Machine Learning Enabled Solutions for Design and Optimization Challenges in Networks-on-Chip based Multi/Many-Core Architectures

MD FARHADUR REZA, Department of Mathematics and Computer Science, Eastern Illinois University, USA

Due to the advancement of transistor technology, a single chip processor can now have hundreds of cores. **Network-on-Chip (NoC)** has been the superior interconnect fabric for multi/many-core on-chip systems because of its scalability and parallelism. Due to the rise of dark silicon with the end of Dennard Scaling, it becomes essential to design energy efficient and high performance heterogeneous NoC-based multi/many-core architectures. Because of the large and complex design space, the solution space becomes difficult to explore within a reasonable time for optimal trade-offs of energy-performance-reliability. Furthermore, reactive resource management is not effective in preventing problems from happening in adaptive systems. Therefore, in this work, we explore machine learning techniques to design and configure the NoC resources based on the learning of the system and applications workloads. Machine learning can automatically learn from past experiences and guide the NoC intelligently to achieve its objective on performance, power, and reliability. We present the challenges of NoC design and resource management and propose a generalized machine learning framework to uncover near-optimal solutions quickly. We propose and implement a NoC design and optimization solution enabled by neural networks, using the generalized machine learning framework. Simulation results demonstrated that the proposed neural networks-based design and optimization solution improves performance by 15% and reduces energy consumption by 6% compared to an existing non-machine learning-based solution while the proposed solution improves NoC latency and throughput compared to two existing machine learning-based NoC optimization solutions. The challenges of machine learning technique adaptation in multi/many-core NoC have been presented to guide future research.

CCS Concepts: • **Networks** → **Network on chip**; • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Multicore architectures**; **Heterogeneous (hybrid) systems**; **System on a chip**; • **Hardware** → **Electronic design automation**; **Power and energy**; *Very large scale integration design;*

Additional Key Words and Phrases: Machine Learning (ML), multi/many-core systems, Network-on-Chip (NoC), training (learning), prediction (inference), online learning, classification, regression, design and optimization, design-time, run-time, energy-efficiency, power, thermal

Author's address: Md F. Reza, Department of Mathematics and Computer Science, Eastern Illinois University, 600 Lincoln Ave, Charleston, Illinois, USA, 61920; email: mreza2@eiu.edu.

1550-4832/2023/06-ART23 $15.00
https://doi.org/10.1145/3591470

ACM Reference format:
Md Farhadur Reza. 2023. Machine Learning Enabled Solutions for Design and Optimization Challenges in Networks-on-Chip based Multi/Many-Core Architectures. *ACM J. Emerg. Technol. Comput. Syst.* 19, 3, Article 23 (June 2023), 26 pages.
https://doi.org/10.1145/3591470

1 INTRODUCTION

With the advancement and miniaturization of transistor technology, hundreds to thousands of cores can be integrated on a single chip [9]. We can even build a data-center on a chip [48]. But the power consumption of transistors does not decrease in the same way as transistor size due to the problem with energy scaling of transistor technology, which results in Dennard scaling failure [24, 79]. For example, while transistor density increases by $2x$, transistor power efficiency only improves by $1.4x$ in every processor technology, resulting in a $2x$ deficit in power budget [79] following the same trend of core integration. [55] presents the importance of interconnect (e.g., NoC) in terms of area, power, and performance in multi-core architectures. Bus, crossbar, point-to-point networks, and NoC topologies are shown in Figure 1. Traditional bus has been used for communication between the cores in multi-core architectures. However, traditional bus has limitations in the latency perspective, where only one core can transmit or receive at a time in a cycle, and wired bus transmission suffers from high latency. Hierarchical bus can increase parallelism, but it increases the complexity of the network and reduces scalability. Point-to-point network needs a lot of dedicated links between cores, and for many-core systems, it needs very long links where link delay increases with the increase in wire-length. Furthermore, point-to-point incurs high hardware costs because of many long-links, which makes it infeasible for many-core systems. Similarly, crossbar requires a lot of long links and multiplexers, resulting in high hardware costs.

NoC offers several important benefits over the traditional bus, for example, in terms of scalability, parallelism, throughput, and power efficiency [5, 19, 21, 27, 37, 58]. NoC supports parallelism as multiple cores can communicate with each other if there is no interference between two communications. NoC provides scalability as resources can be added (capacity increases) in the network without changing the existing components, e.g., routers, timing buffers, and so on. NoC is fault-tolerant as nodes can communicate even if a component fails because of the path diversity property of NoC. Because of the modular design of NoC, regions or a component of the NoC can be switched off (power-gating) without affecting other components [17]. This power-gating capability is a very important feature for large-scale systems, as un-utilized resources can be power-gated to reduce static power consumption. Short point-to-point links (wires) in NoC have lower dynamic power consumption than the long wires in bus systems. A lot of research work has been done on NoC communication, showing its advantages over traditional bus, crossbar, and point-to-point networks. The comparisons between NoC and bus have been summarized in Table 1. However, the relative power consumption of NoC has been increasing with the increase in the number of cores in a chip. For example, NoC consumes 36% of the total chip power in 16-tile MIT Raw [80]. At 22*nm* and 8*nm* process technology, respectively, 21% and 50% of the chip resources may have to be dark (deactivated) [28]. Besides power limitations, a component of the chip cannot exceed the manufactured maximum temperature at any instant of time as high temperature increases several failures (e.g., electro-migration) and can cause permanent damage to the chip (e.g., burn the chip). As NoC consumes a significant percentage of chip energy [80], energy efficiency of NoC becomes a critical issue. Dark silicon (turned off) and dim silicon (voltage/frequency (V/F) scaled) become the major solutions for current energy limitation in multi/many-core chip systems [28, 79]. Besides dark/dim silicon, heterogeneous cores (CPU, GPU, accelerator, etc.) are being integrated to

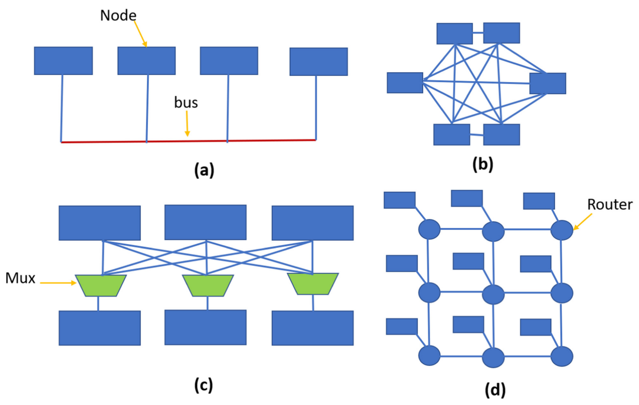

Fig. 1. Topologies: (a) Bus (b) Point-to-Point (c) Crossbar (d) NoC.

increase energy-savings and/or throughput of chips [54]. As significant power-thermal sources, core and NoC resources (links and routers) need to be configured at design-time and/or reconfigured at run-time depending on traffic demands, to decrease pressure on meeting the power and thermal budgets of multi/many-core chips.

The design space of a multi/many-core NoC involves trade-offs of many design and optimization parameters, such as power and thermal budgets, latency and throughput requirements, power-gating of resources, link bandwidth, node voltage, and router buffer size. As these parameters are interrelated, the design space for multi/many-core systems increases exponentially with the increase in the number of nodes in the system and the number of tasks and applications to be allocated. NoC is used for communicating traffic between tasks of an application. Task-resource co-allocation in many-core networks can introduce many problems (energy hotspots, load-imbalance, etc.). Due to limited power and thermal budgets, the task-resource mapper needs to determine the required active (and dimmed) resources while deactivating other resources. Reactive task-resource allocation may not produce a solution that meets task requirements or may not produce a solution within a reasonable period because of the size and complexity of the multi/many-core chip design space. Reactive task-resource allocation can give a temporary solution, which might lead to problems in networks for subsequent mapping of tasks and applications at run-time. Because of heterogeneous application demands, run-time mapping (i.e., allocating tasks to resources) needs to proactively determine the required resources in the next time interval. Heuristics (e.g., genetic algorithm and simulated annealing) and linear programming optimizers (e.g., integer linear programming solver) may not be able to compute results in time to satisfy run-time requirements because of their reactive nature and/or because of the exponential increase in time complexity with the increase in NoC resources. This motivates us to use **machine learning (ML)** techniques, such as regressions and **neural networks (NNs)**, to monitor and predict resource utilization in the future and take appropriate actions (prior to the system reaching that state) accordingly to prevent any kind of problems, such as resource overloading, in the network. Existing mapping solutions do not consider an ML solution together with dark and dim silicons during task-resource allocation. The goal is to design and configure the heterogeneous NoC using ML for optimized balancing of the total workloads while meeting various chip constraints (power, thermal, etc.) and application demands. The contribution of this work (which is the extended version of [69]) is outlined below:

- *Survey on Existing ML Solutions for NoC:* We explore the existing ML solutions for NoC-based multi/many-core architectures design and optimization in Section 2.

Table 1. NoC vs Traditional Bus

Type	Traditional Bus	Network-on-Chip
Scalability	Not scalable	Scalable modular design
Parallelism	Does not support	Support multiple communications
Fault Tolerance	Not Supported	Multiple paths between pairs
Power-gating	Cannot be applied	Can switch-off parts of network
Dynamic Power	High for long wire	Low for short link

- *Design-time and Run-time Optimization Challenges in NoC:* We have explored the design-time and run-time optimization challenges in NoC-based multi/many-core architectures in Section 3.
- *ML Framework for NoC Design and Optimization:* We propose an ML-enabled generalized learning and configuration framework to address the design and optimization challenges of NoC in Section 4.
- *NN for NoC Design and Optimization:* In Section 5, we proposed and implemented an NN-enabled NoC design and optimization solution using the generalized ML framework.
- *Challenges for ML Models implementation in NoC:* We identify and discuss the limitations and challenges to apply ML for multi/many-core NoC design and optimization in Section 6.

