One Protocol to Rule Them All: Wireless Network-on-Chip using Deep Reinforcement Learning

Suraj Jog[†], Zikun Liu[†], Antonio Franques[†], Vimuth Fernando[†], Sergi Abadal[⋆], Josep Torrellas[†], Haitham Hassanieh[†]
University of Illinois at Urbana Champaign[†], Polytechnic University of Catalonia[⋆]

Abstract– Wireless Network-on-Chip (NoC) has emerged as a promising solution to scale chip multi-core processors to hundreds and thousands of cores. The broadcast nature of a wireless network allows it to significantly reduce the latency and overhead of many-to-many multicast and broadcast communication on NoC processors. Unfortunately, the traffic patterns on wireless NoCs tend to be very dynamic and can change drastically across different cores, different time intervals and different applications. New medium access protocols that can learn and adapt to the highly dynamic traffic in wireless NoCs are needed to ensure low latency and efficient network utilization.

Towards this goal, we present NeuMAC, a unified approach that combines networking, architecture and deep learning to generate highly adaptive medium access protocols for wireless NoC architectures. NeuMAC leverages a deep reinforcement learning framework to create new policies that can learn the structure, correlations, and statistics of the traffic patterns and adapt quickly to optimize performance. Our results show that NeuMAC can quickly adapt to NoC traffic to provide significant gains in terms of latency, throughput, and overall execution time. In particular, for applications with highly dynamic traffic patterns, NeuMAC can speed up the execution time by $1.37\times - 3.74\times$ as compared to 6 baselines.

1 Introduction

Recently, there has been an increasing interest from both industry and academia to scale network-on-chip (NoC) multicore processors to hundreds and thousands of cores [11, 21, 25, 49]. To enable such massive networks on chip, computer architects have proposed to augment NoC multicore processors with wireless links for communication between the cores [7, 9, 54, 65, 91]. The broadcast nature of wireless networks enables the NoC to significantly reduce the number of packets that the cores need to communicate to each other as well as the latency of packet delivery [1, 38]. Both aspects play a central role in scaling the number of cores on an NoC multicore processor (See Background Section 3 for details) [1, 8, 38, 50, 56]. These benefits have motivated RF circuits designers to build and test wireless NoC transceivers and antennas that can deliver multi-Gbps links while imposing a modest overhead (0.4–5.6%) on the area and power consumption of a chip multiprocessor [31, 93, 99, 100].

While the use of wireless can significantly benefit NoCs, it brings on new challenges. In particular, the wireless medium is shared and can suffer from packet collisions. Designing efficient medium access protocols for wireless NoCs is, however, difficult. The traffic patterns in NoCs tend to change drastically across applications. Even during the execution of a single application the traffic pattern can change as fast as tens of microseconds [4, 38]. As a result static MAC protocols such as TDMA, FDMA and CSMA perform poorly [17, 33, 35, 61, 70, 71, 89]. Further, due to thread synchronization primitives likes barriers and locks in parallel programming, the wireless NoC exhibits complex hard-to-model dependencies between packet delivery on the network and execution time. As a result, even adaptive protocols that try to switch between TDMA and CSMA or optimize for long-term throughput [40, 65, 66], perform poorly in the context of wireless NoCs since they remain agnostic to these domain specific and intricate dependencies. Hence, the design of efficient medium access protocols has been identified as a key bottleneck for realizing the full potential of a wireless NoC multiprocessor [6, 12].

In this paper, we present NeuMAC, a unified approach that combines networking, architecture and deep learning to generate highly adaptive medium access protocols for a wireless network on chip architecture. NeuMAC leverages a reinforcement learning framework with deep neural networks to generate new MAC protocols that can learn traffic patterns and dynamically adapt the protocol to handle different applications running on the multi-core processor. Reinforcement Learning (RL) has proved to be a very powerful tool in AI for generating strategies and policies that can optimize for complex objectives [68, 81]. RL allows NeuMAC to make better decisions by learning from experience. In particular, many basic functions, like FFT, graph search, sorting, shortest path, etc., tend to repeatedly appear in many applications. Past work also shows that a number of unique periodic traffic patterns emerge in multiple different programs, and as the number of cores increases, the traffic patterns show increasingly predictable spatiotemporal correlations and dependencies [3, 4]. NeuMAC learns these statistics and correlations in the traffic patterns, to be able to both predict future traffic patterns based on traffic history and adapt its MAC protocol to best suit the predicted future traffic. Furthermore, RL enables NeuMAC to account for hard-to-model complex dependencies between execution time and delivery of packets. In particular, we carefully engineer the reward function in RL to optimize for execution time rather than to simply improve the latency and throughput of the network.

Indeed, RL has been leveraged for wireless MAC protocols in the context of heterogenous wireless networks [43, 101], sensor networks [41], and IoT networks [62]. However, bringing these benefits to wireless networks on chip faces a num-

ber of unique challenges. First, past work runs RL inference for every packet at each time step, which is not feasible for WNoCs since the time scale of operation in a multicore processor is in the order of nanoseconds. Hence, per time-slot inference would significantly delay every packet transmission. Second, due to compute resource constraints, it is also not feasible to run RL inference at every core of the wireless NoC. While the second challenge can be addressed using a centralized controller for the RL model, it would still incur significant communication overhead and latency to collect the states from the nodes (e.g. traffic injections or buffer occupancy) and to inform the nodes when to transmit.

NeuMAC addresses these challenges by designing a framework where the controller is trained to generate high-level MAC policies simply by listening to on-going transmissions on the wireless medium. This allows NeuMAC to eliminate any communication from the cores to the controllers. Moreover, to amortize the overhead of inference and policy updates, NeuMAC only updates the cores with a new MAC policy once every interval spanning many execution cycles (e.g. ten thousand cycles). We also train NeuMAC to learn policies that are highly adaptive and simple to update, to reduce communication overhead from the controller to cores.

Finally, NeuMAC also needs to operate within the strict timing and resource constraints of the multicore processor. Modern deep neural networks, however, are designed with up to a billion tunable parameters and operate on high dimensional input spaces [47, 80]. Consequently, they require large amounts of memory and computational resources, and also suffer high inference latencies (tens of milliseconds) [46, 63]. To address this, we design NeuMAC's RL framework such that the input and output of the neural network scale linearly with the number of cores. This ensures that NeuMAC is expressive enough to service the highly dynamic network traffic while at the same time operate under the limited memory and computational resources. Specifically, NeuMAC's neural network requires three orders of magnitude less parameters, and adds a small area overhead to the multicore processor. It also has an inference latency that is small enough to meet the strict timing constraints of the multicore during run-time as we show in detail in Appendix A.

We evaluate NeuMAC by integrating it with a cycle-level architectural simulator for CPU-GPU heterogeneous computing that faithfully models the intricacies of multi-core processors [87]. We augmented the simulator with an on-chip wireless network that accurately models transmissions, collision handling and packet losses. We test NeuMAC's performance on real applications chosen from diverse domains such as graph analytics, vision and numerical simulations. We compare NeuMAC against six baselines including wired NoC, standard CSMA, TDMA, optimal CSMA protocols [79], adaptive protocols [38, 65], and an optimal oracle. Our evaluation reveals the following:

- For a 64-core NoC, NeuMAC is capable of learning traffic patterns and adapting the medium access protocol at a granularity of $10\mu s$ to achieve a median gain of $2.56\times - 9.18\times$ in packet latency and $1.3\times - 17.3\times$ in network throughput over different wireless NoC baselines.
- NeuMAC's throughput and latency gains translate into an average of $10\% - 47\%$ speedup in execution time over wireless NoC baselines which goes up to $1.37\times - 3.74\times$ for certain applications. The results also show a $3.4\times$ speedup on average over a purely wired NoC.
- NeuMAC's gains in execution time are close to the upper bound that can be achieved by a wireless network with infinite capacity and zero latency.
- As the number of cores scale up to 1024 cores, NeuMAC's performance gain increases to 3 orders of magnitude lower latency and up to $64\times$ higher throughput over baseline protocols.
- NeuMAC is robust to lossy channels, and sees minimal degradation in performance with upto 10% packet losses. We also test NeuMAC's sensitivity to noise in the observed state and show almost no loss in performance.

Contributions: We make the following contributions:

- We introduce the first MAC protocol that can learn and adapt to the highly dynamic traffic at very fine granularity in a wireless NoC processor. The protocol also accounts for non-trivial dependencies between packet delivery and computation speedups by optimizing for execution time.
- We design a lightweight deep reinforcement learning framework that introduces little overhead to the multi-core processor and can operate within tight timing, power and area constraints of chip multicore processors.
- We extensively evaluate our design and demonstrate significant improvement in network performance and reduction in the overall execution time on the multicore processor.

2 Motivation and Insights

The wireless traffic patterns on a multicore processor have been shown to vary significantly across different applications. Even for a single application, the traffic can vary across different cores (spatially) and across different time intervals (temporally) [4, 6, 12, 38, 83].

Fig. 1(a) shows examples of traffic traces captured from a cycle-level architectural simulator for three different common benchmark applications on a 16-core multiprocessor. The x-axis shows the time in clock cycles, the y-axis shows the core ID, and the scatter points show the injection of traffic at each core. For clarity, we only show a portion of the execution spanning ten thousand cycles. Some applications, like *PageRank* shown in Fig. 1(a)(i), have almost constant traffic on all cores and can benefit from a contention-free protocol like TDMA. Other applications, like computing the *Shortest Path in a Graph* shown in Fig. 1(a)(ii), have very bursty traffic and can benefit from a contention-based protocol like

