2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2235-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/HPCA51647.2021.00066

Adapt-NoC: A Flexible Network-on-Chip Design for Heterogeneous Manycore Architectures

Hao Zheng, Ke Wang, and Ahmed Louri
Department of Electrical and Computer Engineering, George Washington University, Washington, D.C. 20052
Email: {haozheng, cory, louri}@gwu.edu

***Abstract*—The increased computational capability in heterogeneous manycore architectures facilitates the concurrent execution of many applications. This requires, among other things, a flexible, high-performance, and energy-efficient communication fabric capable of handling a variety of traffic patterns needed for running multiple applications at the same time. Such stringent requirements are posing a major challenge for current Network-on-Chips (NoCs) design. In this paper, we propose Adapt-NoC, a flexible NoC architecture, along with a reinforcement learning (RL)-based control policy, that can provide efficient communication support for concurrent application execution. Adapt-NoC can dynamically allocate several disjoint regions of the NoC, called subNoCs, with different sizes and locations for the concurrently running applications. Each of the dynamically-allocated subNoCs is capable of adapting to a given topology such as a mesh, cmesh, torus, or tree thus tailoring the topology to satisfy application's needs in terms of performance and power consumption. Moreover, we explore the use of RL to design an efficient control policy which optimizes the subNoC topology selection for a given application. As such, Adapt-NoC can not only provide several topology choices for concurrently running applications, but can also optimize the selection of the most suitable topology for a given application with the aim of improving performance and energy efficiency. We evaluate Adapt-NoC using both GPU and CPU benchmark suites. Simulation results show that the proposed Adapt-NoC can achieve up to 34% latency reduction, 10% overall execution time reduction and 53% NoC energy-efficiency improvement when compared to prior work.**

***Index Terms*—Network-on-Chips (NoCs), Reinforcement Learning, Reconfigurable Topology, Flexible NoC Designs.**

I. Introduction

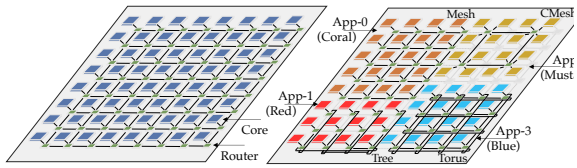
Modern heterogeneous manycore architectures are comprised of a large collection of computing resources such as CPUs, GPUs, and accelerators. While the increased computational capability and diversity facilitate the concurrent execution of multiple applications, it puts a large burden on the on-chip communication fabric (or the Network-on-Chip (NoC)). The NoC for heterogeneous architectures should provide flexible connectivity and adequate support for many traffic patterns generated by the currently running applications, all with high-performance and energy efficiency. Static NoCs are often optimized for a subset of applications and are thus inefficient in satisfying the various communication requirements of different applications running simultaneously. The mismatch between various communication demands and restricted NoC flexibility inevitably confines the communication performance and energy efficiency.

There is a large body of work [1]–[4] on flexible NoC design. Prior work either places express links to bridge long-distance routers [1] or fully customizes network connections [2]–[4] based on static communication task graphs. However, the flexibility of such schemes is still restricted to the fixed application mapping and behavior, and thus they have limited applicability in modern manycore architectures where multiple applications are scheduled to run concurrently. These applications are often dynamically allocated into different regions of compute and memory resources [5]–[7], leading to erratic application mapping and diverse regional communication behavior.

Moreover, running multiple applications simultaneously has been shown to cause inter-application interference - where applications compete for access to the shared resources. Significant research [8]–[15] has recently been done to avoid such interference and the performance degradation resulting from it. In NoCs, prior work either separates [11], [12] or dynamically allocates the shared NoC resources [13]–[15] to different types of communication traffic. While these techniques can reduce the traffic interference, they are inherently unable to fundamentally resolve the crux of the problem - the inadequate communication fabric.

Designing such an adaptable NoC architecture is challenging due to the fact that it needs to efficiently support different access patterns (e.g., one-to-many, many-to-one, all-to-all, etc.), performance and power requirements, and the dynamic interactions and behavior of the simultaneous execution of many applications. The problem is further exacerbated by the dynamically sized and allocated application mapping which leads to a significant increase in on-chip communication requirements. To this end, we propose **Adapt-NoC**, an application-aware flexible NoC design, along with a reinforcement learning (RL)-based control policy, that can support the diverse communication demands required by concurrently executing applications. Specifically, Adapt-NoC can dynamically allocate several disjoint regions of the NoC, called subNoCs, with different sizes and locations for various running applications. Each of the dynamically-allocated subNoC is capable of supporting a given topology such as a mesh, cmesh, torus, or tree, and thus satisfying different types of communication in terms of latency, bandwidth, power, and traffic patterns. In addition to the architectural design, we explore the use of RL to automate a control policy that can dynamically select the most suitable subNoC topology for a given running application with the goal of maximizing performance and energy-efficiency. The specific contributions

978-1-6654-2235-2/21/$31.00 ©2021 IEEE
DOI 10.1109/HPCA51647.2021.00066



(a) 8×8 Mesh-based Manycore Architecture (b) 8×8 Adapt-NoC

Fig. 1. (a) 8×8 mesh-based manycore architecture, (b) 8×8 Adap where four applications are running simultaneously.

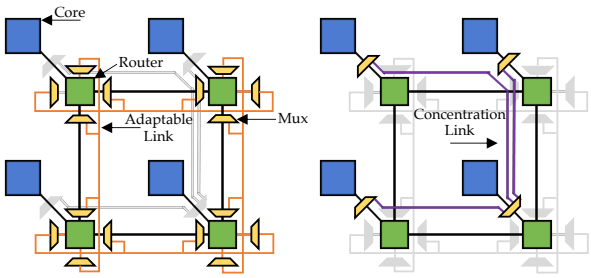
of this paper are as follows:

- **Adapt-NoC architecture:** We propose a new NoC a tecture composed of (1) adaptable routers, (2) adap links, and (3) concentration links that can dynam reconfigure any subNoC into a popular topology su a mesh, a cmesh, a torus, and a tree. While we consider these four topologies in this paper (due to limitation), the proposed design methodology allow any other topology to be deployed. The adaptable can be segmented into several independent forwar reverse links to connect multiple non-adjacent ro This provides several communication benefits incl router bypassing. The concentration links allow fo creasing router radix at runtime thus improving net diameter and consequently latency.
- **RL-based Control Policy:** We propose a per sub based RL control policy that can dynamically sele energy and performance efficient subNoC topology given running application with different size, location, and communication demands. The combined proposed hardware and control policy allow for the dynamic provisioning of multiple optimized topologies in various parts of the network architecture satisfying the communication demands of the multiple concurrently running applications.

We evaluate the proposed Adapt-NoC using full system simulation with both Parsec and Rodinia benchmark suites. Our simulation results show that Adapt-NoC can achieve up to 34% latency reduction, 10% overall execution time reduction, and 53% energy-efficiency improvement, as compared to prior work [1], [15], [16].

II. Designing Adapt-NoC

The goal of Adapt-NoC is two-fold: (1) provide several topology choices for concurrent applications at runtime, and (2) optimize the mapping of a given application to a selected topology with the aim of significantly improving performance and energy efficiency. For example, as compared to a conventional NoC fabric (Fig. 1(a)), Adapt-NoC is able to provide each of the running applications (applications 0 - 3 shown with different colors), with a separate subNoC, that is of different size, location, and topology as shown in Fig. 1(b). Consequently, the NoC fabric is dynamically divided into several simultaneous subNoC topologies, where each subNoC is optimized for a given application. In what follows we provide details of the architecture.



(a) 2×2 SubNoC connected by adaptable Links, and (b) concentration links

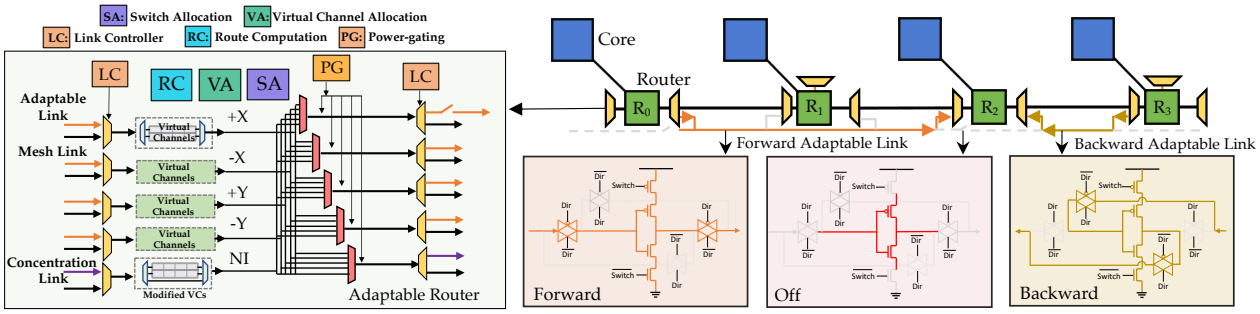
Fig. 2. 2×2 subNoC connected by (a) adaptable links and (b) concentration links.