2 SURVEY ON EXISTING MACHINE LEARNING SOLUTIONS FOR NOC

We discuss the existing applications of ML models in multi/many-core systems and on-chip networks in this section and summarize the addressed problems and solutions of the related work in Table 2. ML techniques, such as regressions and NNs, are being explored for design and optimization in multi/many core systems and NoC. [50] proposed a framework for many-core systems design using data-driven ML techniques, while addressing the challenges in the computation, interconnection, and memory layers. The framework integrates both learning and domain knowledge of many-core systems. [3] proposed a **reinforcement learning (RL)**-enabled **dynamic voltage-frequency scaling (DVFS)** control policy for multi-core architectures. Per-core RL agents learn and improve their control policies independently, while retaining the ability to coordinate their actions to accomplish system-level power management objectives. [83] proposed a distributed RL-based approach that allows agents at runtime to map the tasks onto NoC by learning an optimal policy. [71] proposed an RL-enabled routing technique that selects the best routing algorithm using NoC utilization and congestion information to improve communication performance. [68] proposed an RL technique to proactively configure the NoC link bandwidths in heterogeneous architectures for energy-efficiency. NoC link bandwidths are configured dynamically based on the learning of the system (e.g., link utilization, buffer utilization). [70] proposes a **deep reinforcement learning (deep RL)** technique to configure the NoC V/F-levels depending on demands of the tasks, where an NN is used to approximate Q-values for different V/F actions. NN was used to remove the need of large state-action mapping table for large-scale NoC. [82] uses distributed RL agent per router to select an optimal operating policy with the objective of achieving high performance, increased reliability, and reduced power consumption. The same authors also used per-router RL to provide a proactive fault-tolerant mechanism for NoC in [81]. [86] proposed a deep RL approach for efficient NoC arbitration. The proposed self-learning decision-making mechanism reduces packet latency, resulting in improved NoC throughput. The authors in [34] proposed a proactive energy management strategy that relies on an offline-trained regression model and provides a wide variety of V/F pairs. Distributed RL agents are then used to select the appropriate V/F on a per-router basis without the need for global coordination, thereby reducing the overhead and complexity. [87] proposed ML techniques to learn arbitration policies from simulation traces.

Table 2. Comparison of Related Work

Related Work	Addressed Problems	Solutions
[50]	Optimize computation, interconnection, and memory layers simultaneously	Data-driven framework integrates learning and domain knowledge of all layers
[3]	System level power management	RL-enabled DVFS control
[83]	Task mapping onto NoC	RL-enabled mapping policy
[87]	Reduce area for NoC arbitration logic	Deep learning-based arbitration policy
[71]	NoC congestion and performance	RL-enabled routing algorithm selection
[70]	Improve NoC performance and energy	Deep RL and Q-learning for DVFS control
[82]	Optimize performance/power/reliability	Distributed RL selects operating policy
[86]	NoC arbitration decisions	Deep RL-enabled NoC arbitration
[34]	NoC energy management	Linear regression and distributed per-router RL for DVFS control
[72]	NoC energy-savings	NN-based configuration of nodes and links
[25]	Proactive fault-tolerant in NoC	Decision tree-based fault-tolerant solution
[26]	Optimize NoC performance and energy	Decision tree-based solution provides additional bandwidth and power-gates resources
[46]	Prevent NoC hotspots	NN-based hotspot prediction and congestion mechanism remove hotspots
[51]	Efficiently route NoC traffic	NN-based routing algorithm selection
[67]	NoC packet latency prediction	Support vector regression predicts latency
[76]	Dynamic power management in NoC	NN predicts links on/off status
[45]	Thermal prediction in many-core	Lasso regression-based thermal prediction
[23]	Placing communication links in 3D NoC	STAGE ML guides the local search

[72] proposed a run-time predictive configuration of both nodes and links of NoC using NN online learning for energy-savings while addressing both power and temperature constraints. [25] proposed decision tree-based proactive, fault-tolerant schemes to predict and mitigate errors before the fault affects the system. The decision tree-based model achieves a reduction in packet re-transmission and energy savings. [26] also proposed a decision tree algorithm to predict traffic flow and link utilization. The proposed method can dynamically provide additional bandwidth when required in order to improve performance and accurately power-gate the links and buffers to improve power dissipation. [46] presented an NN-based intelligent hotspot prediction mechanism that was used with a congestion-control mechanism to handle hotspot formations efficiently. Here the NN uses online statistical data to dynamically monitor the NoC interconnect fabric, but reactively predicts the location of hotspot(s). [51] proposed an NN for predictive routing algorithm that uses anticipated global network state and congestion information to efficiently route traffic. [67] proposed an ML-based approach for NoC performance prediction. The authors developed a systematic ML framework that uses the kernel-based support vector regression method to predict the channel average waiting time and the traffic flow latency. The authors in [76] proposed an intelligent dynamic power management policy for NoCs. They feed the link utilization information of a sub-network to an NN to predict link on/off status based on the pre-defined utilization threshold. [45] developed a learning-based framework using Lasso regression to enable fast and accurate transient thermal prediction. [23] leverages ML to intelligently explore the design space of 3D NoC to optimize the placement of both planar and vertical communication links for energy efficiency. The authors use the STAGE online ML algorithm, which can intelligently select promising starting states that guide the local search procedure towards significantly better solutions compared to traditional local optimal algorithms, such as hill-climbing and simulated annealing. Therefore, there have been some existing works that adopt ML techniques to address NoC-based multi/many-core design and performance improvement challenges. To address NoC design

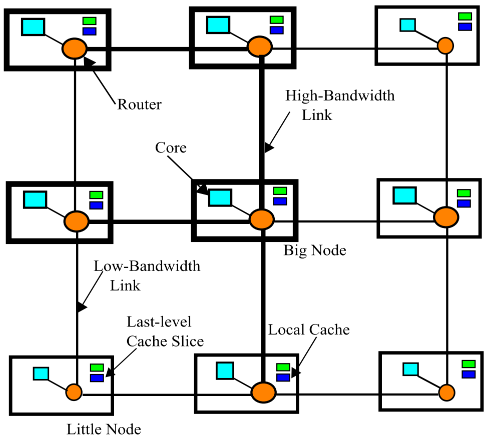

Fig. 2. Heterogeneous link and core configuration in 3×3 NoC.

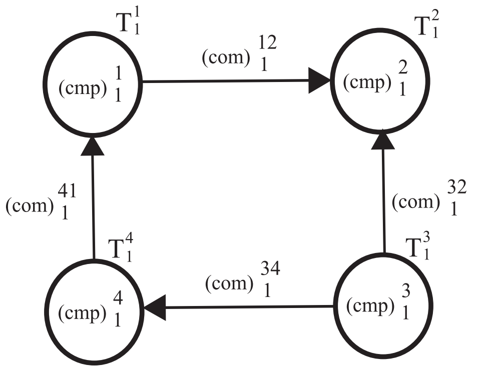

Fig. 3. An example of application task graph G.

and optimization challenges, we have proposed a generalized ML framework that works for most supervised ML algorithms without any changes in datasets and/or ML phases (training/inference). We have then implemented an NN-based solution using the generalized framework for NoC design and optimization. We discuss the details of design-time and run-time optimization challenges in multi/many-core NoC in the next section.

3 MULTI/MANY-CORE NOC DESIGN AND OPTIMIZATION CHALLENGES

Efficient task-resource co-allocation policies are required for running diverse applications on NoC-based multi/many-core platforms. During task-resource co-allocation in many-core systems, performance optimization requires us to quickly determine high-performance points in a vast allocation space, as well as anticipate and respond to dynamically changing workload demands. Coordinated management of multiple interacting resources on multi/many-core architectures remains an open problem. The requirements for faster speed and lower energy consumption are ever-increasing. Single-core performance improvement by increasing clock frequency has reached its limit due to high energy consumption at high frequency. However, multi/many-core on-chip systems have been built to fulfill the high-performance requirements. As multi/many-core systems can provide multithreaded performance, performance can be increased indefinitely by theoretically increasing the number of cores. However, the interconnection between cores imposes a limit on performance gain from many-core systems. With the increase in the number of cores, communication latency between distant nodes increases. The fraction of time spent on communication for an application can be more than 50% for large-scale systems [6], which significantly hampers the parallelism performance (as computation depends on communicated data). The key element to scalable chip performance is the on-chip interconnects connecting the cores (and memory). NoC has been introduced to maximize the advantages of network communication. Figure 2 shows a 3×3 2D NoC, where a tile contains a router, a processor, a local cache, and a slice of the last level cache. Routers are connected by links to form the NoC. Here, the goal is to design and configure links and routers of the NoC heterogeneously at design-time and/or run-time and power-gate the idle resources.

A set of K applications can arrive at the multi/many-core NoC system with N nodes for running. An application A_k can have M variable tasks, and m^{th} task of A_k is represented by T_k^m. Computation demand of T_k^m is $(cmp)_k^m$, and communication requirement between tasks m and n is $(com)_k^{mn}$. A sample task graph G of an application with computation and communication demands is shown in Figure 3. A mapping solution needs to satisfy both computation and communication demands of the tasks. Task-resource co-allocation in NoC becomes even more critical for a system with many cores. The heterogeneity in both processing nodes and workloads makes it more challenging to design NoC architecture to achieve both computation and communication efficiency.