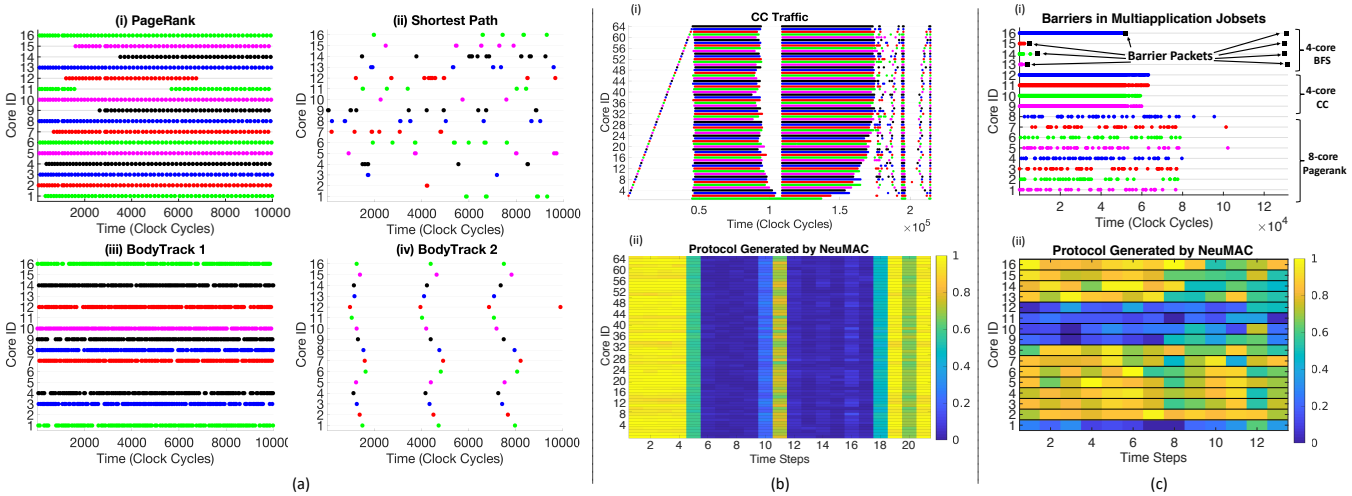

Figure 1: **Illustrative Examples: (a)** Traffic Pattern on a 16-core multiprocessor for different applications. The X-axis shows clock cycles, and the Y-axis corresponds to each of the 16 cores. The figures depict the scatter plots representing the packet injections into the buffer of each core. The different colors for packet injections are used for different cores. **(b)** NeuMAC can quickly adapt to fast changing traffic thus ensuring efficient network utilization throughout the application's execution. In the generated protocol, high probability values (closer to yellow in colormap) represent a CSMA-like protocol whereas low probability values (closer to blue) represent a TDMA-like protocol. **(c)** NeuMAC can learn and optimize for the intricate dependencies between the executions on different cores, and in turn optimize directly for end-to-end execution.

CSMA. Moreover, in most applications, the traffic pattern changes within the execution of the application. For example, Fig. 1(a)(iii)-(iv) show the traffic patterns at different times in the execution of *BodyTrack*, a computer vision application for tracking body pose. In the first time interval, since there is steady injection of packets into the network on the 10 active cores, a contention-free scheme will be optimal to minimize collisions, whereas in the second time interval, a CSMA-like based scheme for all 16 cores will perform better due to the sparse traffic injection. Next, we present concrete examples showcasing the range of protocols that NeuMAC can generate for different traffic patterns.

A. Adapting to Dynamic Traffic Patterns: To further appreciate the spatial and temporal changes across the execution of an entire application, we show the traffic trace for the application CC (Connected Components of a graph), running on a 64-core processor in Fig. 1(b)(i). Here we can see that the traffic varies significantly across the application's execution.

Fig. 1(b)(ii) presents the protocol generated by NeuMAC. At a very high level, NeuMAC's protocol is simple. Each core gets its own dedicated time slot where it can transmit with probability 1 if it has traffic. Additionally, core i can also transmit in time slots assigned to the other cores with some contention probability p_i. By setting these probability values p_i for each core, NeuMAC dictates the MAC protocol on the wireless NoC. The figure shows these contention probabilities p_i's for each core generated by NeuMAC. We present NeuMAC's protocol design in more detail in Section 4.3.

From Fig. 1(b)(ii), we can see that NeuMAC is able to adapt quickly to the changes in the traffic patterns, becoming more TDMA-like when the traffic is dense (contention probabilities p_i's are 0 and everyone transmits only in their assigned slot), and becoming more CSMA-like with sparse traffic (contention probabilities p_i's are high and cores can start transmitting in other's assigned time slots). In the case of CC, we can see that initially the traffic pattern is extremely sparse and structured such that a simple "Aloha" protocol would suffice. As a result, in the beginning the cores contend for the channel aggressively under NeuMAC's protocol. However, once the traffic pattern becomes more dense, NeuMAC adapts the protocol to be more TDMA-like, thus ensuring high network utilization. Finally, once the traffic pattern becomes less dense after $18 * 10^4$ cycles, the cores again start to contend for the channel with higher probability, thus emulating a CSMA-like protocol. Note that, while NeuMAC is able to quickly detect traffic changes from dense to sparse at time steps 11 and 18 (From Fig. 1(b)(ii)), it does not immediately increase contention probabilities for the cores. Instead the change is gradual, and this is because of the outstanding packets remaining in the buffers immediately after the phase with dense traffic injection. As a result, immediately switching the probabilities would lead to large number of collisions.

The above example demonstrates that NeuMAC is able to learn fine-grained highly dynamic MAC protocols that can quickly adapt to support different kinds of traffic patterns, while accounting for subtle characteristics of network operations such as buffer build-ups even though this information is not explicitly fed into NeuMAC's RL model. While there has been a lot of work on adaptive and optimal CSMA protocols [51, 73, 77, 102], these works are theoretical and make unrealistic assumptions. In particular, they optimize for long term throughput and assume that the protocol can reach a steady-state operation much faster than the variation in traffic patterns, which does not hold for wireless NoCs. As a result, these protocols perform poorly as we show in section 6.

B. Optimizing for Synchronization Primitives: Another challenge in designing efficient protocols stems from synchronization primitives. These primitives impose intricate dependencies between the execution of threads on different cores, leading to a non-trivial relationship between the delivery time of packets on the NoC and the progress of execution on each core. For example, in parallel computing it is common practice for software developers to use `barriers` for synchronization. These barriers are placed throughout the code of a multithreaded application in order to force each thread to stop at a certain point, blocking its execution until all participating threads catch up. Most standard libraries for parallel programming use barriers in many of its primitive routines in order to ensure the correctness of the program, such as OpenMP's `For` loop [23], or MPI's `Send/Recv` [45]. Therefore, there is complex but predictable structure in the traffic patterns caused by these synchronization primitives that can be exploited to improve parallel speedup and scalability of high performance applications. Hand tuning protocols to account for these dependencies is non-trivial. For example, the cores themselves do not explicitly know that they are involved in a barrier before they actually reach the barrier and execution halts. [28, 82]. Past work on designing MAC protocols mainly optimizes for throughput and latency, and is agnostic to such dependencies.

As a concrete example, consider the multiapplication jobset comprising of three concurrent applications, namely a 4-core BFS, a 4-core CC and a 8-core Pagerank, running on a 16-core multiprocessor as shown in Fig. 1(c)(i). In the traffic trace, one can observe two sets of barrier packets in the execution of BFS, denoted by black squares. The other two applications have no barriers in this portion of their executions. Here, note that core 16 has significantly more packets to transmit before arriving at its barrier, whereas core 13, 14 and 15 arrive at their barriers sooner. As a result, the execution on cores 13, 14 and 15 is blocked until core 16 clears its barrier, thus rendering the compute resources of these three cores useless as they idly wait for core 16. Additionally, at the same time core 16 also has to contend for the channel with traffic from CC, which itself has a lot of ongoing communication. Ideally, the MAC protocol in this case should prioritize traffic of the core that is falling behind, so that it arrives to the barrier and clears it as soon as possible, allowing the blocked cores to proceed execution and thus optimizing overall execution time. In Fig. 1(c)(ii), we can see that NeuMAC can learn to account and optimize for such dependencies. At the start, NeuMAC assigns high contention probabilities to cores 13 to 16 so that it can clear the barrier point at the earliest, while assigning low contention probabilities to cores 9 to 12. Once the barrier is cleared, NeuMAC increases the contention probabilities for the CC cores, so that it can transmit on the channel while the other applications go through low communication periods, thereby ensuring high network utilization.

Protocols like CSMA, TDMA and even adaptive protocols cannot optimize for such situations as they would treat every packet in the network as equally important, thus sharing the channel equally between BFS and CC here. This would result in core 16 clearing its barrier much later, thus harming end-to-end execution time. However, since NeuMAC is trained to directly optimize the high-level objective of end-to-end execution time instead of network metrics like latency, it is able to learn to prioritize the packets of some cores over others. In this example, with NeuMAC's protocol, core 16 arrives at its barrier 2.4× faster as compared to CSMA, and 3.75× faster as compared to TDMA. This in turn leads to an overall improvement in execution time of 43% and 81% over CSMA and TDMA respectively.

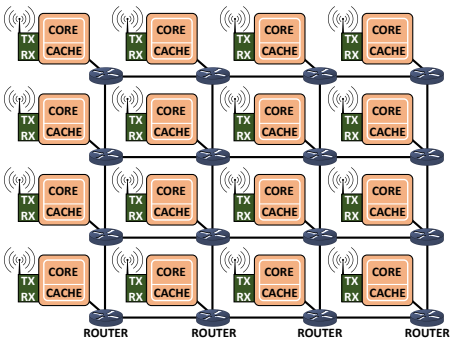

Figure 2: NoC Architecture with Wireless Links

3 Background

3.1 Wireless Network on Chip

Network-on-Chip (NoC) architectures have played a fundamental role in scaling the number of processing cores on a single chip which led to unprecedented parallelism and speedups in execution time [30, 75, 88, 94]. Prior to NoC, multicore processors used a shared bus architecture which had very poor scalability. As the core count increases, the power required to drive the bus grows quickly due to the increase in the capacitance of the bus wires [15]. The bus also starts to suffer from large latency [74]. As a result, shared buses become impractical for designs beyond 16 cores [59].

Unlike a shared bus, wired NoCs use packet-switched communication with every core connected to a router as shown in Fig. 2 [78]. As the packet moves from source to destination, it is buffered, decoded, processed, encoded, and retransmitted by each router along the multi-hop path. However, as we scale the number of cores, computation slows down due to the high communication latency and overhead of the network [10, 57, 97]. This problem is known as the "Coherency Wall" [58], where the execution on each core is faster than the NoC's ability to ensure that the memory caches of the cores are coherent. Hence, the speedup gained by parallelism and multithreading is outweighed by the network's communication cost for keeping the caches coherent [5, 8, 58].

Recent work proposes to augment NoC multicore pro-

cessors with wireless links for communication between the cores [7, 9, 54, 65, 91]. Wireless links benefit chip multicore processors in two important aspects:[1]

- **Lower Latency:** Wireless enables every core to reach every other core in just a single hop. In contrast, in a purely wired NoC, a packet must go through multiple NoC routers, incur queuing, transmission, and processing delay at every hop which ends up taking multiple execution cycles [1]. Hence, as the number of cores increase, wireless can deliver packets with significantly lower latency and within the tight timing requirements of execution on the cores [1].
- **Broadcast:** Since wireless is a broadcast medium, transmitted packets are directly heard at all other cores which significantly simplifies the NoC's ability to ensure the coherency of the memory caches. In particular, any local changes in the memory cache of a core can instantaneously be replicated at all other cores through a single packet transmission [38]. In contrast, today's wired NoCs must send multiple parallel unicast/multicast transmissions to synchronize the caches, which leads to a large overhead that scales poorly as the number of cores increases [8, 50, 56].