A. Adapt-NoC Architecture

Adapt-NoC architecture consists of four novel elements (and associated logic), namely adaptable routers, adaptable links, concentration links, and an RL-based control policy. The effective utilization of these components results in increased functionality and a number of network features such as bypassing and concentration that can be used to design diverse subNoC topologies, which can provide better communication support for concurrently running applications. For example, the adaptable links connecting adaptable routers vertically and horizontally can provide express links (bypassing) between routers as shown in Fig. 2(a). The concentration links provide router-core connections to connect multiple cores to a single router (concentration) as shown in Fig. 2(b). We describe the details of these designs in the following subsections.

1) Adaptable Router: As each subNoC requires different topology, the adaptable router is designed to provide a variable radix and connection. However, deploying a high-radix router requires additional input ports which leads to increased area overhead and sophisticated router arbitration (timing overhead). To solve this critical issue, we propose an adaptable router to provide desired router radix and connection but without additional input ports.

The adaptable router consists of a set of muxes connected to both input and output ports as shown in Fig. 3(a). These muxes, together with the adaptable and concentration links can result in different network connections without increasing router complexity. Specifically, the muxes of the input ports at +x, -x, +y, and -y directions, can selectively connect to mesh links or adaptable links to bridge long-distance routers (bypassing) without implementing additional dedicated input ports. The mux connected with the injection port (network interface (NI)) provides core-router connections, creating external concentration [17] without requiring additional input and crossbar ports. To further reduce buffer delays at the injection port, we add an additional bypass link at the virtual channels of input port at the NI. In addition to these designs, we implement a power-gating controller to power off unused ports and the crossbar thus reducing static power consumption. Moreover, the link controller is deployed to control the link directions and turn on/off the link connections of adaptable links (this is discussed in the next subsection). Routing computation (RC) unit includes a reconfigurable routing table to support different



(a) Microarchitecture of adaptable router, and (b) adaptable link design

Fig. 3. (a) Microarchitecture of adaptable router, and (b) adaptable link design.

routing algorithms.

2) Adaptable Link: As any region of the NoC can be reconfigured to a desired topology with different size, this requires much more connectivity between any pair of routers. For example, in an 8×8 Adapt-NoC, each row/column requires at least 28 (i.e. C_8^2) bi-directional links to cover all possible subNoC connections which could exhaust limited wiring resources. To solve this issue, an adaptable link design is proposed to fully utilize the on-chip wiring resources by providing two functionalities: link segmentation and link reversal. Link segmentation dynamically adjusts the link length to fit the transmission distance, and link reversal dynamically allocates the link resources to different directions without requiring additional wires.

To achieve both functionalities, we implement a bi-directional adaptable link across each row and column. Each adaptable link consists of a number of quad-state repeaters [18] which can disable signal propagation (link segmentation) and switch the signal propagation directions (link reversal). For example, Fig. 3(b) shows an adaptable link connecting four routers, R_0-R_3, horizontally. To provide two disjoint inter-router connections (R_0-R_2, R_2-R_3), the quad-state repeater at R_2 should be switched off by disconnecting the transistors from GND and V_{dd}. For each of the segmented links, the signal 'Dir' controls the direction of signal propagation. When 'Dir' signal is disabled, the signal is propagated in a forward direction between R_0 and R_2, and similarly, the signal is propagated in a backward direction between R_2 and R_3 when 'Dir' signal is enabled.

B. SubNoC Construction

In this section, we illustrate the methodology to form a subNoC. Without loss of generality, we select cmesh, torus and tree as case studies in a 4×4 allocated subNoC. These topologies are of the most popular NoC topologies and cover diverse network latency, bandwidth, and traffic pattern. It should be noted that the methodology is applicable for any topology.

1) Cmesh: Cmesh (Concentrated Mesh) [19] is to connect multiple nodes (e.g., cores, caches, and memory controllers) to a single router. As a result, it can reduce the number of routers deployed within the network and shorten the network diameter. However, Cmesh increases router complexity which results in additional area and timing overhead due to the increased router radix.

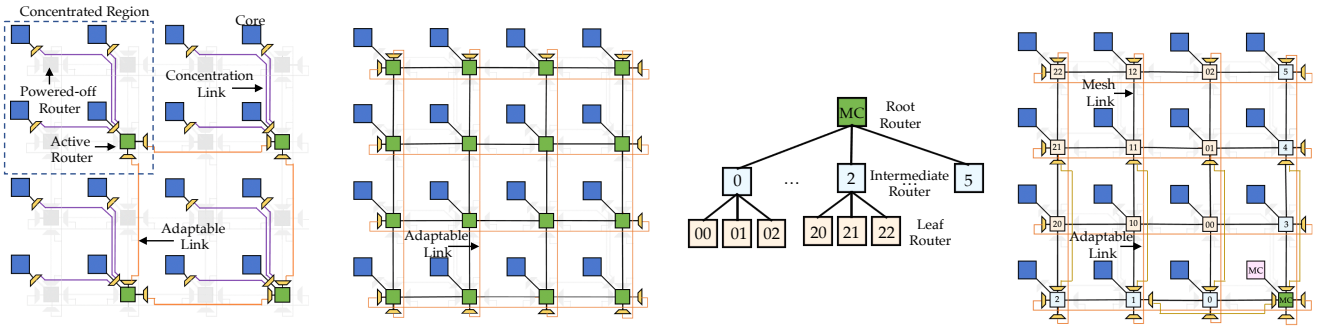
The Adapt-NoC realizes cmesh topology by using external concentration [17] with the aim of reducing router complexity as shown in Fig. 4 (a). Cores within each concentrated region are directly connected to the same router through the concentration links. The idle routers are powered-off to reduce the static power consumption, but the powered-off router could disconnect the network. To keep the network connectivity, the adaptable links connect non-adjacent routers vertically and horizontally. The composed cmesh topology can benefit those application phases with sparse communication traffic.

2) Torus: Bypassing is to place express links to bridge long-distance routers or skip routers, and it allows packets to bypass the intermediate routers and reduce latency. In addition to the reduced network latency, the bypass links across the network also increase the network bi-section bandwidth. The torus is an example of placing express links horizontally and vertically to connect peripheral routers.

In Adapt-NoC, the subNoCs can be configured in any region of the network. The differently-sized subNoCs require various lengths of wrap-around links to connect peripheral routers for composing the torus. To provide the desired wrap-around links, the adaptable links are segmented to the desired length and connect to the peripheral routers of the configured subNoC as shown in Fig. 4 (b). These wrap-around links not only reduce the network diameter but also increase network bi-section bandwidth, benefiting those application phases with high demand for both latency and bandwidth.

3) Tree: The heavy reply traffic from the memory controllers (i.e. one-to-many traffic) has been identified as a bottleneck of on-chip communication in throughput-oriented processors [20], [21]. The intensive reply traffic causes packet congestion at the injection port, and it increases queuing latency [22]. Such an undesirable queuing latency mainly results from insufficient injection bandwidth and poor load balancing in grid-like topologies such as the mesh. To solve this issue, the tree topology has shown to be a good candidate, as it provides high injection bandwidth at the root node and balanced load distribution. However, the deployment of tree topology in on-chip networks faces a major challenge: limited on-chip wiring resources.

The Adapt-NoC solves the wiring issue by reversing one



(a) Cmesh in a 4×4 SubNoC (b) Torus in a 4×4 SubNoC (c) Logic diagram of the tree connection (d) Tree in a 4×4 SubNoC

Fig. 4. SubNoC topologies: (a) Cmesh, (b) torus, (c) logic diagram of tree connection, and (d) tree in a 4×4 subNoC.

bi-directional adaptable links to two uni-directional adaptable links, and it relies on the fact that the reply traffic moves towards the same direction from the same destination as shown in Fig. 4 (c). With the sufficient adaptable links, we maximize the fanout of the root router (i.e. MC) together with the bypass link, to provide sufficient injection bandwidth, so that packets are injected into the input buffers of intermediate routers without buffering at the injection router. Furthermore, the root and intermediate routers are connected to downstream routers, vertically and horizontally to evenly distribute reply traffic. Fig. 4 (d) shows the physical connection of a composed tree topology. The root, intermediate and leaf routers are fully connected by a collection of links (mesh and adaptable links) horizontally and vertically. The adaptable links with different directions are depicted in different colors. Such link connections (Fig. 4 (d)) are designed to couple with dimensional-ordered routing (generic routing algorithm) which enforces turn restriction. Consequently, the reply traffic from the MC can be delivered to all routers within two hops in the illustrated example. We note that the request traffic still goes through the mesh topology. The composed tree topology can benefit those applications that demand intensive off-chip memory accesses.