Design-time task-resource co-allocation needs to consider chip constraints in terms of power budget, thermal budget, computation and communication capacity, and area budget. In addition to these design-time challenges, run-time optimization needs to configure the system and networks dynamically under different workloads of different applications. An application with unknown demands can arrive in the system, and/or an application demand may change at run-time. The utilization of heterogeneous NoC resources needs to be monitored for efficient allocation of resources for incoming and (already) running tasks and applications. The multi/many-core system and network configuration consists of many parameters, including (but not limited to) topology, task-to-node assignment, task migration, setting V/F of nodes and links, setting link width and type, and router design. The design space for task-resource co-allocation in multi/many-core NoC increases exponentially with the increase in the number of cores and NoC resources. NoC configuration involves many interrelated system design parameters, such as power/thermal budgets and latency/throughput requirements. Furthermore, a system can face several challenges and problems, such as workload variation, link failure, router failure, deadlock, and process variation. The design-space of multi/many-core NoC design/optimization, including multi/many-core NoC constraints and application constraints, are presented in the following paragraphs.

3.1 Computation Constraint

A multi/many-core architecture can have cores (CPU, GPU, accelerator, etc.) with heterogeneous computation capacity. Applications also have heterogeneous tasks. A task cannot be mapped to a node (core) i if the computation demand $(cmp)_k^r$ of the task exceeds the maximum computation capacity of the selected node. However, more than one task can be mapped to a node if node capacity allows. Computation demand can be represented as the amount of data processed per unit of time. Optimal task-to-core allocation is needed for an energy- and performance-efficient solution.

3.2 Communication Constraint

The capacity of a router is determined by the number of ports in the router, the number of buffers in a port, and the size of each individual buffer. The size of a buffer determines a flit (flow control unit) size, as a flit (part of a packet) needs to be stored on a buffer due to contention in NoC. Multiple traffic flows may simultaneously traverse through a communication link. A link of a NoC has a limit on maximum possible communication bandwidth as higher capacity link has higher power consumption. Bandwidth of a link is calculated by multiplying the link width configuration (e.g., 32/64/128 bits) with the corresponding link frequency. Since the task mapping result may significantly affect the traffic distribution, it is essential to configure NoC router design (e.g., buffer size) and link bandwidth according to the traffic needs.

3.3 Chip-Level Power Budget Constraint

The sum of power consumption of all the chip resources (nodes i + links ij) cannot exceed the designed maximum power consumption budget of the chip. Dynamic power consumption is calculated by multiplying the frequency with the square of the voltage-level of components. In addition to dynamic power consumption, resource leakage power and activation/deactivation power also contribute to the overall chip power consumption.

3.4 Tile-Level Thermal Budget Constraint

A tile contains core, router, and cache. The temperature of a tile is measured using ambient temperature, power consumption, surface area, and thermal resistance of the nodes (core/router/cache) and corresponding links. The thermal consumption of any tile can not exceed the designed maximum thermal consumption threshold of a chip.

3.5 Chip Area Constraint

The area is measured by the amount of space occupied by the NoC resources (links and routers), cores, and other uncore components (memory controller, shared cache, etc.). For example, in a $22nm$ process technology, the chip area required for 1-bit width link is 200 μm^2, where the wire pitch and length are $80nm$ and $2.5mm$, respectively. The NoC designer needs to determine various parameters, including the optimal bandwidth for each link and the number of router ports and buffers at a port. During NoC resource configuration, the solution must adhere to the total chip area budget.

3.6 QoS: Task Deadline Requirement

Task deadline is a QoS requirement for an application. A deadline constraint means that a task has to finish its execution and all of its communication with its neighbors in the task graph within the corresponding deadline. If a task m of an application has $((T_{cmp})_k^m)^p$ execution time at processor p and $((T_{com})_k^{mn})^{pq}$ communication time with task neighbor n that is mapped to processor q, and given a deadline D_k^m for task m, then the task m needs to finish its execution within deadline D_k^m.

3.7 Distributed Resource Management

Due to heterogeneous application demands, run-time resource allocation and configuration needs to determine the required resources in the next time interval. A resource manager monitors and configures the NoC resources within its managed space, such as a cluster, a node, or the entire NoC. The centralized manager approach, presented in some works [15, 18, 33], is not feasible for hundreds-of-nodes-based NoC due to high communication traffic around the controller. Distributed (decentralized) resource management is superior to the centralized mechanism due to it scalability in many-core NoC. Resource monitoring and controlling can be done by the distributed run-time manager in every cluster, where the chip is partitioned into a number of clusters of uniform size. The clustering decision is an NP-hard optimization problem as the time required to find an exact solution increases exponentially with the size of the NoC. A cluster manager monitors and configures the resources within a cluster. A cluster manager collects statistics at fixed intervals for the corresponding cluster, including tasks-to-cores mappings, resources activation/deactivation status, capacity utilization of resources, and tasks start-finish status. These statistics are used for mapping decision of incoming tasks, power-gating of idle resources, and activation of the required resources in the next interval. Run-time agent-based distributed application mapping solutions for on-chip communication have been presented in [32, 52].

3.8 Task Migration

For optimal mapping, such as in terms of energy consumption, in NoC, tasks may need to be migrated (moved or swapped) among cores during run-time [74]. For example, an incoming task may need to run on a big core while a big core is being utilized by a small task. In this case, it might be optimal to migrate the small task to another little core and then map the incoming big task to a big core. Task migration can also be performed if the migration reduces overall power consumption or thermal hotspots. Task migration involves a lot of overhead, e.g., saving variables and routing the information through NoC to the migrated core. Therefore, the overhead of task migration needs to be considered to justify its effectiveness in terms of overall energy consumption and the penalty of task migration. A feasibility study on supporting task migration in multi-core systems was discussed in [7]. A task migration mechanism for distributed many-core operating systems presented in [38].

3.9 Heterogeneity of Applications and Tasks

Many heterogeneous applications with unpredictable demands can enter the system for execution, making run-time mapping and configuration a challenging task. An application can be communication-intensive (e.g., memory-intensive) and/or computation-intensive (e.g., throughput-intensive). Additionally, an application has heterogeneous traffic requirements among its tasks. Therefore, tasks need to be allocated to the heterogeneous resources of the NoC efficiently, considering the optimization objective(s). Thus, the design challenges of NoCs include run-time resource monitoring and configuration to meet the demands of heterogeneous applications.

3.10 Resource Heterogeneity

NoCs can connect to different types of computing resources, such as CPU cores, specialized accelerators, and GPU cores, in multi/many-core architectures [13, 44, 73]. Resource heterogeneity presents an opportunity to better match the needs of applications with specialized resources. For example, in a heterogeneous CPU-GPU system, GPU cores can run throughput-intensive tasks, while CPU cores can run latency-sensitive tasks. Therefore, resource heterogeneity can provide increased parallelism with high efficiency [54]. A computing system also has local L1 cache, L2 shared cache, memory, and memory controller(s) for efficient handling of application traffic. NoCs can also have heterogeneous routers (big/little) and links (with different bandwidths) for communication. Because of the heterogeneity of resources, NoC design needs to be optimized to meet the heterogeneous communication demands among the resources.

3.11 NoC Topology

The NoC topology plays a vital role in running tasks and managing resources in multi/many-core architectures. As communication paths between tasks depend on the NoC topology, task mapping must consider communication paths. For example, deadlocks can form in the NoC topology because of communication dependencies among packets or flits in the network. Packet rerouting because of faults in the network depends on the NoC topology and routing algorithms. Designing and optimizing NoC topologies to satisfy the requirements of the applications is a greatly challenging problem to address. Many different types of topologies are available for NoC [35], such as 2D Mesh, Torus, Folded Torus, Ring, Octagon, Star, Spidergon, Clos, Fat Tree, Butterfly, Flattened Butterfly, SPIN, and 3D. The topology can be selected based on the energy and performance objectives. The scalability of the NoC topology is crucial for the success of many-core architectures. For example, Ring NoC is not scalable for many-core systems since the hop count grows linearly with the number of interconnected elements. With the increase in hop count, latency and energy consumption increase. The silicon chip area is 2D, so 2D Mesh utilizes the chip area effectively. 3D integration stacks multiple dies in NoC, which decreases network hop count, wire delay, and power consumption compared to conventional 2D integration [66]. However, thermal hotspots (due to heat dissipation) and manufacturing difficulty are the major challenges of 3D **integrated circuits (IC)**. Several NoC architectures are evaluated with respect to their performance/energy/area and their trade-offs in [10, 65].

3.12 Interconnection Type

Interconnection between NoC routers can have different technologies, e.g., wireline, wireless [64], **Radio-Frequency (RF)** [16], and optical [1]. The selection of appropriate interconnection technologies depends on many factors, such as chip budgets and application QoS requirements. Performance, power, and cost trade-off analysis needs to be conducted for selecting a specific technology. NoCs can have multiple interconnection technologies in the same system. Long-range (greater than one hop) wired interconnect has been used to enhance the communication between pairs of frequently communicating routers [61]. For wireless interconnect, carbon-nanotube, $mmWave$, and ultra-wideband technologies are available. High latency and energy dissipation of long-distance multi-hop communication between cores in conventional NoCs can be addressed by introducing long-range, high bandwidth, and low power wireless links between the distant cores [64]. Optical technology can also be a solution for the high latency/energy of electrical NoCs in large-scale systems.