Several wireless NoC transceivers and antennas have been built and shown to deliver 10 to 50-Gbps links while imposing modest overhead (0.4–5.6%) on the area and power consumption of a chip multiprocessor [31, 39, 93, 99, 100]. The wireless transceivers typically operate in the millimeter-wave and sub-THz spectrum which enables miniaturizing the antennas and avoids antenna coupling. Antennas are either planar integrated dipoles or vertical monopoles drilled through the silicon die [24, 86]. The wireless signals propagate through the enclosed chip packaging and attenuate by few tens of dBs [85, 86]. On-Off Keying (OOK) is the choice of modulation since it requires significantly lower powerand achieves a very low Bit Error Rate (BER) for on-chip wireless links [39, 60, 99]. We adopt the collision and packet loss handling protocols from past work [1, 38].

3.2 Deep Reinforcement Learning

We provide a brief primer on RL based on [84]. In RL, an *agent* interacts with an *environment*, and learns to generate a policy directly from experience as shown in Fig. 3. In our case, NeuMAC is the *agent*, the multiprocessor is the *environment*, and the generated MAC protocol is the policy.

• **Agent & Environment:** The *agent* starts with no apriori knowledge. Then, at each time step t, the agent observes the state s_t of the environment, and takes an action a_t. Following the action, the environment transitions to state s_{t+1}, and the agent receives a reward r_t. The state transitions and the rewards are stochastic and assumed to have the Markov property.

[1]Note that other technologies such as optical links have poor performance [2, 9, 37].

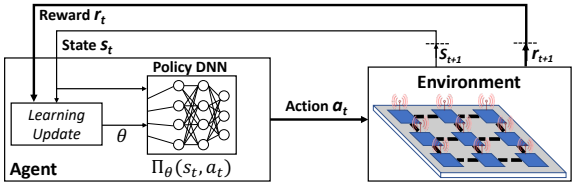

Figure 3: Deep Reinforcement Learning Framework.

During training, the *agent* gains experience by taking actions and observing the state transitions and rewards in response to these actions. The actions the *agent* takes aim to maximize an objection function known as the expected cumulative discounted reward: $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where $\gamma \in (0, 1]$ is the discount factor for future rewards.

• **Policy:** The action a_t picked by the *agent* is dictated by a *policy* π, where π represents a probability distribution over the space of actions and states : $\pi(s,a) \rightarrow [0,1]$. That is, $\pi(s,a)$ is the probability that action a is taken in state s by the agent following policy π. For most large-scale practical problems, the policy π is modeled with a Deep Neural Network (DNN), as they are very powerful function approximators. The DNN is parameterized by θ, which are the learnable parameters of the model, and we represent the policy as $\pi_\theta(s,a)$. θ is also referred to as the *policy parameters*.

• **Training:** The objective of training in RL is to learn the policy parameters θ so as to maximize the expected cumulative reward received from the environment. Towards this end, we focus on a class of RL algorithms called *policy gradient algorithms*, where the learning takes place by performing *gradient descent* on the policy parameters. In practice, the training methodology follows the *Monte Carlo method* where the agent samples multiple trajectories obtained by following the policy π_θ, and uses the empirically computed cumulative discounted reward as an unbiased estimator of the expected value. This empirical value is then used to update the policy parameters via the gradient descent step. The result is a known algorithm: REINFORCE which we use in this paper. For more details, we refer the reader to [84].

4 NeuMAC Design

4.1 Overview

NeuMAC consists of two components. (1) A standard NoC multicore processor with N cores where each core has been augmented with a wireless transceiver as shown in Fig. 2. (2) A NeuMAC agent that periodically generates new medium access policies based on the traffic patterns it sees on the wireless NoC. The agent is housed in a simple neural accelerator that resides on the same chip with a small area and power overhead (See Appendix A for hardware details).

Fig. 4 shows the working of NeuMAC. The NeuMAC agent is equipped with a wireless transceiver through which it can listen on the channel, and also send protocol updates to the

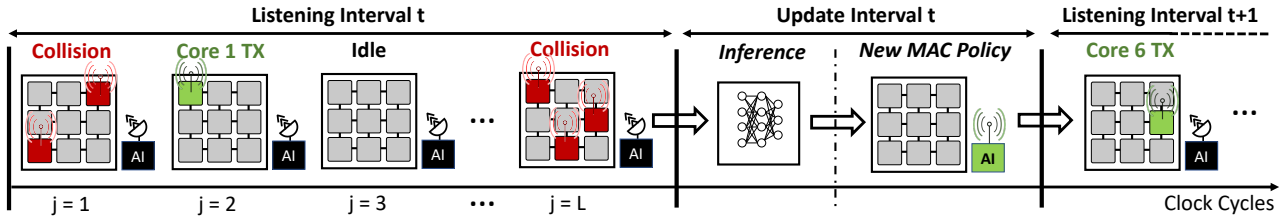

Figure 4: An Overview of NeuMAC's Protocol

cores. The NeuMAC agent listens on the wireless channel for a period called the "*Listening Interval*" where it collects traffic data about core transmissions, collisions and idle slots. It, then, feeds this data to a trained RL neural network that implicitly predicts the future traffic patterns and generates a new policy to be used as the medium access protocol during the next *Listening Interval*. NeuMAC updates the policies at the cores by sending an update message with the policy parameters. Each *Listening Interval* and *Update Interval* constitute a single step in the RL framework.

One point to note is that, although the cores share a common clock for their normal CPU operation[2], it is infeasible to coordinate medium access for each clock cycle through a shared centralized scheduler, since the exchange of control messages between the cores and the scheduler would itself incur latencies of multiple clock cycles. [6]

4.2 Design Challenges

The above design is governed by several strict timing and resource constraints of wireless NoC. In particular, it must address the below challenges while at the same time ensuring NeuMAC's ability to generate versatile and expressive medium access protocols to service the dynamic and fast-varying traffic patterns.

C1. *Centralized Agent:* Ideally, we would have wanted NeuMAC to adopt a distributed design where every core is equipped with its own NeuMAC agent that dictates its own MAC protocol. However, introducing a neural accelerator at every core would be prohibitively expensive in terms of area and power. Hence, NeuMAC is constrained to a centralized approach with a single agent.

C2. *Cores to Agent Communication Overhead:* To obtain an accurate view of traffic patterns, NeuMAC must obtain the packet injection rate and buffer occupancy across time at each core in the network. However, relaying this information from every core back to the centralized agent would result in huge communication overhead. Instead, NeuMAC leverages the broadcast nature of wireless networks to collect traffic patterns simply by listening for transmissions on the wireless medium. While the collected information is less expressive than the history of packet injection and buffer occupancy at each core, it retains sufficient information to allow NeuMAC to predict traffic patterns while at the same time completely eliminating communication overhead from the cores to the centralized agent.

C3. *Agent to Cores Communication Overhead:* One option is to have the agent tell each core whether to transmit or not at every CPU clock cycle. However, this would require running inference and relaying information to each core at every clock cycle which would lead to prohibitively large communication overhead. To address this, NeuMAC amortizes the communication overhead (*Update Interval*) from the agent to the cores by performing inference once every *Listening Interval* spanning thousands of clock cycles. In our implementation, we use an interval of $L = 10{,}000$ clock cycles ($10\mu s$) which is large enough to reduce the overhead to less than 6% and small enough to ensure that the traffic patterns remain stable and can be learned by the RL agent.

C4. *Complexity of the MAC Policy:* NeuMAC generates a policy that dictates the MAC protocol of each core for the following *Listening Interval*. Ideally, NeuMAC would generate a deterministic transmission schedule for every core to follow. Such a design is extremely expressive since it could allow NeuMAC to generate any possible schedule. However, such a design would require the RL deep neural network to output an action space with $N \times L$ dimensions where N is the number of cores and L is the number of clock cycles (e.g. 10,000). Such a neural network would be unsuitable for a resource-constraint setting like NoC. To address this, we carefully design a parameterized MAC policy that can support a flexible range of medium access protocols while ensuring that the neural network only needs to output a few parameters to dictate the desired policy.

C5. *Reward engineering:* The reward during training needs to be designed so as to guide NeuMAC towards the high-level objective. While most past work on learning link-layer and network-layer protocols only use network-level metrics such as throughput and latency for the reward signal, in our case we need to choose domain specific rewards so as to optimize for the end goal, which is application execution speedup on the multicore.

C6. *Low Footprint Neural Network:* NeuMAC's neural network must adhere to strict timing, power and area constraints of a chip multiprocessor. Thus, our design cannot simply adapt a known RL model as it would require large amounts of memory and computational resources, and would also suffer high inference latencies (tens of milliseconds) [46, 63].

[2] Unlike a distributed system of machines, a shared clock for a manycore system is feasible since all cores are housed on the same silicon die.

To address this, we design NeuMAC's RL framework such that the state space (input to the neural network) and action space (output) scale linearly with the number of cores. Our design ensures that NeuMAC is expressive enough while at the same time can operate under NoC's resource constraints.

4.3 NeuMAC's MAC Policy

As discussed above, the MAC policy that the agent dictates to the cores should have the following properties:

1. The policy should span a wide range of protocols, all the way from TDMA to CSMA.
2. It should be possible to describe the policy with few parameters to reduce the communication overhead and the output of the neural network.
3. It should allow for a simple neural network architecture to learn a mapping from observed traffic patterns to the most efficient MAC protocol.

In order to achieve these properties, we adopt a two-layer protocol design. The first layer consists of a deterministic underlying TDMA schedule, where each core is assigned a unique time slot for transmission in a round-robin fashion. For example, for time slots $j \in [1, \cdots, L]$, core i is assigned the slots $\{j \mid j \mod N = i\}$ where N is the number of cores. The second layer consists of a probabilistic transmission schedule like CSMA, where each core is assigned a contention probability. Specifically, during its assigned time slot, core i transmits on the channel with probability 1 if it has an outstanding packet in its buffer. During other cores' assigned time slots, core i can transmit with probability p_i. In the event of a collision, exponential backoff is implemented by halving p_i of the colliding cores similar to CSMA. On the other hand, if a transmission is successful, p_i is reset to it's initial value.