Tree Scalability: While the example depicted in Fig. 4 (d) shows a fully connected tree topology, the tree suffers from scalability issues with an increased subNoC size. In the case of large subNoC size, we follow the same design principle: maximize injection bandwidth and evenly distribute reply packets. As a result, we still maximize the fanout of root router, but connect the root and intermediate routers with their downstream routers at an evenly-spaced distance in each row/column.

4) Possible subNoC topologies: The subNoC formation methodology can be generalized to more topologies, including different degrees of network concentration, express link placements, and combined topologies. For example, the torus (Fig. 4 (b)) and tree (Fig. 4 (d)) could be combined together to simultaneously optimize both request and reply networks for memory-intensive applications. Moreover, the wrap-around torus links can be segmented to several short express links to bypass routers. While a number of topologies can be generalized, in this work, we concentrate on four popular topologies - mesh, cmesh, torus, and tree - to demonstrate the performance and energy benefits of simultaneously deploying different subNoC topologies.

C. SubNoC Management

In this section, we discuss the rules for simultaneously managing several subNoCs, including the dynamic subNoC allocation, memory controller sharing design, and deadlock avoidance.

1) Dynamic SubNoC Allocation and Switching: The nature of dynamic subNoC allocation is to allocate a collection of cores, memory modules, routers, and links within a region of the manycore architecture. Prior research [7], [23], [24] has been proposed to efficiently allocate computing and memory resources within a region by modifying cache coloring, page replacement, and mapping policy, and thus the data is placed closer to the computation. As a result, the application can take full advantage of data locality [23], [24] and mitigate inter-application interference [7].

The remaining challenge for subNoC allocation is to timely reconfigure the subNoC topology and routing algorithm dynamically, in a deadlock-free manner [25], in response to the dynamic application mapping. The latency overhead of subNoC configuration mainly results from network stall and packet drainage for preventing unroutable packets and cyclic dependency caused by new and old routing algorithms [26], [27]. In our design, all topology choices adopt minimal, dimensional-ordering routing (e.g., XY) disallowing the same "U-turns". That is, the torus, tree, and cmesh topologies either add or remove routing paths to or from the mesh topology. We will use the following walk-through example to illustrate the process of subNoC reconfiguration. It should be noted that each routing algorithm in our design is deadlock-free, and the routing algorithm of mesh topology will not form any dependency cycles when combined with other routing algorithms.

Walk-through Example: To configure a $N \times M$ subNoC, it first takes $(M+N-2)\times(T_r+T_l)$ cycles to notify all the routers within the subNoC to initiate the dynamic reconfiguration, where T_r is the hop latency, T_l is the link latency. Upon receiving the notification, each router asynchronously takes T_s to set up the mesh connectivity and its routing algorithm R_{mesh}, if necessary. After the R_{mesh} is added, the router disables old routing algorithm R_{old} following its order of the

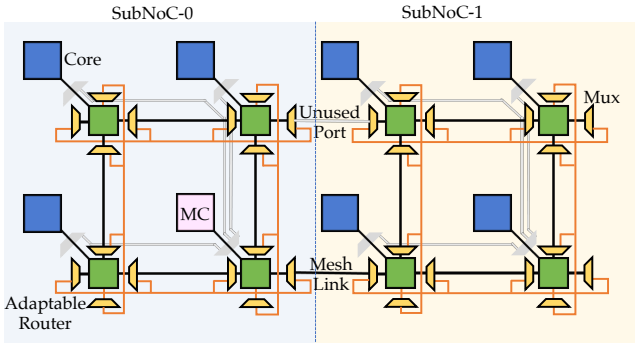


Fig. 5. The memory controller sharing design

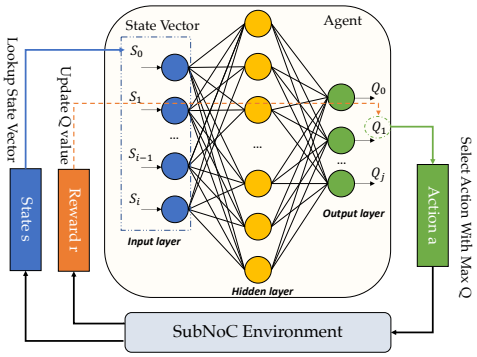


Fig. 6. The agent-environment interaction in Deep Q-networks.

channel dependency graph, when no packets need the routing choices from R_{old}. This maintains the network connectivity and avoids unroutable packets. Once all routers are done with removing R_{old} (and their old link configuration), each router starts setting up the R_{new} (and its new link configuration). The last step is to remove R_{mesh} when R_{new} is set, if necessary. Note that when R_{mesh} is a subset of the routing algorithms of torus and tree, R_{mesh} is unnecessary to be removed. These operations satisfy all the sufficient conditions for deadlock-free reconfiguration according to Lysne's methodology [28], and avoid the network stall and package drainage.

In this paper, we use regular topologies and turn-restricted routing algorithms to illustrate the deadlock-free dynamic subNoC reconfiguration. However, the proposed design can support irregular topologies with deadlock-free prevention techniques such as Lysne's methodology [28] and Double Scheme [29].

2) Memory Controller Sharing Design: The memory controllers (MCs) are responsible for managing off-chip memory accesses and are often distributed across the network. In Adapt-NoC, we implement one MC to each 2×4 subNoC in an 8×8 NoC, and therefore it guarantees that eight applications all have their independent MC access. However, memory-intensive applications place additional demands on the memory bandwidth and thus require concurrent access to multiple MCs. Therefore, we propose a memory sharing design that allows the applications to access the MCs of their adjacent subNoCs. Such MC accesses rely on the connections between the peripheral routers of the adjacent subNoCs. For example, when the application located in subNoC-1 demands additional memory bandwidth and requires accessing MC of subNoC-0, the unused mesh link will connect the peripheral routers of both subNoCs which can enable the external MC access, as shown in Fig. 5.

Memory Controller Scalability: The growth of MC number is often lagging behind the increase in core count, and as a result, each MC has to be shared among a larger number of cores. This could increase subNoC size and thus restrict the maximum number of co-running applications. To solve this issue, we can use the proposed MC sharing design to connect multiple subNoCs to one MC, and as a result, it can support the simultaneous execution of up to 32 applications in a 256-core system with 8 memory controllers. Furthermore, the Adapt-NoC could simultaneously accommodate more applications if each subNoC allows the mapping of multiple applications. This shows the scalability of Adapt-NoC for handling the limited number of MCs for hundreds of cores projected in the future.

3) Deadlock Avoidance: Network deadlock can occur due to protocol dependency or circular channel dependency. Specifically, the protocol dependency is eliminated by separating the request and reply packets to different virtual networks so that the protocol deadlock is avoided. To prevent the circular channel dependency, we use dimensional-ordered routing to restrict all turns from y-dimension to x-dimension. The turn restriction routing is effective for mesh and cmesh, however it is not suitable for torus as it can create circular channel dependency [22], [30]–[32]. We use the simple yet effective dateline [22] to avoid such circular channel dependency. It should be noted that the prerequisite of forming circular channel dependency is that a given router is connected to at least two other routers. When a given subNoC requires remote memory access from its adjacent subNoCs, only one of the routers is allowed to connect with the MC outside the subNoC. This precludes the formation of cyclic channel dependency within any subNoC. Additionally, some routing algorithms such as static bubble [33] can be implemented to prevent deadlock in irregular topologies.

To configure irregular topologies or adaptive routing, significant amount of techniques have been proposed to prevent the reconfiguration-induced deadlock. These techniques can also be deployed to further enhance Adapt-NoC performance.

III. Reinforcement Learning-based Control Policy

A. Reinforcement Learning Model

We present a reinforcement learning (RL)-based control policy for optimizing subNoC selection with the aim of improving subNoC energy-efficiency and performance for any given application. Specifically, we implement RL controllers in MCs for subNoC selection. The RL controller interacts with manycore system, learns from the past system performance, and selects a number of subNoC topologies with the objective of maximizing the long-term total rewards (e.g., performance and energy efficiency). The Q-learning, an RL algorithm [34],

is applied to find an optimal action in a given state based on a table-based Q-table.

The Q-table records the observed system environment as a state vector s, and maps long-term total rewards of each action a to the observed state s. Therefore, each Q-table entry, Q(s,a), represents the long-term total rewards of action a in the state s. At each time step, the Q-learning algorithm often chooses the action with the highest reward from the Q-table based on the observed state. After taking the selected action, the Q-table entry Q(s,a) is updated using Equation 1 based on the action a, reward r, and new state s'.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma maxQ(s',a) - Q(s,a)] \quad (1)$$

where α is the learning rate, γ is the discount factor, and $maxQ(s',a)$ is the maximum Q value over all possible actions in state s'.

