency* and *Link_utilization* perform slightly better than *Fixed_reward*. Compared to the features, we found that the choice of reward is less critical as long as the criterion is associated with a performance metric that designers target for improvement.

TABLE II: Average packet latency reduction at saturation point after using the DQL model

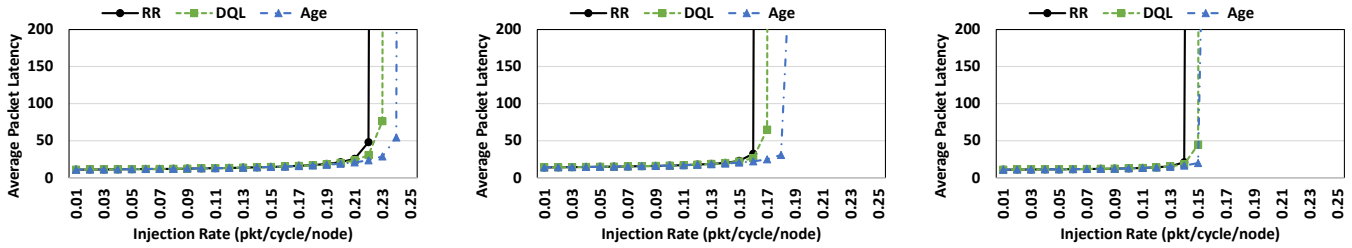
		RR	DQL	DQL (trained in 4x4 mesh)
4x4 mesh 2VC/class	Uniform Random	5087.4	19.2	-
	Bit-complement	1339.7	22.7	-
	Transpose	3497.6	402.5	-
8x8 mesh 1VC/class	Uniform Random	10397.7	3055.3	2765.0
	Bit-complement	3237.9	43.8	652.8
	Transpose	5640.9	1693.1	1118.5

3) Network Configurations: In this study, we investigate the proposed model with two different network configurations. The first system is a 4×4 mesh network, but we provide two VCs for each message class in contrast to only one in the previous study. The second system is an 8×8 mesh network, and each message class has one VC. Table II shows the latency results. For the last column, we apply the neural network that is trained in a 4x4 mesh 1-VC system to an 8x8 mesh 1-VC system and disable on-line training. The purpose is to study whether a neural network that is trained in one system can be applied to another system to reduce training overheads. Notice that the number of input buffers (neurons) for a 1-VC router differs from a 2-VC router, thus we cannot use the same neural network in the 2-VC system. Results show that our proposed system is capable of reducing packet latency compared to the RR policy in all cases. We also found that a pre-trained neural network is effective in a different system. As we stated, this is a preliminary work of utilizing machine learning to assist in NoC design. We will look into the interpretability of the neural network and broader feature exploration in future work. Overall, we believe that the proposed model is applicable to a wide range of systems and can be used to guide the design of NoC arbitration policies.

V. Related Work

In additional to the work outlined in Section II-A, iSLIP [13] is an RR-based policy that performs multiple iterations to find a conflict-free input-to-output mapping. Ping-pong arbitration [14] is another RR-based policy that divides the inputs into groups and applies arbitration recursively to provide fair sharing of switch bandwidth among inputs. Das *et al.* [15] propose a slack-aware arbitration policy that utilizes memory access criticality information for packet scheduling within the NoC.

Machine learning in microarchitecture is not new, and there is a large body of prior work that utilizes machine learning techniques to improve architectural designs. The perceptron



(a) Uniform Random (b) Bit-complement (c) Transpose

Fig. 4: Throughput comparison of round-robin (RR), age-based (Age), and DQL-based (DQL) arbitration policies.

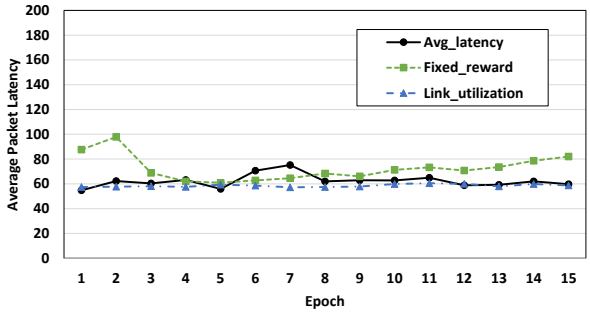


Fig. 5: Training results comparison for agent neural networks with different reward criteria under uniform random traffic.

branch predictor [16] uses a linear classifier that is trained online to predict whether a branch is taken or not. Ipek *et al.* [1] proposed a reinforcement learning-based memory controller that interacts with the system to optimize performance. Teran *et al.* [17] proposed perceptron learning for reuse prediction, which uses tags and program counters to learn correlations between past cache access patterns and future accesses. More recently, Zeng *et al.* [2] proposed a long short-term memory-based memory prefetcher that learns to capture regular memory access patterns. Hashemi *et al.* [3] relate contemporary prefetching strategies to n-gram models in natural language processing and propose a recurrent neural network-based prefetcher that handles more irregular benchmarks.

VI. Conclusion and Future Work

In this work, we demonstrated the effectiveness of applying deep Q-learning to the design of NoC arbitration policies. The proposed DQL-based arbitration model observes router states and evaluates the performance impact of selecting a candidate input buffer for arbitration. Through online learning, the model learns to make decisions that maximize the long-term NoC performance. We evaluated the proposed approach with various system configurations and synthetic traffic patterns, and the experimental results show that the DQL-based arbitration model is effective in reducing packet latency. While a full DQL algorithm is not practical to directly implement in a real NoC, we believe that the proposed approach can assist designers in exploring potentially useful features to design more efficient NoC arbitration policies.

Acknowledgment

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

References

[1] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, 2008.

[2] Y. Zeng and X. Guo, "Long short term memory based hardware prefetcher: A case study," in *Proceedings of the International Symposium on Memory Systems*.

[3] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," *eprint arXiv:1803.02329 [cs.LG]*, 2018.

[4] M. M. Lee, J. Kim, D. Abts, M. Marty, and J. W. Lee, "Probabilistic distance-based arbitration: Providing equality of service for many-core cmps," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010.

[5] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Symposium Proceedings on Communications Architectures &Amp; Protocols*, 1989.

[6] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proceedings of the ACM Symposium on Communications Architectures &Amp; Protocols*, ser. SIGCOMM '90.

[7] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature, 518(7540):529–533*, Feb 2015.

[9] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, May 1992.

[10] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 33–42.

[11] D. Abts and D. Weisser, "Age-based packet arbitration in large-radix k-ary n-cubes," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, ser. SC '07.

[12] M. Poremba, I. Akgun, J. Yin, O. Kayiran, Y. Xie, and G. H. Loh, "There and back again: Optimizing the interconnect in networks of memory cubes," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

[13] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, Apr.

[14] H. J. Chao, C. H. Lam, and X. Guo, "A fast arbitration scheme for terabit packet switches," in *Global Telecommunications Conference, 1999. GLOBECOM '99*, vol. 2, 1999, pp. 1236–1243 vol.2.

[15] R. Das, O. Mutlu, T. Moscibroda, and C. Das, "Aergia: Exploiting packet latency slack in on-chip networks," in *ISCA '10*.

[16] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001.

[17] E. Teran, Z. Wang, and D. A. Jiménez, "Perceptron learning for reuse prediction," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.