To generate this policy for an NoC with N cores, the RL neural network needs to output an action space that can be defined as $a_t = [a_{1,t}, a_{2,t}, \ldots, a_{N,t}]$ where $a_{i,t} \in [0,1]$ represents the initial contention probability of core i during *"Listening Interval" t* (i.e., time step t in the RL framework). The contention probability of core i is then initialized as $p_i = a_{i,t}$. Different choices of a_t result in different protocols on the multicore. For instance, setting $a_{i,t} = 0$ for all i results in a simple TDMA protocol since every core only transmits on the channel during its assigned slot. On the other hand, $a_{i,t} = c > 0$ for all i mimics a CSMA-like protocol with varying degrees of aggressiveness on the channel. The pseudo code for NeuMAC's protocol is presented in Alg. 1.

The above formulation satisfies our design objectives. First, it enables NeuMAC to gracefully shift between a pure TDMA and a CSMA scheme, while supporting all intermediate protocols. The design also gives the flexibility to control each core individually, so that the NeuMAC can potentially increase contention probabilities for cores that observe high traffic intensity. Second, since the MAC protocol at core i is characterized by only one number (the contention probability $a_{i,t}$), there is very small communication overhead during the *Update Interval*, where the NeuMAC agent has to transmit a single broadcast packet with N numbers. Each core, receives the packet and extracts it own contention probability. Finally, the design keeps the action space constrained and linear in the number of cores which allows for a simple neural network that can be easily trained and is more likely to converge.

Algorithm 1 NeuMAC Protocol

$L \leftarrow$ Number of Clock Cycles in Listening Interval
$[a_{1,t}, a_{2,t}, \ldots, a_{N,t}] \leftarrow$ Action space generated by RL agent at time step$t$
$[p_1, p_2, \ldots, p_N] \leftarrow [a_{1,t}, a_{2,t}, \ldots, a_{N,t}]$

At core i:
for $j \in \{1, \cdots, L\}$ **do**
  $Buffer_i(j) \leftarrow$ Outstanding packet in the buffer for core i
  if $Buffer_i(j) \neq \emptyset$ **then**
    if $j \mod N = i$ **then** ▷ TDMA Slot Assigned to Core i
      Transmit with probability 1
    else
      Transmit with probability p_i
  if Transmission from Core i collides **then**
    $p_i = p_i/2$
  else
    $p_i = a_{i,t}$

4.4 RL Formulation and Training

Given the above design, we now formalize the state space, reward, policy and training of NeuMAC's RL framework.

• **State Space Design:** The NeuMAC agent takes state information s_t as input and generates a MAC policy characterized by the action space a_t described above. The state information is generated purely by listening to ongoing transmissions on the channel. As described earlier, this allows us to eliminate all communication overhead from the cores to the RL agent. However, it only provides information about the activity on the channel rather than the traffic injection into the network. Moreover, in the event of a collision, NeuMAC cannot know which cores attempted to transmit. Despite these limitations, NeuMAC's state space retains enough information to infer traffic patterns. In particular, during each CPU cycle, NeuMAC will either detect an idle channel, a collision, or a successful transmission from some core i. We define our state at time step t, s_t, as an $(N+1) \times 1$ vector that keeps track of the number of successful transmissions from each core and the number of collisions observed during the cycles in the RL time step (*Listening Interval*). Specifically, the i^{th} element of s_t counts the number of successful packet transmissions by core i, and the $N+1^{th}$ element counts the number of collisions. The number of idle slots is implicitly encoded in the state since it is equal to $L - \sum_{i=1}^{N+1} s_{i,t}$ where L is the number of cycles in a *Listening Interval*. The state s_t is then used by the NeuMAC agent to generate the MAC protocol policy for the next time step.

• **Reward Engineering:** The reward signal is designed to guide the agent towards policies that optimize for the desired objective. Most past work that uses RL for learning networking protocols employs network-level metrics like throughput or latency as the reward signal. However, in our case, we need the reward signal to directly represent our end goal, which is to optimize for speedups in application execution time on the multicore. While network-level metrics like throughput are correlated to the execution time, they do not always capture the intricate dependencies between the execution on threads and packet delivery on the network. In Section 6, we see that there are instances where a protocol performs significantly worse in terms of average network throughput, but still has better end-to-end application execution time.

As a result, we design our reward signal to reflect our high level objective of minimizing application execution time. Specifically, for each time step t, the reward is set to $-L_t$ where L_t represents the number of clock cycles where the application was executing. Hence, for all but the last time step, the reward signal r_t is set to $-L$. For the last time step, reward is set to $-k$, where k is the number of clock cycles at which the application terminates execution. The intuition behind this choice for the reward signal is as follows. Recall that the objective of reinforcement learning is to maximize the cumulative reward, i.e. $-\sum_t L_t$. This is equivalent to minimizing $\sum_t L_t$, which ultimately means the application utilizing fewer CPU clock cycles for execution. While this choice of reward signal does correlate with improving network-level metrics such as packet latency and throughput, it is not the central objective and thus it is possible that sometimes the NeuMAC agent compromises on network performance for improvement in execution time. Note that in our formulation, we set the discount factor $\gamma = 1$.

• **Policy:** We represent our policy π as a deep neural network (also called policy network) which takes as input the state s_t, and maps it to a_t in the action space. Note that in our problem, the action space is continuous. In such cases it is common to discretize the continuous action space $a \in [0,1]^N$ similar to [52], and convert the problem into a classification problem where the agent now chooses which combination of a_i's to pick. However, an obvious issue with this approach is the curse of dimensionality. Even with 2 quantization levels for each a_i, the total number of discretized actions in $a \in [0,1]^N$ becomes 2^N. Thus the neural network architecture needs to have an output dimension of 2^N which becomes infeasible for our resource constrained environment.

Therefore, we avoid discretizing the action space and, instead, model the actions as following a Gaussian distribution with mean μ and variance σ. The deep learning model is now trained to output the parameters of this Gaussian distribution, as described in [84]. The NeuMAC agent picks the action for the next time step simply by sampling from the distribution $\mathcal{N}(\mu, \sigma)$. In NeuMAC, the policy network outputs N parameters μ_i corresponding to N distributions, one for each core i. The variance σ is set to 1 at the start of training to encourage exploration, and annealed down to 0.05 as NeuMAC's policy improves. Finally, during inference, the variance σ is set to 0.05, the action $a_{i,t}$ for core i is sampled from the corresponding distribution $\mathcal{N}(\mu_i, \sigma)$, and clipped to ensure that $a_{i,t} \in [0,1]$.

• **Training Algorithm:** We train our policy network end-to-end in an *episodic* setting. In each episode, an instance of an application is executed on the multicore, and the wireless network on chip follows the MAC protocol as dictated by the NeuMAC's policy network. The episode terminates when the application completes execution. In order to learn a policy that generalizes well, we train the network for multiple episodes with each episode observing a different application trace. For every episode, we run M separate Monte Carlo simulations to explore the probabilistic space of possible actions using the current policy, and use the resulting data to improve the policy for all applications. Specifically, we record the state, action, and reward information for all time steps of each episode. We then use this data to train our policy using the popular REINFORCE algorithm along with a baseline subtraction step, as described in [67].

4.5 Neural Network Architecture

Our network is composed of three fully connected layers with 128, 128 and 64 neurons respectively. The first two layers are followed by ReLU activation units, whereas the final layer is followed by a sigmoid unit to output the probability values a_i's between 0 and 1. During training, the weights use 16 bit floating points. Once trained, the learned weights are quantized to 8 bit fixed points for the inference stage. This is standard for run-time optimization in deep learning [53], and does not adversely affect performance.

The proposed fully connected network architecture here is simple and ties in very well with our design objectives. Recall that NeuMAC performs one inference step every 10,000 CPU clock cycles, and we require the inference step to add little overhead. The architecture here is composed of 32,000 learnable parameters, and at 8-bit quantization, it can be stored in a 32 KB on-chip SRAM cache to ensure fast memory accesses. Since inference latencies in most neural network architectures tend to be memory bound (including Fully connected and CNN architectures) [26, 53], improving memory access latencies plays a big role in speeding up overall inference time. Further, the simple structure of a fully connected network allows for straightforward memory access patterns, since the inference step is a straightforward computation amounting to consecutive matrix multiplications. In Appendix A we provide energy-delay characterization of this architecture.

One point to note is that NeuMAC's deep RL agent is trained offline, and does not undergo any training during runtime since training is resource intensive. However, retraining can be triggered periodically depending on performance re-

Name	Description
BFS [13]	Breadth-first search
Bodytrack [20]	Tracking a body-pose through images
Canneal [20]	Compute optimal routing for gates on a chip
CC [13]	Compute connected components of a graph
Pagerank [13]	Compute pagerank for nodes in a graph
SSSP [13]	Single source shortest path
Volrend [96]	Rendering of 3D objects
StreamCluster [20]	Cluster streams of points
Community [13]	Compute modularity of a graph

Table 1: Summary of Applications

quirements and this retraining will be performed offline. The updated model parameters can then be migrated to the neural hardware accelerator by simply rewriting the SRAM memory blocks on the accelerator corresponding to the neural network's model parameters. This update can happen through the multicore's wireless NoC communication channel and won't add much overhead since our model is restricted to just 32,000 parameters, each of 8 bits.

5 Implementation

Evaluation Environment: We evaluate NeuMAC on a cycle-level execution-driven architectural simulator, Multi2sim [87]. Multi2sim is a popular end-to-end heterogenous system simulator tool used in the architecture community to test and validate new hardware designs with standard benchmarks. We evaluate NeuMAC for multicores with core count $n = 64$ at 22nm technology running at 1GHz. We use the same architecture parameters as [38]. We augment Multi2sim with an on-chip wireless network that accurately models transmissions, collision handling and packet losses.

While NeuMAC could be potentially trained directly using multi2sim, it is extremely slow and would result in prohibitively large training times. Therefore, for NeuMAC's training phase, we use a light-weight custom-built Wireless Network-on-Chip simulator along with traffic traces captured from Multi2sim. Our custom simulator models the data dependencies and synchronization primitives (such as locks and barriers) in the applications, so as to faithfully mimic the behavior of multi-threaded applications.

In order to evaluate NeuMAC's generalizability and effectiveness for a broad use case, we test NeuMAC on 9 different applications chosen from diverse domains such as graph analytics, vision, and numerical simulations (Summary in Table 1). Additionally, we also test with multi-application jobsets where different groups of cores are executing different multithreaded applications. While training is performed using our custom simulator, we evaluate NeuMAC using Multi2sim. We integrate Multi2sim with NeuMAC's trained RL agent, and our evaluations account for the RL agent's DNN inference latency and communication latency between the multicore and RL agent.