3.13 Interdependency of Technologies

Different types of power and thermal minimization technologies, such as DVFS [62, 85], power-gating, and clock-gating [4], can be applied in the NoC. Multiple technologies can be applied simultaneously in the system. Power-gating is implemented by turning off the supply voltage to functional blocks and routers [43]. Multiple V/F-levels are available for DVFS to satisfy performance requirements, for example, a commercial processor may have five different voltage-levels: 0.8, 0.9, 1.0, 1.1, and 1.2 Volts are available. V/F-levels can be configured based on the resource utilization (e.g., NoC buffer) and performance requirements, such as latency/throughput. Lower V/F-levels of DVFS can help minimize the energy consumption of various applications/benchmarks [42] in many-core systems. Furthermore, a V/F island [62] can be formed in large-scale systems to allow cores and network elements (routers and links) that behave similarly to share the same V/F values without significant performance penalties. A voltage regulator can have several voltage levels to control the voltages of a node, region, or island. However, different cores or islands running at different V/F-levels require asynchronous interfaces and a complex power delivery system. Moreover, voltage scaling increases sub-threshold currents and leakage power, which introduces design challenges for circuits. The chip's leakage power percentage also increases significantly with the decrease in transistor size. As noted in [11], leakage power increases by five times each transistor generation as dynamic power remains constant (according to scaling theory). Therefore, a trade-off analysis needs to be conducted on dark/dim silicon to evaluate their effect on the latency and throughput of applications and systems.

3.14 Dark Silicon

The dark silicon problem is a major obstacle for many-core systems [28]. Across two process generations, we can either increase core count by 2× or frequency by 2× due to power budget limitations. The remaining potential results in either dark silicon or dim silicon, depending on the preference for frequency versus core count. Furthermore, high temperature increases leakage power, which is the major power contributor in deep-submicron technologies. The temperature of an active core is affected by its neighbor nodes' temperature. If a neighbor node is dissipating heat, then some percentage of that heat also reaches its neighborhood. To reduce the thermal impact of an active core, we can mix the active and dark cores together (as dark cores do not have heat impact on an active core) instead of separating dark cores from active regions [47]. The distribution of active and dark cores can help us run more tasks with the same power and thermal budgets. Challenges related to temperature, reliability, and variability in the dark silicon era have been presented in [77]. Power and thermal limitations in the dark silicon era have been discussed in [28, 79]. Efficient mappings of applications in many-core systems considering dark silicon have been presented in [41].

3.15 Reliability

Due to the miniaturization of transistor technology, circuit elements become more susceptible to failure, and manufacturing and process variations increase. Furthermore, lower voltage and higher frequency can negatively impact system reliability; for example, a small charge is needed to flip a bit in memory (soft error) in a low voltage state. Aggressive power management policies such as power gating and DVFS can decrease the overall component reliability due to the degradation effect of temperature cycles on modern chip materials [78]. Transistor aging and variability degradation reliability issues have been discussed in [12]. Reliability issues also arise due to **negative bias temperature instability (NBTI)** and **hot carrier injection (HCI)**, which can cause circuits to fail to meet timing constraints [53]. In addition to thermal hotspots, spatial temperature variations across the chip can lead to performance degradation or logic failures. A substantial increase in the failure rate is forecasted for future CMOS technologies [12]. Core and NoC resources can become faulty or unreliable during run-time. If a task is running on a faulty core, that task needs to be migrated from the faulty core to a normal core. If a link and/or a router becomes faulty, communication traffic needs to be rerouted to avoid faulty resource(s). Efficient routing algorithms are needed to have a robust NoC to handle fault and reliability issues. Proactive techniques can help to determine the reliability of nodes and links in the NoC before allocating tasks/traffic on those resources. As resource reliability issues increase in large-scale systems, proactive techniques are crucial for efficient resource management and for a robust multi/many-core NoC.

3.16 Routing - Router Design and Routing Algorithm

There has been a lot of research on router design for efficient routing. Routers are the major contributors to the area and power consumption in NoCs, for example, routers consume 40% of total on-chip network power in 16-tile MIT Raw [40]. Router design includes designing buffers, virtual channels, ports, arbitration mechanisms, and the number of routing stages, and so on. Buffer is one of the major area contributors to the router's size and complexity and NoC latency and energy consumption. Active buffer size can be different in different routers to meet the application's requirements, for example, allocate more buffering resources only to the heavy loaded routers. Virtual channels are used to increase the utilization of NoC resources and improve the NoC throughput [84], and further, to prevent deadlock in NoC [20]. Due to limited buffering resources in on-chip systems, wormhole switching is the most promising technique for NoC. Furthermore, a router can also be designed without buffers to reduce the cost/complexity [60]. However, a bufferless router solution can degrade the performance of NoC. Circuit switching is not an effective technique for dynamic application traffic due to its static resource assignment. However, circuit switching performs better for application-specific NoC when the application traffic is known. Also, circuit switching can offer guaranteed QoS as required by an application. However, a hybrid combination of circuit and packet switching techniques can co-exist in a NoC [57, 59].

Many routing algorithms and designs have been proposed for efficient communication in NoC [27, 63]. Implementation complexity and performance requirements are two major concerns in selecting deterministic and adaptive routing strategies. Deterministic routing (e.g., XY or YX Routing) is more appropriate if the traffic generated by the application under consideration is predictable. Adaptive routing algorithms depend on the real-time network state information (e.g., congestion, arbitration delay) for performance and power efficiency. Deterministic routing requires fewer resources while guaranteeing an orderly packet arrival, while adaptive routing capability increases router complexity and hardware requirements. Deterministic and partially adaptive algorithms based on the turn model [36] guarantee free deadlock and livelock operation, while fully adaptive strategies require extra precaution. There is a need for proactive techniques that can learn the NoC performance states and required decisions from past experiences by inspecting readily available features and outputs, and then predict the routing decision based on current network traffic in real-time.

3.17 Scalability

Scalable interconnect is one of the major technological challenges in many-core and exa-scale systems [6]. The relative overhead of interconnect increases with the increase in cores (as transistor technology advances) for various reasons, including increased communication latency and increased arbitration overhead [55]. Due to the increase in the size of many-core systems and the need to run multiple applications on the system, it becomes difficult to provide an optimal solution within a reasonable time both at design-time and run-time. At run-time, a centralized system manager may not satisfy the requirements of tasks within a reasonable time or other constraints (e.g., performance, power) because of the overhead associated with the decisions in large-scale systems. Therefore, NoC configuration control needs to be distributed in large-scale systems so that NoC and the system can have the ability to satisfy the QoS requirements (e.g., delay) of an increasing number of applications. An energy-efficient scalable NoC architecture for many-core systems was presented in [56].

3.18 Approximate Communication

Communication is the bottleneck for highly parallel computing machine such as many-core chip systems, as it costs more time and energy than computation. Many compute and data intensive applications, including pattern-recognition, image processing, and scientific computing, can tolerate errors with acceptable accuracies. The error resiliency of the applications can be exploited by approximating the data between compute nodes. Approximate communication techniques reduce the data movement by leveraging the error-tolerance of applications. Some approximation techniques to reduce data movement include reducing data protection, compressing data, and predicting data [8]. However, the quality of the output must be maintained while approximating the data [49]. The error tolerance requirement of an application can suddenly change at run-time. Thus, an approximation technique needs to measure the requirements of an application at run-time and make a approximation decision dynamically to reduce the data movement and/or produce the output within the acceptable error threshold [14]. Furthermore, different applications have different requirements for output quality, making the application of approximate communication on NoC a challenging task.

3.19 Memory Bandwidth and Memory Controller Placement

Memory controllers (MCs) play an important role in forwarding core/processor traffic to/from memory. The placement of MCs affects the average hop count from cores to MCs in NoC. Different MC placements can help improve network latency and bandwidth by spreading the core-memory traffic and balancing the NoC load. A many-core system with hundreds to thousands of cores requires very high memory bandwidth. A lot of research has been conducted on efficient memory architecture design, such as distributed or shared memory and cache coherency protocols. Emerging memory technologies have also been introduced to reduce or remove the memory bottlenecks in on-chip systems. Memory-efficient on-chip networks to achieve higher memory bandwidth was presented in [22]. Due to all of these design-time and run-time design parameters, such as adaptive routing, DVFS, and power-gating, it is difficult to determine which techniques are suitable for a specific multi/many-core NoC design while meeting the application's requirements (e.g., real-time

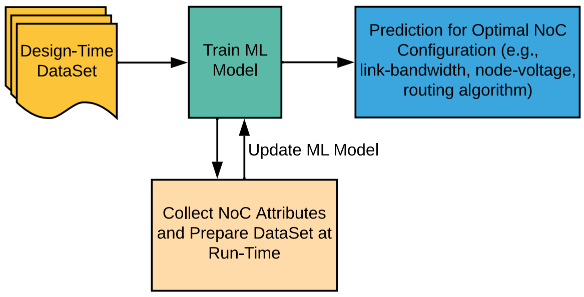
Design-Time DataSet
Train ML Model
Prediction for Optimal NoC Configuration (e.g., link-bandwidth, node-voltage, routing algorithm)
Update ML Model
Collect NoC Attributes and Prepare DataSet at Run-Time

Fig. 4. ML learning and configuration flow for NoC design and optimization.

deadlines or QoS goals). Data-driven techniques can help to find the complex relationships between design parameters and optimization objectives and find the optimal design for multi/many-core NoC. In the next section, we will discuss an ML driven design and optimization framework to address the above mentioned challenges.