As the Q-table needs to map each action to all states, the Q-table size is often exponentially increased due to the large state space. This could lead to slow policy convergence and prohibitive hardware cost. To eliminate such negative effects, deep Q-network (DQN) method [35] is used to approximate the large Q-table by using a neural network. The DQN takes the state $state$ as the input of neural network and calculates the Q-value of each action as the output, as illustrated in Figure 6.

B. Action

In RL, action is what the agent can select to optimize the system environment in each time epoch. As designing NoC topology involves many trade-offs between different objectives such as latency, bandwidth, power consumption, and others, each topology is often optimized for a given objective but at the expense of others. In order to take the full advantages of these topologies, the action space of this work includes four topologies, mesh, cmesh, torus, and tree. In each time epoch (50K cycles), the agent can select one of these subNoC topologies to provide the desired subNoC connections, ensuring adequate communication support needed by the various application phases.

C. State

TABLE I
STATE ATTRIBUTES OF RL

Category	State Attributes
Instruction and Cache Related Metrics	Number of L1D cache miss
	Number of L1I cache miss
	Number of L2 cache miss
	Number of retired Instruction count
Network Related Metrics	Number of coherence packets
	Number of data packets
	Average router buffer utilization
	Average injection port buffer utilization
	Average router throughput
Topology Related Metrics	Current network topology
	Column size of the topology
	Row size of the topology

A well-modeled state space can provide a good indication of the system environment for decision making, and thus improve the predication accuracy of RL. We study a wide range of system parameters and select the following metrics listed in Table I. The combined use of presented parameters provide the optimal performance and energy efficiency.

- *Instruction and Cache Related Metrics:* Instruction count and cache activities are often used to indicate the computational intensity [14], [36]. Consequently, in this work, the number of retired instruction and L1 cache misses are included to reflect the runtime computation behavior. In addition, the number of L2 cache misses are considered to reflect the dynamic communication behavior, as the shared L2 caches are distributed in the manycore architectures and communicate through the NoC.
- *Network Related Metrics:* The network related metrics are included to signify different types of on-chip communication. For example, the multi-threaded applications often rely on the synchronization primitives to enforce the exclusive data access between threads, and these synchronization primitives result in different coherence traffic. As such, the message types (e.g., data and coherence) can provide indications of varying application phases. In addition, the buffer utilization is the most intuitive indication of NoC traffic load. In order to precisely measure the queuing latency, the buffer utilization of injection port is monitored individually.
- *Topology Related Metrics:* As the topology selection not only considers the latency and throughput demands but also the network size, we take the topology information (e.g., row and column size) into consideration. The current topology refers to the topology that is currently running.

D. Reward

As the agent selects actions with the goal of maximizing the long-term reward, the reward instructs the action selection and ultimately determines the system performance. In this paper, with the goal of improving overall network performance and energy-efficiency, we include network latency, queuing latency, and power in the reward function as shown in Equation 2.

$$Reward = -power \times (T_{network} + T_{queuing}) \quad (2)$$

Power refers to the average static and dynamic power measured in each allocated subNoC. The network latency ($T_{network}$) is the time that a packet traverses on the NoC, and the queuing latency ($T_{queuing}$) is the time that a packet waits at the network interface.

E. Training

The approximation of Q-value using non-linear functions can lead to unstable RL training [35]. To stabilize the RL training, target network [35] and experience replay [37] are two common approaches to eliminate the correlation between training samples. However, the target network and experience replay require additional hardware to store a significant amount of training samples which leads to prohibitive hardware costs. To eliminate the prohibitive hardware costs, we use

an off-line training for this work. As a result, the trained model only requires storing a number of weights instead of additional neural network and training samples. We use different parts of applications for training and testing. To improve the robustness of off-line training, the training set includes a wide range of application phases (injection rate, cache miss rate, and traffic load), and the model is trained under different network configurations (e.g., 2×4, 4×4, 4×6, 4×8, and 8×8).

The proposed DQN model consists of one input layer, two hidden layers, and one output layer. The input layer consists of 12 neurons due to the state vector size. Each of the input value is normalized within the range of (0,1) due to the linear region of activation function. Each hidden layer consists of 15 Relu neurons. We have tested hidden layer size from 10-50 neurons, of which the setup of 15 neurons has the optimal performance in terms of latency and energy. The output layer consists of 4 neurons that represent the Q-values of topologies. For the off-line training, two networks, called prediction and target networks, together with experience reply buffer are used to ensure the faster convergence of neural network by eliminating the data correlation. Specifically, the prediction network is used to make the decision about which topology is applied, and the target network is used to update the weights in every iteration. The experience reply buffer consists of 1000 entries, each of which can store a state vector. The target network randomly selects one state vector from the replay buffer instead of using the recent observed state vector, thus eliminating the data correlation. The minibatch gradient descent is used to update the target network. We set the minibatch size of 100 because the experience replay has provided sufficient stability. The target network updates the weights to the target network every 168 iterations. Moreover, we set the learning rate of neural network as 0.0001, as the learning accuracy is more important than speed in the off-line training.

IV. Evaluation

A. Simulation Setup

TABLE II
Benchmark Applications

Categories	Applications
CPU	Blackscholes (BS), Swaptions (SW), X264, Ferret (FR), Bodytrack (BT), Canneal (CA), Fluidanimate (FL)
GPU	Kmeans (KM), Back-propagation (BP), Heart-Wall (HW), Guassian (GA), Breath-First-Search (BFS), Needleman-Wunsch (NW), HotSpot (HS),

We evaluate the proposed architecture under full system simulation with the combined use of architecture-level and circuit-level simulators. The cycle-accurate gem5-GPU simulator [38], and GARNET [39] are used for a detailed timing simulation of the memory and on-chip network. We model the power of all components (including router, links, muxes, RL modules) with Synopsys Design Compiler using 45 nm. To accurately calculate the dynamic power, we feed the power parameters captured by Synopsys to DSENT, which calculates the average dynamic power by profiling the number of buffer writes, crossbar, VA/SA activities, and RL calculations [40]. All the timing overhead (detailed in Section V-B), including subNoC switching latency, mux delays, and link latency, are considered in the simulation. The RL model is implemented with GARNET simulator to provide closed-loop simulation. Since the learning rate α, the discount rate γ, and exploration rate ϵ are hyper-parameters, we set the learning rate α to 0.1, discount factor γ to 0.9, and exploration rate ϵ to 0.05. The studies of these parameters are detailed in Section V-C.

To examine the diverse application behavior, we evaluate the performance of our proposed design using both the Parsec [41] and Rodinia [42] applications listed in Table II, in an 8×8 heterogeneous system. We assume that three applications are dynamically mapped to different regions of the heterogeneous manycore system. The Rodinia application is mapped to a region with the mix of 4 CPUs, 4 MCs, and 24 GPUs, where each 2×4 subNoC consists of 1 CPU, 1 MC, and 6 GPUS. Each GPU core contains 8-wide SIMD lanes and is equipped with a 64KB private L1 cache and scratch memory. Parsec applications are mapped to 28 CPUs and 4 MCs, where each 2×4 subNoC consists of 7 CPUs and 1 MC. Each CPU core also contains private a 64KB L1 instruction/data cache, and 1 MB shared L2 cache. Each core or MC is attached to a router. All the evaluated designs adopt virtual cut-through buffer organization. The number of virtual channels (VCs) of all the designs are set differently to keep their area consistent. Specifically, we set 3 VCs per virtual network, 4 flits per VC for baseline, OSCAR, and shortcut. To keep area consistent with baseline topology, we set 2 VCs per virtual network and 4 flits per VC in Adapt-NoC. In Flattened Butterfly, we set 4 VCs per virtual network and 4 flits per VC. As the link delay is proportional to wire length, we set 1-cycle delay per 4mm for links placed on high metal layers (e.g., bypassing links). The mesh link latency T_l is set as 1 cycle. The link width is set as 256-bit. The connection setup time (T_s) refers to the time for setting up router components such as routing table and link connection, which is set to 14 cycles [43]. During T_s, the routing table will not be available. It should be noted that the network would not be entirely halted for packet injection and drainage which was discussed in Section II-C. We set hop latency T_r as 3 cycles in Flattened Butterfly, and 2 cycles in the other designs.