Training and Evaluation Details: For each application, we collect 500 different traces, each generated with different inputs to the applications in order to capture the variations between different runs. We evaluate NeuMAC using k-fold cross validation, where we train the model on 8 applications and test performance on the ninth application. Thus, we ensure that the NeuMAC agent is never explicitly trained on the application it is being evaluated on, and our results show that NeuMAC can generalize well to different applications. We train NeuMAC for a total of 4000 episodes, and for each episode we run $M = 16$ Monte Carlo simulations in parallel. The policy network is trained using ADAM optimizer [55] with a learning rate of 0.001.

6 Evaluation Results

6.1 Baselines

We compare with the following baselines:

(1) **CSMA with Exponential Backoff:** CSMA/CA protocol from 802.11 networks, with backoff window ranging from 1 to 1024. [1, 71] use CSMA MAC in the context of WNoCs.

(2) **TDMA:** Cores are allocated fixed slots for transmission in round-robin fashion. [5, 34] evaluate TDMA for WNoCs.

(3) **Switch-thresh:** [38, 65] propose a protocol that switches between a static CSMA and a static TDMA protocol based on per-core preset thresholds for channel activity and buffer occupancy. The optimal threshold values vary across applications and we choose values that are best in the average case.

(4) **Optimal CSMA Algorithm:** There is a large body of work that designs throughput optimal CSMA algorithms. However, most of these works are theoretical, and make simplifying assumptions like ignoring collisions or static traffic arrival rates, due to which they perform significantly worse than even regular CSMA protocols in practice. Among the optimal CSMA algorithms we tested, we found queue-based algorithms to perform best. We implement an extension of the popular Q-CSMA algorithm [79], where each node uses its buffer queue buildup to infer its transmission aggressiveness on the channel. While this algorithm is not truly distributed in nature, we ignore the global communication overheads in evaluations to favor the baseline performance.

(5) **Wired Baseline:** We also compare performance against a purely wired baseline, where all cache coherency traffic is serviced through the wired network-on-chip.

(6) **Infinite Capacity Channel:** We also compare NeuMAC's performance against an oracle with infinite channel capacity where the wireless medium can support multiple concurrent transmissions without suffering collisions, and every packet can be transmitted immediately without any channel contention delays. This baseline gives us an upper bound on how much improvement in end-to-end execution time is possible from improving the wireless NoC performance.

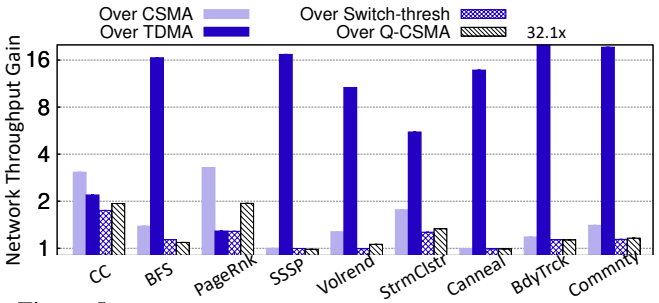

Figure 5: Gains in Wireless Network Throughput. (y axis in logscale)

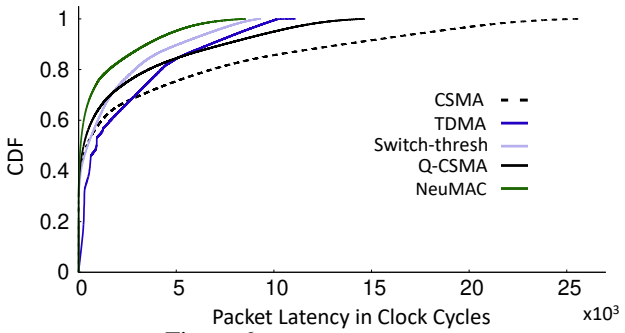

Figure 6: CDF of packet latency

Apps	CSMA	Switch-thresh	Q-CSMA	NeuMAC
CC	75.30%	55.58%	76.24%	8.72%
BFS	50.42%	28.28%	49.57%	3.81%
Pagernk	77.36%	11.26%	77.79%	2.19%
SSSP	11.08%	9.48%	9.44%	8.88%
Volrend	44.17%	7.93%	46.11%	2.49%
Strmclstr	62.57%	19.21%	62.69%	31.24%
Canneal	2.55%	2.87%	2.09%	2.04%
Bdytrck	30.5%	29.06%	29.8%	28.87%
Cmmnty	46.76%	32.02%	49.24%	5.8%

Table 2: % of Collisions

6.2 Quantitative Results

We first evaluate NeuMAC's performance against baselines on single application executions, followed by evaluations on the more realistic scenarios where multiple applications are running on the multicore. We also test NeuMAC's performance under lossy network conditions, and conclude by presenting scaling results where we demonstrate that NeuMAC's gains increase as the multicore scales to thousands of cores.

A. Single Application Wireless Network Performance:
We begin by evaluating the wireless network performance against baselines along three metrics – (i) Wireless network throughput, (ii) Packet latency on the wireless network, and (iii) Number of collisions on the channel. We note that while NeuMAC is not explicitly trained to optimize for network metrics, their performance is correlated to faster execution times on the NoC.

(i) Network Throughput: In Fig. 5, we plot the gains in average network throughput achieved by NeuMAC against the baselines. Compared to CSMA and TDMA, NeuMAC achieves a mean improvement of 1.8× and 9.63× respectively across the benchmarks, and a maximum improvement of 3.3× and 32.1× respectively. TDMA has poor performance for average network throughput since cores have to wait for their turn to transmit even when the traffic is sparse, which leads to underutilization of channel.

Compared to Switch-thresh and Q-CSMA, NeuMAC achieves a mean improvement of 1.2× and 1.33×, and a maximum improvement of 1.7× and 1.9× respectively. While these protocols are improve over CSMA and TDMA, they still cannot react and adapt quickly enough to accommodate the fast changing traffic patterns on the multicore.

(ii) Packet Latency: In Fig. 6, we plot the CDF of packet latency due to queuing in the Wireless Network-on-Chip across all applications. It is interesting to note that while at the tail TDMA performs better than CSMA, in the median case TDMA performs significantly worse than CSMA. This is because the high packet latencies at the tail are due to dense traffic in the network which TDMA is better suited for, whereas at the median where traffic is less dense, TDMA leads to much higher packet latencies. NeuMAC, on the other hand, is able to adapt to all these different scenarios and provides an improvement in packet latency across all baselines. Over CSMA and TDMA, NeuMAC improves median packet latency by 4.11× and 9.18×, and improves 90^{th} percentile latency by 3.89× and 1.92× respectively. Over Switch-thresh and Q-CSMA, the gains respectively are 4.66× and 2.56× at the median, and 1.47× and 2.13× at 90^{th} percentile.

(iii) Collisions on Wireless Channel: In Table 2 we show % of collisions on the wireless channel across different benchmarks. We omit TDMA here since TDMA by design does not suffer from collisions. As observed, NeuMAC has significantly fewer collisions than the CSMA algorithms. Switch-thresh is the next best performing protocol, but NeuMAC in most cases still has fewer collisions.

B. Single Application End-to-End Execution Speedup:

(i) Speedups over Purely Wired Network-on-Chip: In Table 3, we show application speed-ups achieved by NeuMAC and the Infinite Capacity baseline respectively, over the purely wired NoC. NeuMAC can speed up benchmarks by up to 9.7× for StreamCluster and 6.53× for BFS, and on average provides a speedup of 3.42× across benchmarks. Additionally, we see that NeuMAC gets very close to the upper bound of the speedup value, achieving up to 99.5% of the maximum speedup possible in the case of BFS, and 98% of the maximum speedup possible on average. This result demonstrates that NeuMAC is able to fully exploit the potential offered by the wireless NoC.

(ii) Speedups over Baselines: Fig. 7 shows execution time gains of NeuMAC over the baselines on the wireless NoC. As can be observed, there is no one baseline protocol that performs well across all applications. While in applications

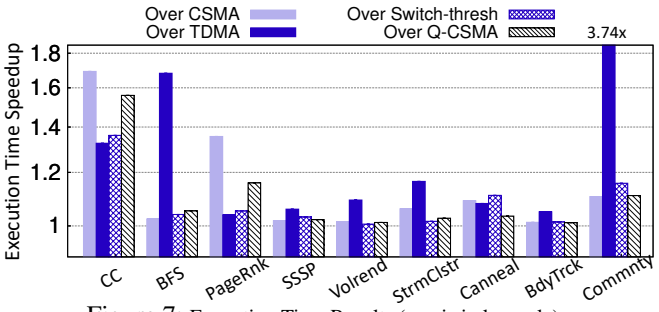

Figure 7: Execution Time Results (y axis in logscale)

Apps	NeuMAC	Inf. Cap. baseline	% Achieved
CC	1.96x	2.06x	95%
BFS	6.53x	6.56x	99.5%
Pagerank	1.07x	1.11x	96.4%
SSSP	2.24x	2.25x	99.5%
Volrend	1.32x	1.33x	99.2%
Strmclstr	9.70x	9.77x	99.28%
Canneal	1.14x	1.15x	99.13%
Bodytrack	1.37x	1.38x	99.3%
Community	3.77x	3.82x	98.6%

Table 3: Speedups over Purely Wired Network-on-Chip.

like Pagerank, TDMA performs the best, in other applications such as BFS it is significantly worse. NeuMAC, on the other hand, performs well across all benchmarks. In Table 4, we see that NeuMAC achieves a maximum of 69.18% speedup over CSMA for CC and 274.56% speedup over TDMA for Community, and compared to Switch-thresh and Q-CSMA, NeuMAC offers speedups up to 37.09%-55.94%.

C. Multi-Application Jobs: In Table. 5, we present execution time speedup results for multiapplication runs on the multicore. For each run, we randomly choose one application among the 9, and execute it using either 4, 16 or 32 threads. We choose a sufficient number of applications such that all 64 cores are utilized, and in total we test on 100 different multiapplication jobsets. Note that the NeuMAC agent was never explicitly trained on such multiapplication traffic traces. From Table. 5, we can see that NeuMAC's gains increase over the baselines compared to single benchmark experiments (Table. 4), and goes as high as 6.15× (515.04%) speedup over TDMA. These higher gains in multiapplication jobsets can be attributed to the more complex nature of packet dependencies between threads, which NeuMAC can exploit to further speed up execution time as illustrated in Section 2.