4 MACHINE LEARNING DRIVEN NOC DESIGN AND OPTIMIZATION FRAMEWORK

The optimizers (e.g., IBM CPLEX) and heuristics can take a long time to produce a solution (decision) and may not produce a solution in finite time because of the large design space of many-core systems. They also need to run their solution repetitively to determine the configuration settings of the system, to adapt to changes in tasks/application demands and changes in system objectives/requirements. Intelligent and autonomous ML solutions can determine the required configuration settings (e.g., V/F of nodes, link bandwidth) for the current load conditions based on past experiences. Therefore, data-driven ML approaches are useful for producing solutions proactively for efficient resource management in NoC. ML techniques can find the intricate dependencies between system parameters and predict the optimal trade-offs of parameters for both design-time and run-time decisions. ML decisions can further be integrated into on-chip systems to drive throttling, scheduling, and mapping decisions. The challenge is to select the right learning and prediction solution from a pool of ML techniques based on the various design and optimization objectives (e.g., low-power, high-performance, temperature-aware, fault-tolerant).

ML can deliver accurate configuration across a wide set of relevant workloads and across a wide range of heterogeneous targets. An ML agent can learn at design-time from statically generated datasets and learn at run-time (online) using measurements from hardware performance counters and sensors. During online learning, an ML agent can dynamically learn how its actions (policies) affect the NoC based system and readjust its model accordingly at a fixed interval. A learning flow for NoC resource monitoring and configuration using an ML model is presented in Figure 4. However, an ML model should use the minimum number of architecture counters, such as floating-point operations per second and memory access, to minimize hardware and run-time overheads.

4.1 Machine Learning Phases

An ML solution typically has four phases for learning, validation, and prediction, as outlined below.

- *Dataset Creation:* The first step of ML is to create a training dataset using raw data (e.g., system statistics). The training dataset consists of selected input features and action (predictions) pairs. The input features of our ML model for multi/many-core NoC can include task computation and communication demands, link communication capacity and utilization, node computation capacity and utilization, node (core and router) and link power (static

Table 3. Example Training DataSet for NoC Design and Optimization using ML Model

Input Features								Actions	
Computation	Communication	Deadline	Node Utilization	Link Utilization	Power Consumption	Thermal Consumption	Execution Time	Node Voltage	Link Width
Cmp1	Comm2	TD1	NU1	LU11	PW1	TH1	EX1	Low	LW2
Cmp2	Comm1	TD2	NU2	LU2	PW2	TH2	EX2	Medium	LW1
Cmp3	Comm3	TD1	NU3	LU2	PW3	TH3	EX3	Medium	LW3
Cmp2	Comm2	TD1	NU1	LU3	PW2	TH2	EX2	Medium	LW2

and dynamic) and thermal consumption, router buffer utilization, latency, throughput, and others. The predictions from the ML agent can include V/F-levels of nodes and links, link bandwidth, and so on. An example training set for multi/many-core NoC-based systems is shown in Table 3. In the training set, the inputs are computation/communication demands and deadline requirements of the tasks, node and link utilization of NoC, power and thermal consumption, and task execution time, where the NoC configuration actions are node voltage and link bandwidth setting. Computation, communication, and deadlines are assigned in different sets with a range of values, e.g., Cmp1 = [0–1000 Flops], Comm1 = [0–100000 bps], and TD1 = [0–10 ns]. Node and link utilization are assigned in different sets with a range of values, for example, NU1 = [1–30%], NU2 = [30–60%], NU3 = [60–80%], and NU4 = [80–100%]. Power and thermal consumption are assigned in different sets with a range of values, for example, PW1 = [1–100 mW] and TH1 = [1–45 °C]. Node voltage is labeled with low, medium, and high voltages from the voltage sets, where the voltage sets are: low = [0.5–0.7 V], medium = [0.8–1.0 V], and high = [1.1–1.2 V]. Link width can have many different width settings as LW1, LW2, LW3 . . . . LWn. Table 3 can be extended to include more NoC features and output actions for an ML model.

- *Training (Learning) Phase:* The training phase uses the training dataset to train the ML model to learn the weight vector of features (e.g., NoC buffer and link utilization), which indicates the importance of particular features to predict the output.
- *Validation Phase:* The validation phase is used to select an ML model from multiple available models, such as models with different V/F ranges.
- *Prediction (Inference) Phase:* The inference phase uses the trained ML model to perform prediction, such as V/F-level decisions, on new test data in real systems.

4.2 Machine Learning Solutions and Categories for NoC

ML algorithms produce several types of solutions for its application in multi/many-core NoCs, as follows. **Classification** identifies to which category an object belongs. For example, ML can classify the type of tasks as computation or communication intensive for effective mapping and configuration in NoC. **Regression** predicts a continuous-valued attribute associated with an object. For example, linear regression can be used for power and thermal predictions in NoC. **Clustering** is the automatic grouping of similar objects into sets. Clustering can be used to identify abnormal behaviour, such as a security attack from abnormal packets, in many-core systems. In **reinforcement learning (RL)**, feedback (training data) is given in the form of rewards and punishments to reinforce actions, for example, V/F configuration decisions. Preprocessing (feature normalization and/or scaling) can be applied to improve the ML solution, as it normalizes NoC features with different units to the same magnitudes and handles highly varying magnitudes.

4.3 Stages of Machine Learning Solution

A sample of the ML algorithmic stages for configuring a multi/many-core NoC-based system is presented in Algorithm 1. First, we need to define the input and output of the ML-enabled system. Then, the data collection stage collects the statistics of the NoC resources and the data

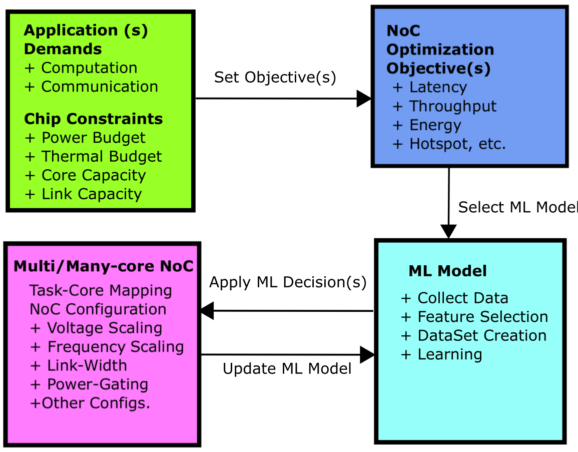

Fig. 5. ML enabled multi/many-core NoC design and optimization framework.

preprocessing stage scales and normalizes the data (depending on the different kinds of features). Secondly, select an ML model and train it using the collected datasets. Then, a part of the dataset is used to validate the performance of the ML model. Finally, apply the trained model to predict the task-assignment and NoC configuration using real-time data from NoC. The overall process of applying ML techniques for multi/many-core NoC design and optimization is demonstrated in Figure 5.

ALGORITHM 1: ML Algorithm for Multi/Many-Core NoC Design and Optimization

```
input  : Tasks Demands, Resource Capacity and Utilization, Power and Thermal Budgets.
output : Best possible NoC configurations corresponding to the input features.
1. Collect the statistics of NoC resources.
2. Preprocess the input features for normalizing the data.
3. Select the ML classifier or regressor model.
4. Train the ML model using a training dataset.
5. Evaluate the ML model using a validation dataset and tune the parameters of the ML model.
6. Feed the new (preprocessed) inputs to the trained classifier or regressor for predicting the
classification or regression.
7. Output of classifier or regressor is the task-to-node assignment, link-width, and node-voltage for
optimal NoC configuration within power and thermal budgets.
8. Update the ML model using prediction error, and the process continues.
```

4.4 Machine Learning based Large-Scale NoC Framework

In a large-scale NoC, it is not feasible to implement a centralized ML agent monitoring and decision framework due to the exponential increase in communication delay and dataset size. Decision delay increases as every node has to communicate with the centralized agent for local configuration decisions of routers and links, where communication latency increases with the distance from the centralized ML agent. The dataset size for ML agent training increases because of the different configuration combinations of various nodes and links, as shown in Table 3. Therefore, distributed ML techniques are needed for large-scale NoC design and optimization. A sample distributed cluster-based ML agent framework is presented in Figure 6. A cluster agent monitors and configures the resources within the corresponding cluster. The cluster size can be decided based on the size of

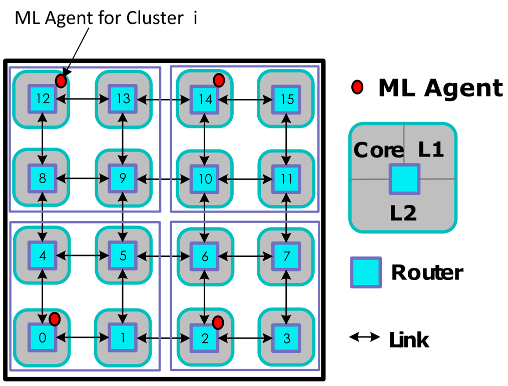

Fig. 6. Distributed cluster-wise ML-agent based large-scale NoC optimization.

the system (e.g., number of cores), NoC topology (e.g., 2D Mesh, concentrated mesh, 3D), and the available hardware resources for ML integration. The cluster-wise ML implementation reduces the hardware overhead for effective implementation in NoC.