We examine the performance of Adapt-NoC by comparing it to the following prior work:

1) **Baseline:** We use a mesh topology as the baseline design.
2) **OSCAR [15]:** We deploy the dynamic VC allocation technique proposed in OSCAR to avoid traffic interference among applications in heterogeneous manycore architectures. We note that the last-level caches used in this paper are SRAM instead of stt-RAM in OSCAR.
3) **Shortcut [1]:** We use a reconfigurable NoC design, where long-distance routers can be connected by additional express links.
4) **Flattened Butterfly (FTBY) [16]:** Flattened Butterfly is

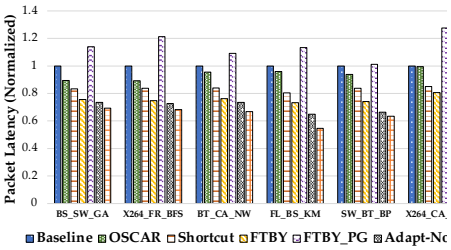


Fig. 7. The latency analysis of mixed workload, no

a high-radix on-chip topology, where tration and bypass are used to reduce

5) **FTBY_PG:** We deploy conventiona gating in Flattened Butterfly to re consumption.
6) **Adapt-NoC-noRL:** SubNoC topolo selected to provide the optimal perform topology choices.
7) **Adapt-NoC:** RL-based control policy i namic subNoC selection.

V. Results

A. Performance Analysis

1) Packet Latency Analysis: Fig. 7 shows the analysis of packet latency which includes both network and queuing latency. Adapt-NoC has lower packet latency over other designs, and it reduces the packet latency of the mixed workload by 34%, 30%, 22%, 15%, 44%, and 9% when compared to the baseline, OSCAR, shortcut, FTBY, FTBY-PG and Adapt-NoC-noRL, respectively. This is because the Adapt-NoC can dynamically provide each running application with a suitable subNoC topology which reduces the hop count and queuing latency. In what follows we provide the detailed analysis of hop count and queuing latency.

Fig. 8 shows the hop count analysis of CPU applications. As expected, the Adapt-NoC achieves 41% of hop count reduction as compared to the baseline and OSCAR, as both of the baseline and OSCAR are deployed with the mesh topology which has higher network diameter than other topologies. In addition, the Adapt-NoC can outperform the shortcut with an average of 31% hop count reduction. This is because the Adapt-NoC can provide multiple ways of reducing hop count such as bypassing and concentration, but the shortcut can only provide a limited number of express links. The Adapt-NoC slightly increases the hop count by 9% as compared to the FTBY. This is because the fully-connected routers in each dimension guarantee the minimum hop count for the FTBY, but, at the same time, the rich connectivity increases the router complexity of FTBY which leads to increased hop latency. As a result, the increased hop latency offsets the latency improvements in FTBY. Besides the hop count and latency, Adapt-NoC outperforms FTBY-PG significantly because the conventional power-gating technique adds substantial latency to resume router's activity from the powered-off state. The Adapt-NoC further reduces the hop count by 6% as compared to Adapt-NoC-noRL.

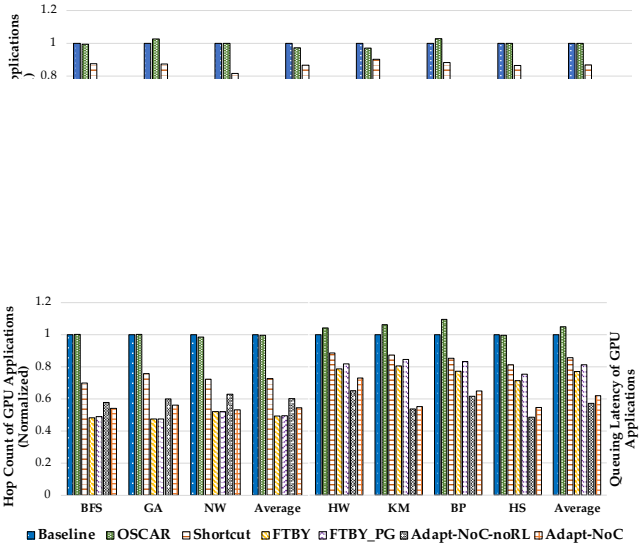


Fig. 9. The hop count and queuing latency analysis of GPU applications, normalized to baseline.

We further present the hop count and queuing latency analysis for GPU applications in Fig 9. Overall, the Adapt-NoC reduces the hop count of GPU applications by 46%, 45%, 18%, and 11% when compared to the baseline, OSCAR, shortcut, and Adapt-NoC-noRL, but it increases the hop count by 10% as compared to FTBY. For the queuing latency, the OSCAR slightly increases the queuing latency by 8% as compared to the baseline, because the dynamic VC allocation results in sub-optimal network utilization which leads to reduced network throughput. Besides OSCAR, the Adapt-NoC-noRL, Adapt-NoC, FTBY, and shortcut reduce the queuing latency by 43%, 39%, 24%, and 16% when compared to the baseline, as the additional path diversity (e.g., increased bi-section) can improve the poor load balance of meth topology. From the simulation result, we observed that the high-radix router used in FTBY can intensify the packet contention at the intermediate routers, as the number of buffers at each input port cannot proportionally scale to the increase in router radix which affects the network throughput. But the improvement in load balancing (e.g., increased bi-section) can still compensate the negative impacts of reduced buffer size, and it ultimately leads to the queuing latency reduction in FTBY. As compared to FTBY, the Adapt-NoC further reduces the queuing latency due to the following two designs. First, the bypass link deployed at the VCs of the injection port can promptly transmit the packets to downstream routers without buffer delays. Second, the tree deployed in the Adapt-NoC not only increases bi-section bandwidth but also avoids the packets contention caused by the insufficient buffers at the intermediate routers. It also should be noted that the FTBY has the same channel bandwidth as other works, as the wiring density of FTBY in an 8×8 NoC still fits the wiring budget. However, the channel bandwidth of FTBY has to be reduced when network size increases to 16×16, as the wiring density of FTBY increases quadratically with the network size. In this case, the queuing latency of FTBY

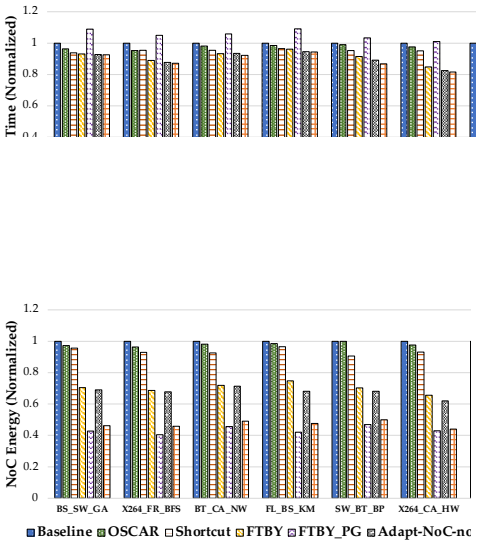
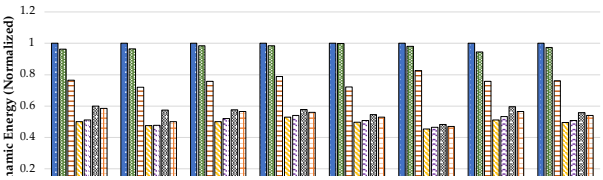


Fig. 11. The energy analysis of mixed workload, norm

would increase by 85% as compared to the as the reduced channel bandwidth significant serialization latency and thus intensifies the pa at the injection port. However, the Adapt-NoC this issue, as the Adapt-NoC only requires on in each row/column which is proportional to network size.

2) Execution Time Analysis: Fig. 10 shows the execution time analysis of Adapt-NoC. Overall, the Adapt-NoC has the shortest execution time over other designs. Specifically, the Adapt-NoC reduces the average execution time by 7% as compared to OSCAR, and this indicates that providing each running application with suitable communication support is a more efficient way to handle the traffic interference. The Adapt-NoC has 5% shorter execution time than the shortcut, as it can provide additional ways of reconfiguring subNoC topologies other than the long-distance express links. Additionally, the Adapt-NoC outperforms the FTBY with a 3% shorter execution time due to the simpler router complexity and better load balancing performance. Furthermore, the Adapt-NoC reduces 16% execution time of FTBY-PG. These advantages of the Adapt-NoC ultimately come to 10% average execution time reduction as compared to the baseline. The Adapt-NoC reduces the execution time by 1.5% as compared to Adapt-NoC-noRL.