C. Lossy Networks: To evaluate NeuMAC's robustness to varying channel conditions, we conduct experiments in lossy network settings. We vary the packet loss rates in the wireless NoC from 0% up to 10%, and in the event of a loss, the packet is retransmitted. In Fig. 9, we compare the average application speedup achieved over the baselines as the loss rate increases. We observe that NeuMAC is able to generalize very well to varying channel conditions and loss rates, and

Speedups	CSMA	TDMA	Switch-thresh	Q-CSMA
Max	69.18%	274.56%	37.09%	55.94%
Min	1.26%	4.88%	0.63%	1.12%
Mean	18.21%	46.90%	9.73%	11.94%

Table 4: Summary of Execution Time Speedups by NeuMAC. The per-application speedups are shown in Fig. 7.

Speedups	CSMA	TDMA	Switch-thresh	Q-CSMA
Max	93.18%	515.04%	48.16%	26.78%
Min	13.3%	24.72%	4.41%	5.82%
Mean	33.93%	166.32%	19.97%	17.48%

Table 5: Summary of Execution Time Speedups by NeuMAC for Multiapplication runs

can maintain the same gains over the baselines throughout. Note that NeuMAC was never trained explicitly for lossy network settings. Despite this, it is able to generalize since it can implicitly infer the channel conditions from the channel activity like increased number of collisions.

We also test NeuMAC's sensitivity to errors in the observed state caused by packet losses at the NeuMAC agent's transceiver during the *"Listening Interval"*. We conduct experiments where we vary the packet loss rate from 0% to 2% in order to introduce noise in the observed state. We find that even under 2% loss rate, NeuMAC's suffers a median performance degradation of only 0.85% across all benchmarks compared to its performance with perfect state information.

D. Scaling Trends: We believe that a learning based approach like NeuMAC can greatly benefit the wireless NoC performance as the number of cores scale to thousands of cores. To demonstrate this we show the gains that NeuMAC achieves over baseline protocols for different metrics as the cores vary from 4 to 1024 in Fig. 8. Since multi2sim and other architectural simulators cannot scale beyond a hundred cores, we evaluate these results in our custom simulator by training a separate NeuMAC model for each core count. From Fig. 8, we can see that NeuMAC's gains over the baselines scale favorably with the number of cores. This is because NeuMAC is able to generate fine-grained MAC protocols by controlling the actions of each core individually, and thus can generate highly optimized protocols that improve substantially upon the baselines at high core counts.

7 Related Work

A. Wireless Network-on-Chip Protocols: The majority of past networking research on wireless NoC does not leverage the broadcast nature of wireless to enable instantaneous cache synchronization and instead focuses on using wireless only between far apart cores to reduce the latency. These complementary works focus on problems related to optimizing network topology [32, 35, 105], packet routing [61, 90, 106], flow control [18, 42] and improving the reliability of the PHY layer for far apart cores [76, 85, 86]. However, such designs have limited gains over wired NoCs [4]. More recent work in architecture research exploits the broadcast nature of wireless to

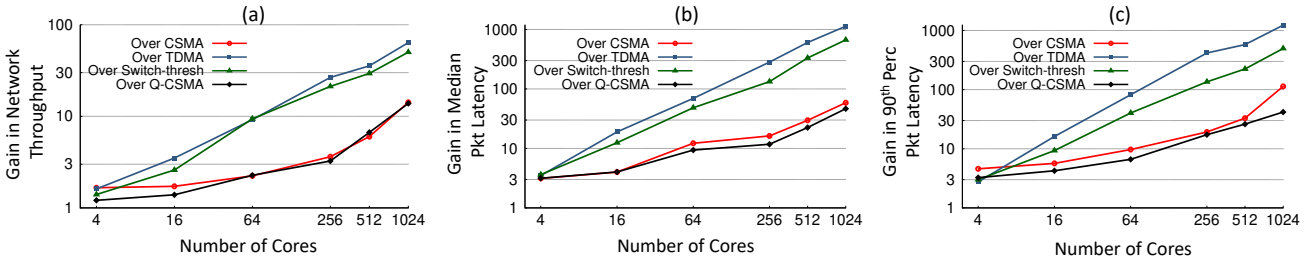

Figure 8: Scaling Trends in NeuMAC's Gains for (a) Wireless Network Throughput (b) Median Packet Latency and (c) 90^{th} Percentile Packet Latency

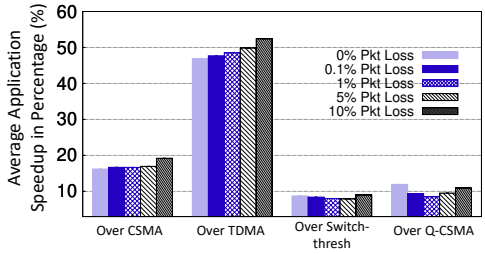

Figure 9: Effect of Packet losses on NeuMAC's application speedup performance compared to Baselines.

boost the performance of wireless enable NoCs [34,38,65,71]. These systems either use *contention-free* mechanisms such as token passing [34] or *contention-based* mechanisms such as carrier sense with exponential backoff [29,71]. The closest to our work are [38,65] which attempt to adapt to traffic patterns by switching between a CSMA or a token passing protocol based on a preset threshold. However, hand tuning the threshold values is a challenging task and does not provide the flexibility and expressibility of NeuMAC to support complex and highly variable traffic patterns.

B. Network-on-Chip Technologies: Past work on wired NoCs proposes the use of deep learning and RL to learn efficient packet routing protocols [98], learn memory access patterns to reduce cache misses [103], and reduce static and dynamic power consumption on an NoC [36]. To the best of our knowledge, ours is the first work that attempts to exploit deep reinforcement learning techniques to generate medium access protocols for Wireless NoCs.

C. Deep Learning in Wireless Networks: Deep RL has recently been applied in wireless networks to optimize duty cycling in sensor networks [64], resource allocation in cellular networks [22,27], dynamic spectrum access [72,92], rate adaptation in CSMA networks [69],and control policies at the PHY layer [52]. [104] provides an extensive survey of deep learning in wireless networks. The closest to our work are [14,16,19,101] which use reinforcement learning to modify the backoff parameters in CSMA or decide whether to transmit or not for every packet at every time step. However, such designs are not applicable in the context of wireless NoCs owing to the unique set of constraints imposed by the NoC, such as the much smaller time-scale of operation rendering neural network inference per transmission slot infeasible, the limited SRAM memory to store model parameters and the enormous action space to explore. These constraints require significant redesign to NeuMAC's deep RL framework where it has to now generate high-level, versatile and adaptable protocols that can be deployed for thousands of clock cycles, and generating such protocols cannot be reduced to a simple classification task per transmission-slot (e.g. transmit or not).

8 Limitations and Discussion

Some points are worth noting: First, given the enormous costs and engineering efforts involved in prototyping a full chip with integrated processors, memory, and NoC, it is outside the scope of this work to implement NeuMAC in hardware. As a result, we evaluate NeuMAC on a full-system cycle-accurate architectural simulator, as is the norm among computer architecture researchers. These full-system simulators exhaustively model all components of a CPU and also ensure that all timing dependencies are simulated accurately [87]. As a result, the trends and insights obtained from such architectural simulations often carry over to full fledged prototypes. Moreover, the wireless channel in this WNoC application domain is in fact very stable as opposed to WLAN channels which are extremely dynamic. This is because the multicore is isolated in a chip package, and the wireless channel can be precisely measured and characterized, thus allowing compensation for multipath fading and other artifacts. As a result, the wireless BER in these environments can be as low as 10^{-16} [33], making such a simulation based evaluation representative.

Second, in parallel programming for multicore processors, programmers today try hard to avoid broadcast transmissions as the overhead of running the cache coherency protocol is high. With wireless NoC, the overhead of broadcast traffic is now limited which opens the door to rewriting applications in a manner that embraces broadcast, and can in turn benefit even more from an adaptive protocol like NeuMAC.

Lastly, in this paper we focus on the MAC layer since it is considered a roadblock to realize the full potential of wireless NoCs. However, studying the challenges and opportunities at the other layers such as PHY remains exciting and promising avenue which we leave for future work.

Acknowledgements: We thank the shepherd and anonymous reviewers for their feedback and comments. We also thank Ameya Patil for helping guide the hardware accelerator design and characterization. The work is funded in part by NSF Award 1750725.

References

[1] S. Abadal, A. Cabellos-Aparicio, E. Alarcon, and J. Torrellas. Wisync: An architecture for fast synchronization through on-chip wireless communication. *ACM SIGOPS Operating Systems Review*, 50(2):3–17, 2016.

[2] S. Abadal, M. Iannazzo, M. Nemirovsky, A. Cabellos-Aparicio, H. Lee, and E. Alarcón. On the area and energy scalability of wireless network-on-chip: A model-based benchmarked design space exploration. *IEEE/ACM Transactions on Networking (TON)*, 23(5):1501–1513, 2015.

[3] S. Abadal, R. Martínez, J. Solé-Pareta, E. Alarcón, and A. Cabellos-Aparicio. Characterization and modeling of multicast communication in cache-coherent manycore processors. In *Computers & Electrical Engineering*, 2016.

[4] S. Abadal, A. Mestres, R. Martínez, E. Alarcon, and A. Cabellos-Aparicio. Multicast on-chip traffic analysis targeting manycore noc design. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 370–378. IEEE, 2015.

[5] S. Abadal, A. Mestres, M. Nemirovsky, H. Lee, A. González, E. Alarcón, and A. Cabellos-Aparicio. Scalability of broadcast performance in wireless network-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 27(12):3631–3645, 2016.

[6] S. Abadal, A. Mestres, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio. Medium access control in wireless network-on-chip: a context analysis. *IEEE Communications Magazine*, 56(6):172–178, 2018.

[7] S. Abadal, M. Nemirovsky, E. Alarcón, and A. Cabellos-Aparicio. Networking challenges and prospective impact of broadcast-oriented wireless networks-on-chip. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, page 12. ACM, 2015.

[8] S. Abadal, B. Sheinman, O. Katz, O. Markish, D. Elad, Y. Fournier, D. Roca, M. Hanzich, G. Houzeaux, M. Nemirovsky, et al. Broadcast-enabled massive multicore architectures: A wireless rf approach. *IEEE micro*, 35(5):52–61, 2015.