5 NEURAL NETWORKS FOR NOC DESIGN AND OPTIMIZATION

We propose an NN based classification solution using the generalized ML framework in Section 4 for NoC V/F and link-width configurations. An NN generates a vector of classification scores (ranging from 0 to 1) for possible NoC configuration actions. As shown in Figure 7, the proposed approach uses fully connected NNs with four layers: an input layer with two neurons, two hidden layers with four neurons per layer, and an output layer with four neurons. The input layer contains two neurons for two features: % of buffer utilization and % of link utilization. Link and buffer utilization are calculated from the communication traffic (flits on the buffer and link) in the NoC and the capacity of the NoC resources (number of buffers and links). These two features can successfully indicate the communication demand at the router and associated links to decide link-width and node V/F settings. The number of features is kept to a minimum to decrease the amount of time needed to calculate predicted values at run-time and reduce the hardware overhead of NN. Also, the features are selected by using the information already present at each router. The output layer of the NN model has four nodes – one for each link-width and voltage-level configuration. Link width and node voltage-level are used as pairs for the four outputs of NNs: .8V/128-bit, .9V/256-bit, 1V/384-bit, and 1.1V/512-bit. The classification pair with the highest value is chosen for NoC configuration at a time. Four layers for an NN and four neurons per hidden layer are chosen empirically as they provide sufficient accuracy with lower hardware overhead. However, the effective number of hidden layers and number of neurons per hidden layer depends on the number of neurons in the input layer and the number of neurons in the output layer. So, the number of hidden layers and number of neurons per hidden layer might need to be changed for different problems (such as a change in the number of features and/or a change in the number of outputs).

The *sigmoid* and $softmax$ activation functions are used at the neurons of the hidden and output layers, respectively.

The activation value ranges from 0 to 1 for the *sigmoid* function in the hidden layers. For multiclass classification, the $softmax$ function is used at the output layer neurons to highlight the largest value for a voltage and link-width classification and to suppress classification values significantly below the maximum value. The four output values range from 0 to 1 after applying the $softmax$ activation function. The output values indicate the probability of the corresponding

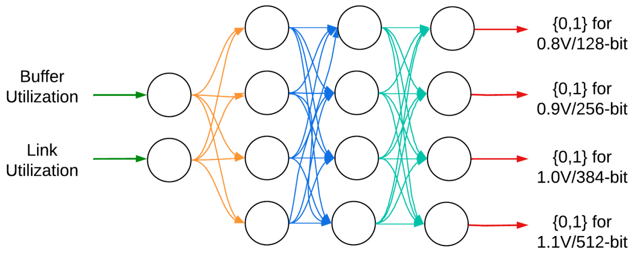

Fig. 7. NoC features and actions for voltage-level and link-width configuration using an NN.

link-width and voltage-level configuration, as the sum of all the four output values is 1. The actual output from the NN will be some continuous numbers, which are then converted to numbers in the interval $\{0, 1\}$. The $softmax$ function highlights the largest value by suppressing other values, and the largest value indicates the corresponding predicted link-width and voltage-level configuration from the NNs.

The θ parameters, known as weights, decide a prediction made by an NN using the input parameters from NoC statistics. An NN is trained with input features and target link-width and node voltage-level data to learn the θ parameters. An NN agent is trained incrementally in a fixed interval, namely online learning, to improve its performance. The backpropagation algorithm [75] is used to learn the θ parameters. The backpropagation aims to minimize the cost function J_θ by adjusting the weights in an NN. As shown in Equation (1), the cost function J_θ is the sum of the error differences between predicted values $y_k^{'}$ and target values y_k for each class. The cost function is then summed up for all samples i in the training data set of 50 entries. The newly collected data replaces older data in the training data set. The backpropagation algorithm uses the gradient descent method to compute the gradient of the cost function. The gradient shows how much the parameters θ need to be changed to minimize the cost function. The smaller cost indicates the higher classification accuracy of an NN.

$$J_\theta = -1/m\left(\sum_{i=1}^{m}\sum_{k=1}^{4} y_k^{(i)}\log(y_k^{'}) - (1 - y_k^{(i)})\log(1 - y_k^{'})\right) \tag{1}$$

where i represents the index of the sample, $y^{'}$ is the predicted outcome, y is the sample output value, k is the class number, and m is the number of samples.

In each training step, the backpropagation algorithm computes the gradient with respect to the θ parameters of the NN. The backpropagation algorithm computes the gradient one layer at a time and iterates backward from the last layer to update (correct the weights) and to learn θ parameters of NNs. The learning rate for weight updates is set to 0.3 to speed up the learning. The momentum term is set to 0.5 to give equal importance to both current gradient and past gradients to avoid noise in the gradients (to ensure stability and to avoid local optima). The goal of learning θ parameters is to correctly predict values for different configuration decisions.

5.1 Neural Networks Results Comparison for NoC

We compare the neural networks-based NoC configuration solution with a traditional non-ML solution [73], which proposed a heuristic-based solution, and with two ML-based solutions [70, 76] in Table 2. Same training and testing ratio is used for a fair comparison when comparing two ML-enabled solutions. We evaluate NoC configuration models using COSMIC benchmark suite [31] and synthetic traffic patterns. COSMIC benchmark suite contains five applications: **Face Recognition (face)**, Cifar, **Ultrasound (ultra)**, **RS-Decoder (RSD)**, and **RS-Encoder**

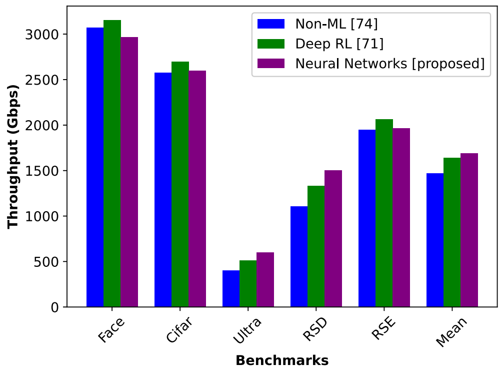

Fig. 8. Throughput comparison under COSMIC in 64-core NoC.

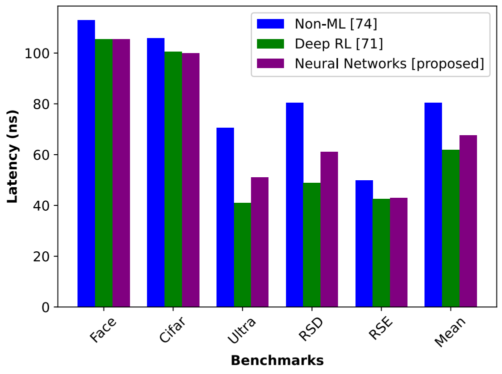

Fig. 9. Latency comparison under COSMIC in 64-core NoC.

(RSE). Four synthetic traffic patterns are used for comparison (with [76]) as [76] only has results for those traffic patterns. GARNET [2] interconnection network on gem5 [30] platform is used to evaluate NoCs performance in terms of throughput and latency. DSENT [29] tool is integrated with gem5 to evaluate the energy consumption and area overheads. A 64-core system connected through an 8 × 8 2D-Mesh NoC is simulated in GARNET, where each core has a router for communication. An NN agent is integrated with each router of NoC for a distributed ML-based NoC configuration solution. In [70], concentrated 2D-Mesh NoC was used, where a router (and an RL agent) supports four cores. Heterogeneous link bandwidths in gem5 are modeled by modifying link latencies to mimic bandwidths in GARNET. Four different link widths are used for link configurations. 128-bit width, 256-bit width, 384-bit width, and 512-bit width links have configured latency of 1, 0.5, 0.33, and 0.25 cycles, respectively, in our simulation. Four different voltage-levels are used for node configurations: 0.8V, 0.9V, 1.0V, and 1.1V. XY-routing is used to route packets in NoC. We run the simulations in fixed-pipeline mode for 1,000 cycles with the maximum number of packets to be injected by each node set at 50,000. NN agents are trained every 50 cycles throughout the simulation. Other NoC configuration settings include ALPHA **instruction set architecture (ISA)**, 3 **virtual networks (VNs)**, 4 **virtual channels (VCs)** per VN, and 4 buffers per VC. We use two features at the router for link-width and node voltage-level configuration predictions: percentage of buffer utilization and percentage of link utilization.

As shown in Figures 8 and 9, the NN-based NoC configurable solution improves both latency and throughput by 15% on average (up to 30%) compared to those of the non-ML based load-balanced solution [73]. The improvement comes from the pro-active and quick configuration of NoC based on the application demands. Furthermore, energy consumption reduces by 6% on average (up to 11%) in the NN-based configuration solution, as shown in Figure 10. This shows the benefits of ML for dynamic and proactive configuration of NoC. The deep-RL-based solution [70] implements a **concentrated mesh (CMesh)** NoC architecture, where four cores are integrated with a router and an RL agent is integrated with a router. The NN-based solution improves throughput by 2.7% (on average) compared to [70], while the NN-based solution has 8% (on average) higher latency than that of [70]. The NN-based solution consumes almost the same amount of energy as the deep RL-based solution [70]. But the proposed NN-based solution for 2D-Mesh NoC has more NN-agents compared to the number of RL agents in [70] because of the use of CMesh NoC in [70]. The performance and energy efficiency could be further improved if the ML implementation includes more NoC configuration parameters, such as buffer sizes and routing algorithms, for design and optimization.

The proposed NN-based solution significantly outperforms [76] in terms of latency, as shown in Figure 11, achieving 400% lower latency for synthetic traffic patterns. The NN-based solution

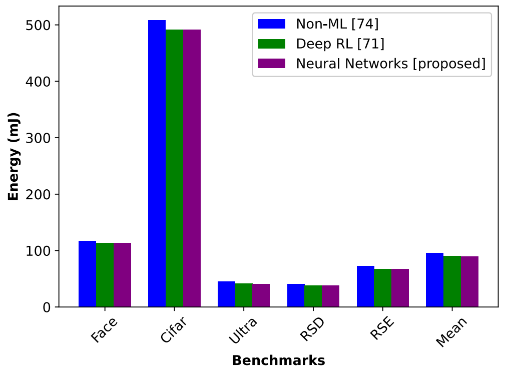

Fig. 10. Energy comparison under COSMIC in 64-core NoC.