3) Energy Analysis: Fig. 11 shows the energy analysis. Overall, the Adapt-NoC improves overall energy savings by 53%, 50%, 48%, 26%, and 23% and when compared to the baseline, OSCAR, shortcut, FTBY, and Adapt-NoC-noRL, respectively. The energy benefits of the Adapt-NoC are resulting from the reduced execution time, the reduced hop count, and the use of power-gating. Specifically, Fig. 12 shows the dynamic energy analysis, in which the Adapt-NoC reduces the dynamic energy by 46%, 43%, 30%, and 5% as compared to the baseline, OSCAR, shortcut, and Adapt-NoC-noRL on average. This is because the packets skip the intermediate routers and thus avoid significant amount of router activities



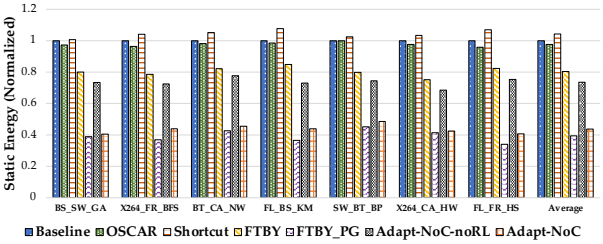


Fig. 13. The static energy analysis of mixed workload, normalized to baseline.

(e.g., buffer write, crossbar traverse), which leads to dynamic energy saving. As expected, the Adapt-NoC slightly increases the dynamic energy by 7% and 4% as compared to the FTBY and FTBY-PG due to the larger hop count. Fig. 13 shows the static energy analysis, in which the Adapt-NoC reduces the average static energy by 56%, 53%, 60%, 37%, and 30% as compared to the baseline, OSCAR, shortcut, FTBY, and Adapt-NoC-noRL, respectively. The static energy savings of the Adapt-NoC results from the reduced execution time and the use of power-gating. The Adapt-NoC is able to power off idle routers or ports of the peripheral routers. The substantial energy saving of Adapt-NoC can compensate the energy overhead caused by additional adaptable links (11.5 mW/link). We also compared the Adapt-NoC to FTBY-PG in which runtime power-gating technique is deployed. The static energy of Adapt-NoC is increased by 7% as compared to FTBY-PG, which leads to less 6% energy savings. However, considering the large latency overhead of FTBY-PG, the energy-delay product (energy $\times$ execution time) of Adapt-NoC would be 8% less than FRBY-PG.

B. Overhead Analysis

1) Area Overhead Analysis: We evaluate the hardware cost through Synopsis Design Compiler using 45 nm technology. The baseline router consists of a crossbar of 17806 um^2, a switch allocator of 4589 um^2, a virtual channel allocator of 1062 um^2, and buffers of 246472 um^2. For an 8×8 mesh topology, the overall NoC area is 17.27 mm^2. With additional ports at the peripheral routers, the Adapt-NoC requires an additional 1.46 mm^2 area. The neural network requires an arithmetic logic unit (e.g., multiplier, relu function) and storage. Since we only implement one RL controller in each 2 $\times$ 4 sub-NoCs, only 8 RL controllers are needed. The total area overhead of the RL controllers is 100232 um^2. The arbiter, the muxes, and additional links account for 107123 um^2 in total. To compensate for the additional area overhead, we reduce the number of buffers in the Adapt-NoC as compared to baseline.

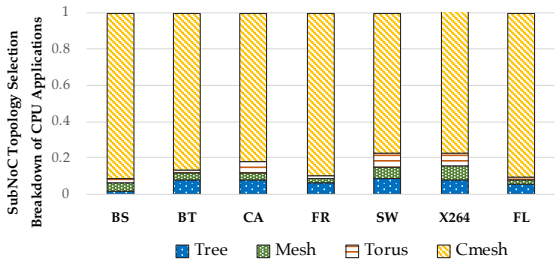


Fig. 14. Topology selection breakdown of CPU Applications.

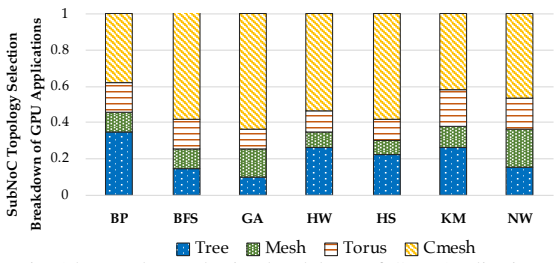


Fig. 15. Topology selection breakdown of GPU Applications.

As a result of fewer buffers, the Adapt-NoC would require 14% less area as compared to the baseline.

2) Wiring Constraints Analysis and Optimization: As the Adapt-NoC requires additional wiring resources (e.g., adaptable and concentration links), in this section, we examine and optimize the wiring resources to fit the wiring budget (or wiring density): the maximum number of wires can across a given area. This is often determined by two factors, namely, metal layer and tile size. While the number of metal layers increases with technology scaling to accommodate the higher routing demands [44], we use the Intel 45nm metal stack model [45] to examine the wiring density of the Adapt-NoC where nine layers of metal (M1-M9) are available. In high metal layers (M7-M8), the wire width is 280 nm, the wire height is 280 nm, and the wire pitch is 560 nm. In intermediate metal layers (M4-M6), the wire width is 140 nm, the wire height is 140 nm, and the wire pitch is 280 nm. Typically, half of the wiring resources are available for on-chip routing. For an 1 mm^2 tile [46], the high metal layers can provide two 256-bit bi-directional links per tile edge, and the intermediate metal layers can provide seven 256-bit bi-directional links per tile edge. The maximum number of adaptable links, concentration links, and mesh links required for each tile edge is four 256-bit bi-directional links, which is within the wiring budget.

3) Timing Analysis and Optimization: We present the detailed router and link timing analysis and optimization schemes in the following.

Router Timing: While we reduced the number of VCs to accommodate the additional logic delay of muxes, there are several alternative ways of hiding the logic delay in NoC routers, such as folding latency or time stealing [47]. We use the Synopsys Design Compiler to verify the critical delays of adaptable router. The simulation result shows that the route computation (RC), virtual channel allocation (VA), switch allocation (SA), and switch traversal (ST) of a conventional 5×5 router take 164 ps, 370 ps, 243 ps, and 256 ps, respectively. Each mux requires 102 ps critical delay. Therefore, we deploy an optimization to solve the potential timing issue. Specifically, we merge mux logic of input and output ports into RC and ST stages, respectively. As the router frequency is limited by the delay dominant router stage (VA), the delays of merged RC and ST (266 ps, 350 ps) are still shorter than that of the VA stage (370 ps). This satisfies the timing of the router, and thus does not affect the router frequency.

Link Timing: As the wire delay increases with the wire length, we implement the tri-state repeaters to keep the delay proportional to the wire length. In this paper, we use the 1.7 $\mu\Omega$*cm as the resistivity of copper metal layer, and 0.2 pF/mm as the wire capacitance [48]. The wire delay of high metal layers is 42 ps/mm, and the wire delay of intermediate metal layer is 200 ps/mm. Therefore, we place the long-distance adaptable links on the high metal layers so that the delay/mm is reduced due to greater thickness and width. In addition, since the reversed quad-state repeaters have increased critical latency (45 ps) due to additional critical delays of the transmission gates, we use the reversed adaptable links to connect to routers with a shorter distance to optimize the transmission latency. For fair comparison, we assume express links in all baselines are placed on high-metal layer.

RL Latency: In this work, we assume the minimal hardware resources (one adder and one multiplier). With this assumption, the DQN calculation is 486 nsec. However, this latency can be overlapped by the large reconfiguration epoch.

C. RL Analysis

1) SubNoC Topology Selection Analysis: The RL automatically learns a policy to select adequate NoC topology at the runtime. In this section, we detail the selection of subNoC topologies for both CPU and GPU applications. Figure 14 shows the topology selection breakdown of CPU applications within a 4×4 subNoC. In general, all applications prefer to select the cmesh, which accounts for 85% of the overall selection due to the sparse communication behavior of CPU applications. However, some of the CPU applications with more memory accesses, like CA, SW, and X264, select about 8% of the tree topology. Mesh and Torus are likely to be less popular than cmesh and tree, both of which are only selected by 3% and 5% on average, respectively. Figure 15 shows the topology selection breakdown of GPU applications within a 4×8 subNoC. As compared to CPU applications, GPU applications have a greater traffic intensity, and thus demanding increased NoC throughput. As a result, the topologies with larger bandwidth and buffers, like mesh, torus, and tree, are selected over 49% of the time. The cmesh is selected between 37% to 64% of the time.

2) SubNoC Size Analysis: As the RL may have different performance in different subNoC sizes, we studied the GPU applications in different sized subNoCs. Figure 16 shows the packet latency and NoC energy analysis of Adapt-NoC. The Adapt-NoC can reduce the packet latency by 5%, 12%, 17%, and 24% as compared to Adapt-NoC-noRL in 2×4, 4×4, 4×8, and 8×8 subNoCs. The Adapt-NoC can achieve 28%-35% energy reduction as compared to Adapt-NoC-noRL.