[9] S. Abadal, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio. Orthonoc: a broadcast-oriented dual-plane wireless network-on-chip architecture. *IEEE transactions on parallel and distributed systems*, 29(3):628–641, 2018.

[10] S. Abadal, *et al.* OrthoNoC: A Broadcast-Oriented Dual-Plane Wireless Network-on-Chip Architecture. *IEEE Trans. Parallel Distrib. Syst.*, 2018.

[11] N. Abeyratne, R. Das, Q. Li, K. Sewell, B. Giridhar, R. G. Dreslinski, D. Blaauw, and T. Mudge. Scaling towards kilo-core processors with asymmetric high-radix topologies. In *Proceedings of the HPCA-19*, pages 496–507, 2013.

[12] A. B. Achballah, S. B. Othman, and S. B. Saoud. Problems and challenges of emerging technology networks- on- chip: A review. *Microprocessors and Microsystems*, 53:1–20, 2017.

[13] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan. Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores. In *2015 IEEE International Symposium on Workload Characterization*, pages 44–55. IEEE, 2015.

[14] R. Ali, N. Shahin, Y. B. Zikria, B.-S. Kim, and S. W. Kim. Deep reinforcement learning paradigm for performance optimization of channel observation–based mac protocols in dense wlans. *IEEE Access*, 7:3500–3511, 2018.

[15] Altera. An alternative to bus-based interconnects for large-scale design. In *White Paper*, 2008.

[16] S. Amuru, Y. Xiao, M. van der Schaar, and R. M. Buehrer. To send or not to send-learning mac contention. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015.

[17] M. Baharloo, A. Khonsari, P. Shiri, I. Namdari, and D. Rahmati. High-average and guaranteed performance for wireless networks-on-chip architectures. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 226–231. IEEE, 2018.

[18] J. H. Bahn and N. Bagherzadeh. Efficient parallel buffer structure and its management scheme for a robust network-on-chip (noc) architecture. In *Computer Society of Iran Computer Conference*, pages 98–105. Springer, 2008.

[19] H. Bayat-Yeganeh, V. Shah-Mansouri, and H. Kebriaei. A multi-state q-learning based csma mac protocol for wireless networks. *Wireless Networks*, 24(4):1251–1264, 2018.

[20] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[21] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baas. A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, pages 1–2. IEEE, 2016.

[22] U. Challita, L. Dong, and W. Saad. Deep learning for proactive resource allocation in lte-u networks. In *European wireless technology conference*, 2017.

[23] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[24] H. M. Cheema and A. Shamim. The last barrier: On-chip antennas. *IEEE Microw. Mag.*, 14(1):79–91, 2013.

[25] Y.-h. Chen, J. Emer, and V. Sze. Eyeriss v2: A Flexible and High-Performance Accelerator for Emerging Deep Neural Networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[26] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[27] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti. Cellular network traffic scheduling with deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[28] D. Culler, J. P. Singh, and A. Gupta. *Parallel computer architecture: a hardware/software approach*. Gulf Professional Publishing, 1999.

[29] P. Dai, J. Chen, Y. Zhao, and Y.-H. Lai. A study of a wire–wireless hybrid noc architecture with an energy-proportional multicast scheme for energy efficiency. *Computers and Electrical Engineering*, 45:402–416, 2015.

[30] G. De Micheli and L. Benini. Networks on chips: 15 years later. *Computer*, (5):10–11, 2017.

[31] S. Deb, K. Chang, X. Yu, S. P. Sah, M. Cosic, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo. Design of an energy-efficient cmos-compatible noc architecture with millimeter-wave wireless interconnects. *IEEE Transactions on Computers*, 62(12):2382–2396, 2012.

[32] S. Deb, A. Ganguly, K. Chang, P. Pande, B. Beizer, and D. Heo. Enhancing performance of network-on-chip architectures with millimeter-wave wireless interconnects. In *ASAP 2010-21st IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 73–80. IEEE, 2010.

[33] S. Deb, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo. Wireless noc as interconnection backbone for multicore chips: Promises and challenges. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(2):228–239, 2012.

[34] D. DiTomaso, A. Kodi, S. Kaya, and D. Matolak. iWISE: Inter-router wireless scalable express channels for network-on-chips (NoCs) architecture. In *2011 IEEE 19th Annual Symposium on High Performance Interconnects*, pages 11–18. IEEE, 2011.

[35] D. DiTomaso, A. Kodi, D. Matolak, S. Kaya, S. Laha, and W. Rayess. A-winoc: Adaptive wireless network-on-chip architecture for chip multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 26(12):3289–3302, 2014.

[36] D. DiTomaso, A. Sikder, A. Kodi, and A. Louri. Machine learning enabled power-aware network-on-chip design. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 1354–1359. European Design and Automation Association, 2017.

[37] R. K. Dokania and A. B. Apsel. Analysis of challenges for on-chip optical interconnects. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, pages 275–280. ACM, 2009.

[38] V. Fernando, A. Franques, S. Abadal, S. Misailovic, and J. Torrellas. Replica: A Wireless Manycore for Communication-Intensive and Approximate Data. In *ASPLOS*, 2019.

[39] D. Fritsche, *et al.* A Low-Power SiGe BiCMOS 190-GHz Transceiver Chipset With Demonstrated Data Rates up to 50 Gbit/s Using On-Chip Antennas. *IEEE Trans. Microw. Theory Techn.*, 65(9):3312–3323, 2017.

[40] S. H. Gade, S. S. Rout, M. Sinha, H. K. Mondal, W. Singh, and S. Deb. A utilization aware robust channel access mechanism for wireless nocs. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[41] S. Galzarano, A. Liotta, and G. Fortino. Ql-mac: A q-learning based mac for wireless sensor networks. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 267–275. Springer, 2013.

[42] A. Ganguly, K. Chang, S. Deb, P. P. Pande, B. Belzer, and C. Teuscher. Scalable hybrid wireless network-on-chip architectures for multicore systems. *IEEE Transactions on Computers*, 60(10):1485–1502, 2011.

[43] A. Gomes, D. F. Macedo, and L. F. Vieira. Automatic mac protocol selection in wireless networks based on reinforcement learning. *Computer Communications*, 149:312–323, 2020.

[44] S. K. Gonugondla, B. Shim, and N. R. Shanbhag. Perfect error compensation via algorithmic error cancellation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 966–970. IEEE, 2016.

[45] W. Gropp, W. D. Gropp, E. Lusk, A. D. F. E. E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[46] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski. Latency and throughput characterization of convolutional neural networks for mobile computer vision. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 204–215, 2018.

[47] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[48] M. Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

[49] Jack Clark. Intel: Why a 1,000-core chip is feasible, Press Release, 2010.

[50] N. E. Jerger, L.-S. Peh, and M. Lipasti. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *2008 International Symposium on Computer Architecture*, pages 229–240. IEEE, 2008.

[51] L. Jiang and J. Walrand. A distributed csma algorithm for throughput and utility maximization in wireless networks. *IEEE/ACM Transactions on Networking*, 18(3):960–972, 2009.

[52] S. Joseph, R. Misra, and S. Katti. Towards Self-Driving Radios: Physical-Layer Control using Deep Reinforcement Learning. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 69–74. ACM, 2019.

[53] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[54] A. Karkar, T. Mak, K.-F. Tong, and A. Yakovlev. A survey of emerging interconnects for on-chip efficient multicast and broadcast in many-cores. *IEEE Circuits and Systems Magazine*, 16(1):58–72, 2016.

[55] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[56] T. Krishna, L.-S. Peh, B. M. Beckmann, and S. K. Reinhardt. Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 71–82. ACM, 2011.

[57] T. Krishna, *et al.* Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication. In *Proceedings of the MICRO-44*, 2011.

[58] R. Kumar, T. Mattson, G. Pokam, and R. V. D. Wijngaart. The case for message passing on many-core chips. In *Multiprocessor System-on-Chip*, pages 115–123. Springer, 2011.

[59] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 408–419. IEEE, 2005.

[60] S. Laha, *et al.* A New Frontier in Ultralow Power Wireless Links: Network-on-Chip and Chip-to-Chip Interconnects. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 34(2):186–198, 2015.

[61] S.-B. Lee, S.-W. Tam, I. Pefkianakis, S. Lu, M. F. Chang, C. Guo, G. Reinman, C. Peng, M. Naik, L. Zhang, et al. A scalable micro wireless interconnect structure for CMPs. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 217–228. ACM, 2009.

[62] T. Lee, O. Jo, and K. Shin. Corl: Collaborative reinforcement learning-based mac protocol for iot networks. *Electronics*, 9(1):143, 2020.

[63] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang. Optimizing {CNN} model inference on cpus. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 1025–1040, 2019.

[64] Z. Liu and I. Elhanany. Rl-mac: a reinforcement learning based mac protocol for wireless sensor networks. *International Journal of Sensor Networks*, 1(3-4):117–124, 2006.

[65] N. Mansoor and A. Ganguly. Reconfigurable wireless network-on-chip with a dynamic medium access mechanism. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, page 13. ACM, 2015.

[66] N. Mansoor, S. Shamim, and A. Ganguly. A Demand-Aware Predictive Dynamic Bandwidth Allocation Mechanism for Wireless Network-on-Chip. In *Proceedings of the SLIP '16*, 2016.

[67] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2016.

[68] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. *arXiv preprint arXiv:1810.01963*, 2018.

[69] N. Mastronarde, J. Modares, C. Wu, and J. Chakareski. Reinforcement learning for energy-efficient delay-sensitive csma/ca scheduling. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2016.

[70] D. W. Matolak, A. Kodi, S. Kaya, D. DiTomaso, S. Laha, and W. Rayess. Wireless networks-on-chips: architecture, wireless channel, and devices. *IEEE Wireless Communications*, 19(5):58–65, 2012.

[71] A. Mestres, S. Abadal, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio. A mac protocol for reliable broadcast communications in wireless network-on-chip. In *Proceedings of the 9th International Workshop on Network on Chip Architectures*, pages 21–26. ACM, 2016.

[72] O. Naparstek and K. Cohen. Deep multi-user reinforcement learning for distributed dynamic spectrum access. *IEEE Transactions on Wireless Communications*, 18(1):310–323, 2018.

[73] J. Ni, B. Tan, and R. Srikant. Q-csma: Queue-length-based csma/ca algorithms for achieving maximum throughput and low delay in wireless networks. *IEEE/ACM Transactions on Networking*, 20(3):825–836, 2011.

[74] J. Oh, M. Prvulovic, and A. Zajic. Tlsync: support for multiple fast barriers using on-chip transmission lines. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 105–115. IEEE, 2011.