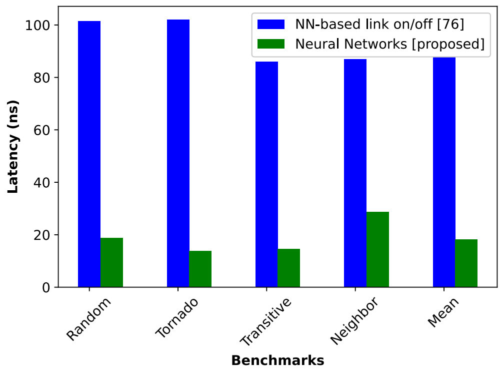

Fig. 11. Latency comparison under Synthetic Traffic in 64-core NoC.

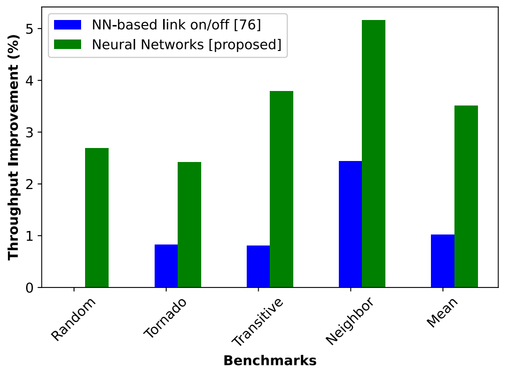

Fig. 12. Throughput comparison under Synthetic Traffic in 64-core NoC.

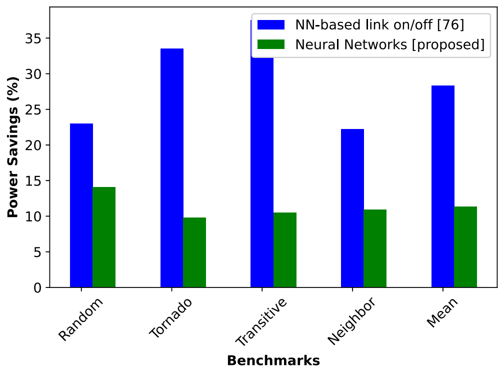

Fig. 13. Power savings comparison under Synthetic Traffic in 64-core NoC.

also achieves a higher throughput improvement compared to that of [76], as shown in Figure 12. While [76] provides 28% power savings compared to a static baseline solution, our proposed solution provides 10% power savings compared to the dynamic non-ML-based solution [73], as shown in Figure 13.

The main reason for the higher power savings and lower performance in [76] is the extensive power-gating of links. Although V/F is scaled in our work, power-gating (complete turn-off) is not significantly applied to ensure higher communication performance of the applications.

5.2 Hardware Overhead for Neural Networks

For input layer to hidden layer connections in the proposed NN (as shown in Figure 7), we have a total of 8 multiplications, 4 additions, and 4 comparisons. For hidden layer to hidden layer connections, we have a total of 16 multiplications, 12 additions, and 4 comparisons. For hidden layer to output layer connections, we have a total of 16 multiplications, 12 additions, and 4 comparisons. This equates to a total of 40 multiplications, 28 additions, and 12 comparisons to gather the features, compute output values, and select the voltage-level and link-width. In $45nm$, the energy cost of a single 16-bit floating point addition is estimated to be 0.4 pJ (pico-joule), and the area cost is 1360 um^2 [39]. The energy cost of a multiply is estimated to be 1.1 pJ, and the area cost is 1640 um^2 in $45nm$ [39]. Therefore, for a single NN, the total energy overhead is 60.0 pJ, and the total area overhead is 0.12 mm^2. As a single NN is integrated with each router in 8 × 8 2D-Mesh NoC, the total energy and area overheads are 3840 pJ and 7.68 mm^2, respectively. The NoC area overhead (excluding NNs) for 64-core (one router per core) is 28.46 mm^2. Therefore, the hardware area overhead for NNs is 21.3% of the total NoC area overhead, compared to 23% overhead for deep RL agents in [70]. Furthermore, the deep RL solution in [70] has a lower number of ML agents in routers because of the CMesh NoC implementation, but it has higher ML overheads per ML agent compared to the proposed NN solution. The hardware overhead in [76] is significantly lower (4%) due to the use of a cluster-wise ML agent (one agent for a 4 × 4 NoC) rather than a per-router ML agent and because they use NNs with only one hidden layer (our work uses two hidden layers). As shown in this work, the results for applying ML in NoC are promising. We predict that innovations in ML techniques will improve their effectiveness in applications of ML in multi/many-core NoC. However, ML has some limitations and challenges (including overhead of ML) for effective application as presented in the next section.

6 CHALLENGES FOR MACHINE LEARNING MODELS IMPLEMENTATION IN NOC

The challenges with the applications of ML driven design and optimization in NoC-based multi/many-core architectures are discussed in this section.

6.1 Hardware for Machine Learning Solution

Additional hardware is needed to collect and store the data/statistics and for computation (of ML) to apply ML for design and optimization in multi/many-core architectures. For computation, ML algorithms, such as classification and regression algorithms, need to record the data in hardware counters and then apply arithmetic operations (e.g., addition and multiplication) to the features (data). For example, the implementation of an NN needs the following hardware for its implementation: registers/counters, buffers, adders, and multipliers. The features for an ML model need to be recorded by using architectural hardware counters. Hardware is needed to record the data of architectures and networks for applying an ML model. Additional buffers are needed to store the data, for example, to store the connection weights of NNs parameters, and store the sample data for training NNs. A many-core system already has a large number of resources in the system within a limited space. Besides area overhead, additional hardware for ML consumes energy/power. Therefore, it is very important to design an ML solution carefully for design and optimization in multi/many-core NoC architectures. The proposed solution incurs comparatively lower hardware overhead for distributed per-router ML implementation in a 64-core NoC, as shown in Section 5.2. Cluster-wise ML or centralized ML will have lower hardware overhead compared to per-router ML, but they would degrade performance and/or be ineffective for many-core systems.

6.2 Availability and Creation of Datasets

Datasets are created for both training and testing of ML models. The datasets contain data on NoC components, including power/thermal consumption and utilization of resources. The datasets can be created using linear programming solvers and/or heuristics depending on the size of the NoC and the application. The dataset consists of two parts: input and output. The inputs of the dataset include application ID, task ID, task computation demands, task communication demands, task deadline, and constraints on NoC resources. The outputs of the dataset include node (core/router)-wise power and thermal consumption, link and node capacity and utilization, and node and link V/F-levels. An ML model requires a large amount of training data. If we do not have sufficient data, we cannot develop a good ML model, for example, a model with low generalization error, to solve a given problem or type of problem on new unseen data. If new unseen data does not resemble training data, then the ML model may not work. In this proposed work, the data creation and collection processes are efficient, and data is created at the router, allowing the NN agent at the router to directly access the data.

6.3 Training Time and Computation Complexity

ML systems are greedy as they require huge sets of training data to improve their accuracy. Training an ML model to high accuracy requires a large amount of computation and long time. Training time for ML computation is one of the major factors that must be considered when adapting ML in a practical system. Many-core systems can have large datasets for node (core and router) and link configuration. Many useful ML models take a significant amount of time to train. For example, in RL, training a deep Q-learning model and achieving significant results on a large-scale NoC system can take many hours, even on high-performance servers. Therefore, innovation in fast computing (e.g., quantum computing) is needed for effective adoption of ML techniques. As the training time for most ML algorithms increases with dataset size in multi/many-core NoCs, trade-off analysis needs to be considered for online and offline training to handle the dynamic nature of applications and systems. The proposed solution incurs significantly lower training time, in the range of nanoseconds (in gem5/garnet), due to the lower number of layers and neurons per layer, as well as the the incremental online learning (training) with a small amount of data (50 entries).

6.4 Online Learning

ML techniques may require lengthy offline (batch) training for proper prediction in real-time systems. Due to dynamic service and performance requirements in multi/many-core systems, these systems need to improve the learning model and policies at run-time to accurately predict the optimal decisions. The system manager(s) need ways to collect run-time performance, power, reliability, and utilization information from the system and incorporate this information into ML techniques to improve the model for accurate prediction. The proposed solution uses an online learning technique to train the NN agents at a fixed interval (50 cycles) using data collected from the NoC.

6.5 Data Storage

The learned parameters in ML models need to be stored for applying learning configuration in the system to optimize its behavior. For example, learned weights in NNs need to be stored and updated in next iteration to handle the dynamic behavior of the system, or state-action-reward-next-state example needs to be stored for training of RL agents (for self-adaptive agents). As memory size is limited in on-chip systems, efficient ML models in terms of data storage (e.g., fewer input features) are needed for resource management and configuration in multi/many-core on-chip systems. In this work, data (weights of an NN agent) is stored at the on-chip router where the NN agents reside. Therefore, NN agents do not have to travel to off-chip memory (or any other part of the chip) to access the weights and update them with new training data at the router.