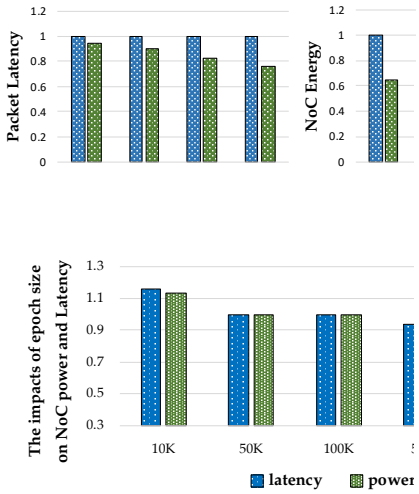


Fig. 17. The epoch size analysis, normalized to th

3) Epoch Size Analysis: As the epoc significant impact on the system perforr epoch size from 10K to 100K cycles in c shows that the epoch size of 50K has th formance and power savings. When the e to 10K cycles, the network latency and increases about 17% and 15%, as com size of 50K. This is due to the unstable caused by frequently switching topology. the power consumption increases with a larger epoch size. This is because the RL policy prefers to select those topologies with higher bandwidth such as tours and tree, resulting in poorer energy efficiency. We note that the Adapt-NoC has very similar network latency and power consumption when the epoch size is between 50K and 100K.

4) Discount Factor Analysis: Figure 18 shows that the discount factor of 0.9 yields the best network performance in terms of latency and power. The smaller discount factor only considers the immediate reward and excludes the impacts of future rewards. On the other hand, a larger discount factor only considers the future rewards.

5) Exploration Rate Analysis: An ϵ-greedy method is used to greedily explore the action space to yield a better policy; however, the value of the exploration rate ϵ is a trade-off between exploration and exploitation. To study the impacts of different exploration rates on performance, we vary the exploration rate from 0 to 0.5. Figure 19 shows that the exploration rate of 0.05 has the best network performance in terms of network latency and power.

VI. Related Work

A. Runtime Reconfigurable NoC Designs

Application behavior has been shown to significantly vary over time [11], [14], [49]–[51], and thus demands an adaptable on-chip communication support. Runtime reconfiguration has been proposed to improve NoC performance, reliability and energy savings. SMART [52] and express virtual channel [53] are two techniques that allow packets to dynamically skip the intermediate routers. However, though such designs can

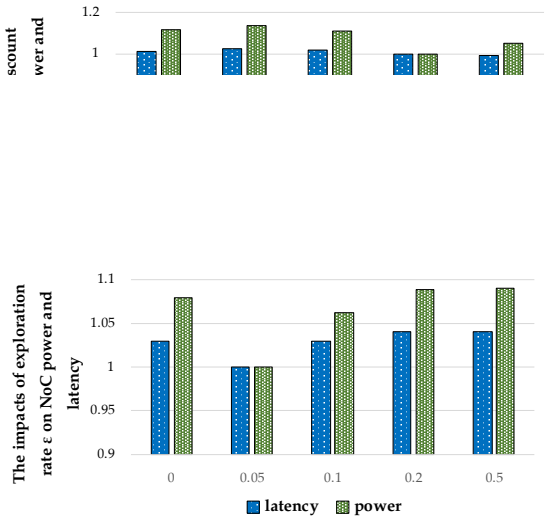


Fig. 19. The exploration rate (ϵ) analysis, normalized to an ϵ of 0.05.

benefit those applications of sparse traffic load, the growing of packet contention in high traffic load will offset their benefits. In [54], a reconfigurable link design can dynamically allocate channel bandwidth between adjacent routers, thereby efficiently improving network throughput and reliability. In [55], a reconfigurable NoC reconfigures NoC topology to detour traffic away from the power-gated routers, improving NoC energy efficiency. Zheng *et al.* [51] designed a flexible NoC topology in the chiplet-based system to support multi-application execution. All of these techniques are orthogonal to the Adapt-NoC design, which could be simultaneously deployed.

B. Heterogeneous NoC Designs

Besides the reconfigurable NoC topologies, significant research [11], [16], [56]–[58] has been proposed to combine the benefits of various topologies. A hierarchical ring topology, with local and global rings, is designed to facilitate the local communication within each region of the NoC using a simple ring topology. Asit *et al.* [11] designed a heterogeneous NoC consisting of two subnetworks with different bandwidth. In addition to these heterogeneous designs, high-radix on-chip networks [16], [57] often deploy both concentration and bypassing techniques to keep the NoC latency scaling to increased NoC size while maintaining the area and wiring costs. However, these designs have restricted flexibility of handling diverse communication behaviors of the co-running applications.

C. Machine Learning in NoCs

Machine learning has been widely applied to aid the optimization of current computer architecture designs, especially the NoC design [59]–[69]. Yin *et al.* [61] explored the use of reinforcement learning to design a self-learned router arbitration policy, and further analyzed the observed RL experience and designed a simple yet effective NoC arbitration policy [62]. Lin *et al.* [63] designed a deep RL framework to optimize the loop placement in routerless NoCs. Wang

et al. [64] used RL to balance the trade-offs among NoC performance, energy efficiency, and reliability. Won *et al.* [65] proposed an artificial neural network-based DVFS technique that can dynamically tune the voltage and frequency of the last-level caches and NoC. Zheng *et al.* [59], [60] used RL to balance the power and performance trade-offs among different low-power techniques.

VII. Conclusions

Modern heterogeneous manycore architectures enable diverse computing choices for concurrent application execution. This requires communication fabric capable of handling various traffic patterns for concurrently running applications. In this paper, we propose **Adapt-NoC**, a learning-enabled flexible NoC architecture together with an RL-based control policy, that can provide efficient communication support for concurrently running applications. The Adapt-NoC can dynamically configure several disjoint regions of the NoC, called subNoCs, with different sizes and locations for various running applications. Each of the dynamically-configured subNoC is capable of implementing a given topology such as mesh, cmesh, torus, or tree depending on the communication need of the running application. In addition, we apply the reinforcement learning (RL) to design an efficient control policy optimizing the subNoC topology selection for a given application with the aim of improving subNoC performance and energy-efficiency. We evaluate the Adapt-NoC using both GPU and CPU benchmark suites. The simulation results show that the Adapt-NoC can achieve 34% latency reduction, 10% overall execution time reduction and 53% NoC energy-efficiency improvement when compared to prior work.

Acknowledgment

This research was partially supported by NSF grants CCF-1702980, CCF-1812495, and CCF-1901165. We sincerely thank the anonymous reviewers for their excellent and constructive feedback.

References

[1] U. Ogras and R. Marculescu. Application-specific network-on-chip architecture customization via long-range link insertion. In *Proceedings of IEEE/ACM International conference on Computer-aided design (ICCAD)*, pages 246–253. IEEE, 2005.

[2] J. Stensgaard, M.and Sparsø. Renoc: A network-on-chip architecture with reconfigurable topology. In *Proceedings of ACM/IEEE International Symposium on Networks-on-Chip (NoCs)*, pages 55–64, 2008.

[3] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad. Application-aware topology reconfiguration for on-chip networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(11), 2010.

[4] M. Kim, J. Davis, M. Oskin, and T. Austin. Polymorphic on-chip networks. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pages 101–112. IEEE, 2008.

[5] B. Beckmann and D. Wood. Managing wire delay in large chip-multiprocessor caches. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 319–330. IEEE, 2004.

[6] C. Kim, D. Burger, and S. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 211–222. ACM, 2002.

[7] R. Das et al. Application-to-core mapping policies to reduce memory system interference in multi-core systems. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 107–118. IEEE, 2013.

[8] J. Lee, S. Li, H. Kim, and S. Yalamanchili. Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(4):48, 2013.

[9] M. Qureshi and Y. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 423–432. IEEE, 2006.

[10] S. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 374–385. ACM, 2011.

[11] A. Mishra, O. Mutlu, and C. Das. A heterogeneous multiple network-on-chip design: an application-aware approach. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 1–10, 2013.

[12] F. Triviño, J. Sánchez, Fr. Alfaro, and J. Flich. Virtualizing network-on-chip resources in chip-multiprocessors. *Microprocessors and Microsystems*, 35(2):230–245, 2011.

[13] L. Chen, K. Hwang, and T. Pinkston. Rair: Interference reduction in regionalized networks-on-chip. In *Proceedings of IEEE International Symposium on Parallel and Distributed Processing Symposium (IPDPS)*, pages 153–164. IEEE, 2013.

[14] R. Das, O. Mutlu, T. Moscibroda, and C. Das. Application-aware prioritization mechanisms for on-chip networks. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 280–291. ACM, 2009.

[15] J. Zhan, O. Kayıran, G. Loh, C. Das, and Y. Xie. Oscar: Orchestrating stt-ram cache traffic for heterogeneous cpu-gpu architectures. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[16] J. Kim, J. Balfour, and W. Dally. Flattened butterfly topology for on-chip networks. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 172–182, 2007.