[75] S. Pasricha and N. Dutt. *On-chip communication architectures: system on chip interconnect*. Morgan Kaufmann, 2010.

[76] M. Rahaman and M. Chowdhury. Improved bit error rate performance in intra-chip rf/wireless interconnect systems. In *Proc. ACM/IEEE Great Lake Symp. VLSI*, 2008.

[77] S. Rajagopalan, D. Shah, and J. Shin. Network adiabatic theorem: an efficient randomized protocol for contention resolution. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 133–144, 2009.

[78] D. Sánchez, *et al.* An Analysis of On-Chip Interconnection Networks for Large-Scale Chip Multiprocessors. *ACM T. Archit. Code Op.*, 7(1), 2010.

[79] D. Shah, J. Shin, et al. Randomized scheduling algorithm for queueing networks. *The Annals of Applied Probability*, 22(1):128–171, 2012.

[80] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[81] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[82] Y. Solihin. *Fundamentals of parallel multicore architecture*. CRC Press, 2015.

[83] V. Soteriou, H. Wang, and L. Peh. A Statistical Traffic Model for On-Chip Interconnection Networks. In *Proceedings of MASCOTS '06*, 2006.

[84] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[85] X. Timoneda, S. Abadal, A. Cabellos-Aparicio, D. Manessis, J. Zhou, A. Franques, J. Torrellas, and E. Alarcón. Millimeter-wave propagation within a computer chip package. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[86] X. Timoneda, S. Abadal, A. Franques, D. Manessis, J. Zhou, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio. Engineer the channel and adapt to it: Enabling wireless intra-chip communication. *arXiv preprint arXiv:1901.04291*, 2018.

[87] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. Multi2sim: a simulation framework for cpu-gpu computing. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 335–344. IEEE, 2012.

[88] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, 2008.

[89] V. Vijayakumaran, M. P. Yuvaraj, N. Mansoor, N. Nerurkar, A. Ganguly, and A. Kwasinski. Cdma enabled wireless network-on-chip. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(4):28, 2014.

[90] C. Wang, W.-H. Hu, and N. Bagherzadeh. A wireless network-on-chip design for multicore platforms. In *2011 19th International Euromicro conference on parallel, distributed and network-based processing*, pages 409–416. IEEE, 2011.

[91] S. Wang and T. Jin. Wireless network-on-chip: A survey. *The Journal of Engineering*, 2014(3):98–104, 2014.

[92] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari. Deep reinforcement learning for dynamic multichannel access in wireless networks. *IEEE Transactions on Cognitive Communications and Networking*, 4(2):257–265, 2018.

[93] N. Weissman and E. Socher. 9mw 6gbps bi-directional 85–90ghz transceiver in 65nm cmos. In *2014 9th European Microwave Integrated Circuit Conference*, pages 25–28. IEEE, 2014.

[94] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, 2007.

[95] H.-S. P. Wong and S. Salahuddin. Memory leads the way to better computing. *Nature nanotechnology*, 10(3):191, 2015.

[96] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: Characterization and methodological considerations. *ACM SIGARCH computer architecture news*, 23(2):24–36, 1995.

[97] X. Xiang, *et al.* A model for application slowdown estimation in on-chip networks and its use for improving system fairness and performance. In *ICCD '16*, 2016.

[98] J. Yin, Y. Eckert, S. Che, M. Oskin, and G. H. Loh. Toward more efficient noc arbitration: A deep reinforcement learning approach. In *Proc. IEEE 1st Int. Workshop AI-assisted Des. Architecture*, 2018.

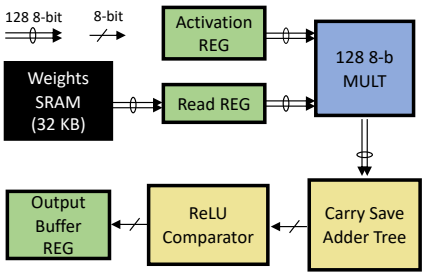

Figure 10: Illustrative Block Diagram of hardware macro employed for overhead characterization of NeuMAC's deep network

[99] X. Yu, J. Baylon, P. Wettin, D. Heo, P. P. Pande, and S. Mirabbasi. Architecture and design of multichannel millimeter-wave wireless noc. *IEEE Design & Test*, 31(6):19–28, 2014.

[100] X. Yu, H. Rashtian, S. Mirabbasi, P. P. Pande, and D. Heo. An 18.7-gb/s 60-ghz ook demodulator in 65-nm cmos for wireless network-on-chip. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(3):799–806, 2015.

[101] Y. Yu, T. Wang, and S. C. Liew. Deep-reinforcement learning multiple access for heterogeneous wireless networks. *IEEE Journal on Selected Areas in Communications*, 37(6):1277–1290, 2019.

[102] S.-Y. Yun, Y. Yi, J. Shin, and D. Y. Eun. Optimal CSMA: A survey. In *ICCS*, pages 199–204, 2012.

[103] Y. Zeng and X. Guo. Long short term memory based hardware prefetcher: A case study. In *Proceedings of the International Symposium on Memory Systems*, pages 305–311. ACM, 2017.

[104] C. Zhang, P. Patras, and H. Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials*, 2019.

[105] D. Zhao and Y. Wang. Sd-mac: Design and synthesis of a hardware-efficient collision-free qos-aware mac protocol for wireless network-on-chip. *IEEE Transactions on Computers*, 57(9):1230–1245, 2008.

[106] D. Zhao, Y. Wang, J. Li, and T. Kikkawa. Design of multi-channel wireless noc to improve on-chip communication capacity. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, pages 177–184. IEEE, 2011.

A Energy and Latency Overhead Characterization

It is widely acknowledged that deep learning inference has high latency and energy overheads. However, since NeuMAC needs to optimize the performance of a multicore CPU, it needs to operate at very small time scales. As a result, it is imperative that NeuMAC's inference step be efficient in time and energy. In this appendix, we characterize the overheads of running inference on NeuMAC's Deep RL agent.

Towards this end, we design an illustrative hardware macro for NeuMAC's neural accelerator (shown in Fig. 10). The trained quantized weights of NeuMAC's network are stored

in the 32 KB on-chip SRAM. The primary compute elements in the macro are the (i) 128 element 8-bit multiplier, that can perform 128 parallel multiplications of 8-bit numbers, (ii) followed by a 7-layer carry save adder tree, which can add up to 128 8-bit numbers. Thus, the multiplier block and adder tree block together can implement either one 128 dimensional dot product, or two 64 dimensional dot products in a one iteration. The ReLU non-linear activation is implemented using comparators, which finally writes the result into an output buffer. It is important to note that this hardware macro is significantly simpler than a full scale neural network accelerator, such as [53].

Next, we elaborate on the pipeline for computing one inference step on NeuMAC's RL agent. Note that computing the value of one element in the first hidden layer of NeuMAC's neural network requires one 64 dimensional dot product[3]. Therefore, computing the values of all elements in the first hidden layer requires a total of 128 counts of 64 dimensional dot products. Similarly, computing the values at the second hidden layer requires 128 counts of 128 dimensional dot products, and computing the final layer requires 64 counts of 128 dimensional dot products. Hence, to compute one inference step in NeuMAC's deep network, we need to perform a total of 192 counts of 128-element dot products, and 128 counts of 64-element dot products. Further, since we can implement two 64-element dot products in parallel, one inference step requires an equivalent of 256 counts of 128 dimensional dot products to compute the output. Using this above macro design along with conservative and widely accepted hardware estimates, we next show that the design of NeuMAC's neural network architecture adds only marginal overheads, allowing it to operate under the resource constrained setting of a wireless NoC.

Latency Overhead: Here we estimate the latency of computing one inference step on NeuMAC's RL agent. The memory array is organized as 16 blocks of 64 by 256 memory elements, making a total of 32 KB storage. For 45nm technology, read access time from such memory sizes can be conservatively estimated to be around 2 ns [95]. Similarly, a 32-dimensional dot product can be computed within 2 ns [44]. Hence, we pipeline the data flow in three stages, first after the memory read, second after adding the outputs of 32 multipliers, and third at the output of the comparator bank. Hence, each stage has a maximum latency of 2 ns. As a result of such pipelining, one 128 element dot product is computed every 2 ns, that is, every 2 clock cycles[4]. As noted previously, one inference step requires 256 counts of 128 dimensional dot products. Hence, the total latency for one inference step is $256 \times 2 = 512$ *ns* (512 clock cycles). This inference latency of 512 cycles results in a small overhead of less than 6% per time step in our RL formulation. One point to note is that, the final deep network output is quantized to 8 bits. Hence, the sigmoid filter after the last layer can be implemented via a 256 element look-up table at a negligible latency overhead.

Energy Overhead: Next, we estimate energy consumption of the hardware macro. We use the energy values from the widely-cited paper [48], which approximately characterizes energy consumption of various compute elements and memory accesses. The dominant energy consumption steps are the reads from the memory array and the computations on the MAC (Multiply-ACcumulate) unit. From [48], 8 bit multiplies consume 0.2 pJ, and 8-bit additions consume 0.03 pJ. One 128 dimensional dot product on the MAC unit involves 128 multiplications and 127 additions. Thus the total energy comes to 29.41 pJ. Memory reads of 64 bits from 2 KB memory blocks requires 5 pJ. Thus, the 128 bit memory reads for each dot product requires 10 pJ. As a result, one 128 element dot product on the hardware accelerator requires 39.41 pJ, and with 256 counts, the energy consumed for a single inference step is 10088.96 pJ. Given that we require one inference every 10,000 ns, the neural accelerator consumes approximately only 1 mW of power on average. In comparison, a single transceiver on the multicore consumes 16 mW [38]. Lastly, note that the numbers in [48] are at 45 nm technology, so 1 mW is a conservative estimate.

Area Overhead: Lastly, the area overhead of the hardware macro is small. Since area is dominated by memory, the 32 KB of SRAM and few registers in the hardware accelerator impose a small overhead in comparison to the 512 KB of cache memory at each of the 64 cores. Thus we envision that such a hardware macro can reside on the same die and share the same clock as the multicore processor.

Thus, even a simple accelerator like the one demonstrated in Fig. 10 can enable NeuMAC's agent to operate under the resource constrained setting of a wireless NoC. Note that we do not employ any other advanced hardware optimization techniques and rely on reported hardware numbers that are widely accepted rather than the state-of-the-art today.

[3] Although NeuMAC's input has 65 elements, for simplicity sake we perform calculations with 64 element input.

[4] Our CPU clock is 1 GHz.