[17] P. Kumar, Y. Pan, J. Kim, G. Memik, and A. Choudhary. Exploring concentration and channel slicing in on-chip network router. In *Proceedings of ACM/IEEE International Symposium on Networks-on-Chip (NoCs)*, pages 276–285. IEEE, 2009.

[18] D. DiTomaso, A. Kodi, and A. Louri. Qore: A fault tolerant network-on-chip architecture with power-efficient quad-function channel (qfc) buffers. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 320–331, 2014.

[19] J. Balfour and W. Dally. Design tradeoffs for tiled cmp on-chip networks. In *Proceedings of ACM International Conference on Supercomputing*, pages 390–401. ACM, 2006.

[20] A. Bakhoda, J. Kim, and T. Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *Proceedings of IEEE/ACM international symposium on microarchitecture (MICRO)*, pages 421–432. IEEE, 2010.

[21] H. Jang et al. Bandwidth-efficient on-chip interconnect designs for gpgpus. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, page 9. ACM, 2015.

[22] W. Dally and B. Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[23] M. Kandemir, J. Zhang, Y.and Liu, and T. Yemliha. Neighborhood-aware data locality optimization for noc-based multicores. In *Proceedings of International Symposium on Code Generation and Optimization(CGO)*, pages 191–200, 2011.

[24] W. Ding, X. Tang, M. Kandemir, and E. Zhang, Y.and Kultursay. Optimizing off-chip accesses in multicores. In *Proceedings of ACM Conference on Programming Language Design and Implementation (PLDI)*, pages 131–142, 2015.

[25] J. Duato, O. Lysne, R. Pang, and T.M. Pinkston. Part i: A theory for deadlock-free dynamic network reconfiguration. *IEEE Transactions on Parallel and Distributed Systems*, 16(5):412–427, 2005.

[26] T. Rodeheffer and M.D. Schroeder. Automatic reconfiguration in autonet. In *Proceedings of ACM symposium on Operating Systems Principles*, pages 183–197, 1991.

[27] D. Teodosiu, J. Baxter, K. Govil, J. Chapin, M. Rosenblum, and M. Horowitz. Hardware fault containment in scalable shared-memory

multiprocessors. In *Proceedings of International Symposium on Computer Architecture*, pages 73–84, 1997.
[28] O. Lysne, T.M. Pinkston, and J. Duato. Part ii: A methodology for developing deadlock-free dynamic network reconfiguration processes. *IEEE Transactions on Parallel and Distributed Systems*, 16(5):428–443, 2005.
[29] R. Pang, T.M. Pinkston, and J. Duato. The double scheme: Deadlock-free dynamic reconfiguration of cut-through networks. In *Proceedings of International Conference on Parallel Processing*, pages 439–448. IEEE, 2000.
[30] C. Carrion, R. Beivide, J. Gregorio, and F. Vallejo. A flow control mechanism to avoid message deadlock in k-ary n-cube networks. In *Proceedings of International Conference on High-Performance Computing*, pages 322–329. IEEE, 1997.
[31] L. Chen and T. Pinkston. Worm-bubble flow control. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 366–377. IEEE, 2013.
[32] S. Ma, Z. Wang, Z. Liu, and N. Jerger. Leaving one slot empty: Flit bubble flow control for torus cache-coherent nocs. *IEEE Transactions on Computers*, 64(3):763–777, 2013.
[33] A. Ramrakhyani and T. Krishna. Static bubble: A framework for deadlock-free irregular on-chip topologies. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 253–264. IEEE, 2017.
[34] R. Sutton, A. Barto, et al. *Reinforcement learning: An introduction*. MIT, 1998.
[35] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
[36] L. Subramanian et al. The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 62–75, 2015.
[37] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.
[38] N. Binkert and et al. The gem5 simulator. In *ACM SIGARCH Computer Architecture News*, May 2011.
[39] N. Agarwal, T. Krishna, L. Peh, and N. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Proceedings of IEEE international symposium on performance analysis of systems and software*, pages 33–42. IEEE, 2009.
[40] C. Sun and et al. Dsent a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of ACM/IEEE International Symposium on Networks on Chip (NoCs)*, pages 201–210, 2012.
[41] C. Bienia and K. Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.
[42] S. Che, M. Boyer, J. Meng, D. Tarjan, JW Sheaffer, S. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of IEEE international symposium on workload characterization (IISWC)*, pages 44–54, 2009.
[43] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural techniques for power gating of execution units. In *Proceedings of the IEEE/ACM international symposium on Low power electronics and design (ISLPED)*, pages 32–37, 2004.
[44] Bohnenstiehl et al. A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In *Proceedings of IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, pages 1–2, 2016.
[45] P. Moon, V. Chikarmane, K. Fischer, et al. Process and electrical results for the on-die interconnect stack for intel's 45nm process generation. *Intel Technology Journal*, 12(2), 2008.
[46] M. Besta et al. Slim noc: A low-diameter on-chip network topology for high energy efficiency and scalability. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 43–55. ACM, 2018.
[47] AK Mishra, N. Vijaykrishnan, and C. Das. A case for heterogeneous on-chip interconnects for cmps. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 389–400, 2011.
[49] A. Mirhosseini, M. Sadrosadati, B. Soltani, H. Sarbazi-Azad, and T. Wenisch. Binochs: Bimodal network-on-chip for cpu-gpu heterogeneous systems. In *Proceedings of IEEE/ACM International Symposium on Networks-on-Chip (NoCs)*, pages 7–15. ACM, 2017.
[48] N. Weste and D. Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education, 2015.
[50] Y. Yao and Z. Lu. inpg: Accelerating critical section access with in-network packet generation for noc based many-cores. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–26. IEEE, 2018.
[51] H. Zheng, K. Wang, and A. Louri. A versatile and flexible chiplet-based system design for heterogeneous manycore architectures. In *proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.
[52] O. Chen et al. Smart: a single-cycle reconfigurable noc for soc applications. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 338–343. IEEE, 2013.
[53] A. Kumar, L. Peh, P. Kundu, and NK Jha. Express virtual channels: Towards the ideal interconnection fabric. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2007.
[54] Mohammad Al F. et al. Configurable links for runtime adaptive on-chip communication. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 256–261. IEEE, 2009.
[55] P. Ritesh, D. Reetuparna, and B. Valeria. Power-aware nocs through routing and topology reconfiguration. In *Proceedings of ACM/IEEE Design Automation Conference (DCA)*, June 2014.
[56] R. Das et al. Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 175–186. IEEE, 2009.
[57] B. Grot, J. Hestness, S. Keckler, and O. Mutlu. Express cube topologies for on-chip interconnects. In *Proceedings of IEEE International Symposium on Hig-Performance Computer Architecture (HPCA)*, pages 163–174. IEEE, 2009.
[58] H. Zheng and A. Louri. Ez-pass: An energy & performance-efficient power-gating router architecture for scalable nocs. *IEEE Computer Architecture Letters*, 17(1):88–91, 2018.
[59] H. Zheng and A. Louri. An energy-efficient network-on-chip design using reinforcement learning. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.
[60] H. Zheng and A. Louri. Agile: A learning-enabled power and performance-efficient network-on-chip design. *IEEE Transactions on Emerging Topics in Computing*, 2020.
[61] J. Yin et al. Toward more efficient noc arbitration: A deep reinforcement learning approach. In *Proceedings of the 1st International Workshop on AI-assisted Design for Architecture (AIDArc)*, 2018.
[62] J. Yin, S. Sethumurugan, Y. Eckert, A. Smith, Patel C., Morton E., M. Oskin, and Loh GH. Enright Jerger, N. Experiences with ml-driven design: A noc case study. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.
[63] T. Lin, D. Penney, M. Pedram, and L. Chen. A deep reinforcement learning framework for architectural exploration: A routerless noc case study. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.
[64] K. Wang, A. Louri, A. Karanth, and R. Bunescu. Intellinoc: a holistic design framework for energy-efficient and reliable on-chip communication for manycores. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pages 589–600, 2019.
[65] JY Won et al. Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 308–319. IEEE, 2014.
[66] K. Wang, H. Zheng, and A. Louri. Tsa-noc: Learning-based threat detection and mitigation for secure network-on-chip architecture. *IEEE Micro*, 40(5):56–63, 2020.
[67] Y. Chen and A. Louri. Learning-based quality management for approximate communication in network-on-chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3724–3735, 2020.
[68] K. Wang and A. Louri. Cure: A high-performance, low-power, and reliable network-on-chip design using reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 31(9):2125–2138, 2020.
[69] Y. Li and A. Louri. Alpha: A learning-enabled high-performance network-on-chip router design for heterogeneous manycore architectures. *IEEE Transactions on Sustainable Computing*, 2020.