6.6 Error Diagnosis and Verification

ML models, such as NNs, are harder to debug because of their black-box nature. For example, given a dataset and a multi/many-core NoC architecture, there could be two NNs with different weights but the same result. ML is also prone to hidden and unintentional biases depending on the data provided to train them. This raises doubts about the reliability and biases of ML. ML can also make errors, but diagnosing (and correcting) them can be difficult because it requires going through the underlying complexities of the algorithms and associated processes. Another limitation of ML is the lack of verification tools. ML explainability is one of the major obstacles to its implementation in systems. ML deals with statistical correctness based on data, and they (ML systems) may not understand context. In situations that are not included in the historical/training data, it will be difficult to prove with complete certainty that the predictions made by an ML system are suitable in all scenarios. In this proposed solution, NN agents are becoming more trained (weights are changing) with more data, which shows that NN agents are fixing errors to improve their predictions.

6.7 Scalability

An ML solution needs to be trained and provide a solution (inference) quickly to adapt to changing system requirements. With the increase in system and NoC size, the dataset size increases quickly due to the increase in cores, routers, links, and so on. The training and inference times of an ML solution also increase with the size of the dataset. Therefore, an ML solution needs to be scalable in terms of the size of multi/many-core systems and datasets. The proposed solution is scalable because of the distributed NN agents, where the traveling distance of the decision from the NN agent does not increase with the NoC size. Here an NN agent can be trained using local router data and make decisions at the corresponding router.

6.8 Autonomous Solution

The objective of using ML techniques for multi/many-core NoCs is to achieve a self-adaptive, autonomous solution. ML algorithms need to be fed with correct information, and the predictions of ML algorithms need to be interpreted and verified correctly for further information processing. Manual intervention is needed if the system encounters a new situation (e.g., extreme values for the features, new features) that were not experienced before. Thus, achieving perfect autonomy may not be possible with ML techniques. Our proposed solution is fully autonomous as the agents are automatically trained using router data at a fixed interval.

ML techniques can provide intelligent, efficient, proactive, and autonomous solutions for NoC-based multi/many-core systems. However, ML adaptation faces aforementioned challenges for effective implementation in multi/many-core architectures.

7 CONCLUSION

In this article, we explored the challenges in design-time and run-time optimization in NoC-based multi/many-core architectures. We also explored the existing machine learning solutions for multi/many-core NoC optimization. Then, we proposed a generalized machine learning-based framework to address the design and optimization challenges of traditional heuristic and mathematical solutions in large-scale systems. We presented different types of machine learning techniques for implementation in multi/many-core NoC and proposed and implemented neural networks-enabled solution using the generalized framework. We demonstrated several simulation results showing the effectiveness of the neural networks solution over a traditional non-machine-learning solution and over two machine learning solutions for NoC optimization. Finally, we discussed the limitations and challenges of applying machine learning techniques in multi/many-core on-chip systems design and optimization. We hope for a promising future in which machine learning helps to solve the design and optimization challenges in large-scale NoC-based systems by providing intelligent and autonomous solutions.

REFERENCES

[1] José L. Abellán, Chao Chen, and Ajay Joshi. 2016. Electro-photonic NoC designs for kilocore systems. *J. Emerg. Technol. Comput. Syst.* 13, 2, Article 24 (Nov. 2016), 25 pages. https://doi.org/10.1145/2967614

[2] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha. 2009. 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 (ISPASS'09)*. 33–42. https://doi.org/10.1109/ISPASS.2009.4919636

[3] Yuxin Bai, Victor W. Lee, and Engin Ipek. 2017. Voltage regulator efficiency aware power management. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*. 825–838. https://doi.org/10.1145/3037697.3037717

[4] A. Banerjee, R. Mullins, and S. Moore. 2007. A power and energy exploration of network-on-chip architectures. In *First International Symposium on Networks-on-Chip (NOCS'07)*. 163–172. https://doi.org/10.1109/NOCS.2007.6

[5] L. Benini and D. Bertozzi. 2005. Network-on-chip architectures and design methods. *IEE Proceedings on Computers and Digital Techniques* 152, 2 (March 2005), 261–272. https://doi.org/10.1049/ip-cdt:20045100

[6] Keren Bergman et al. 2008. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. (2008).

[7] Stefano Bertozzi, Andrea Acquaviva, Davide Bertozzi, and Antonio Poggiali. 2006. Supporting task migration in multi-processor systems-on-chip: A feasibility study. In *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings (DATE'06)*. European Design and Automation Association, 3001 Leuven, Belgium, 15–20. http://dl.acm.org/citation.cfm?id=1131481.1131488.

[8] Filipe Betzel, Karen Khatamifard, Harini Suresh, David J. Lilja, John Sartori, and Ulya Karpuzcu. 2018. Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems. *ACM Comput. Surv.* 51, 1, Article 1 (Jan. 2018), 32 pages. https://doi.org/10.1145/3145812

[9] B. Bohnenstiehl, A. Stillmaker, J. J. Pimentel, T. Andreas, B. Liu, A. T. Tran, E. Adeagbo, and B. M. Baas. 2017. KiloCore: A 32-nm 1000-processor computational array. *IEEE Journal of Solid-State Circuits* 52, 4 (April 2017), 891–902. https://doi.org/10.1109/JSSC.2016.2638459

[10] Luciano Bononi and Nicola Concer. 2006. Simulation and analysis of network on chip architectures: Ring, spidergon and 2D mesh. In *Proceedings of the Conference on Design, Automation and Test in Europe: Designers' Forum (DATE'06)*. European Design and Automation Association, 3001 Leuven, Belgium, 154–159. http://dl.acm.org/citation.cfm?id=1131355.1131388.

[11] S. Borkar. 1999. Design challenges of technology scaling. *IEEE Micro* 19, 4 (July 1999), 23–29. https://doi.org/10.1109/40.782564

[12] S. Borkar. 2005. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro* 25, 6 (2005), 10–16. https://doi.org/10.1109/MM.2005.110

[13] Shekhar Borkar and Andrew A. Chien. 2011. The future of microprocessors. *Commun. ACM* 54, 5 (May 2011), 67–77. https://doi.org/10.1145/1941487.1941507

[14] R. Boyapati, J. Huang, P. Majumder, K. H. Yum, and E. J. Kim. 2017. APPROX-NOC: A data approximation framework for network-on-chip architectures. In *Proceedings of International Symposium on Computer Architecture (ISCA'17)*. 666–677. https://doi.org/10.1145/3079856.3080241

[15] E. Carvalho, N. Calazans, and F. Moraes. 2007. Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs. In *Proceedings of IEEE/IFIP International Workshop on Rapid System Prototyping (RSP'07)*. 34–40. https://doi.org/10.1109/RSP.2007.26

[16] M. F. Chang, J. Cong, A. Kaplan, C. Liu, M. Naik, J. Premkumar, G. Reinman, E. Socher, and S. Tam. 2008. Power reduction of CMP communication networks via RF-interconnects. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. 376–387. https://doi.org/10.1109/MICRO.2008.4771806

[17] L. Chen and T. M. Pinkston. 2012. NoRD: Node-router decoupling for effective power-gating of on-chip routers. In *Proc. IEEE/ACM International Symposium on Microarchitecture*. 270–281. https://doi.org/10.1109/MICRO.2012.33

[18] Chen-Ling Chou and R. Marculescu. 2007. Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels. In *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'07)*. 161–166.

[19] William Dally and Brian Towles. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[20] W. J. Dally and H. Aoki. 1993. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Trans. Parallel Distrib. Syst.* 4, 4 (April 1993), 466–475. https://doi.org/10.1109/71.219761

[21] William J. Dally and Brian Towles. 2001. Route packets, not wires: On-chip inteconnection networks. In *Proceedings of the 38th Design Automation Conference*. 684–689. https://doi.org/10.1145/378239.379048

[22] M. Daneshtalab, M. Ebrahimi, P. Liljeberg, J. Plosila, and H. Tenhunen. 2012. Memory-efficient on-chip network with adaptive interfaces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 1 (Jan. 2012), 146–159. https://doi.org/10.1109/TCAD.2011.2160348

[23] S. Das, J. R. Doppa, D. H. Kim, P. P. Pande, and K. Chakrabarty. 2015. Optimizing 3D NoC design for energy efficiency: A machine learning approach. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'15)*. 705–712. https://doi.org/10.1109/ICCAD.2015.7372639

[24] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. 1974. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits* 9, 5 (1974), 256–268. https://doi.org/10.1109/JSSC.1974.1050511

[25] D. DiTomaso, T. Boraten, A. Kodi, and A. Louri. 2016. Dynamic error mitigation in NoCs using intelligent prediction techniques. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783734

[26] Dominic DiTomaso, Ashif Sikder, Avinash Kodi, and Ahmed Louri. 2017. Machine learning enabled power-aware network-on-chip design. In *Proceedings of IEEE Design, Automation and Test in Europe (DATE'17)*. 1354–1359. http://dl.acm.org/citation.cfm?id=3130379.3130699.

[27] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel. 2002. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[28] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2012. Dark silicon and the end of multicore scaling. *IEEE Micro* 32, 3 (May 2012), 122–134. https://doi.org/10.1109/MM.2012.17

[29] Chen Sun et al. 2012. DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proc. NOCS*. 201–210. https://doi.org/10.1109/NOCS.2012.31

[30] Nathan Binkert et al. 2011. The Gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (2011), 1–7. https://doi.org/10.1145/2024716.2024718

[31] Zhe Wang et al. 2014. A case study on the communication and computation behaviors of real applications in NoC-based MPSoCs. In *Proc. ISVLSI*. 480–485. https://doi.org/10.1109/ISVLSI.2014.36

[32] Mohammad Abdullah Al Faruque, R. Krist, and J. Henkel. 2008. ADAM: Run-time agent-based distributed application mapping for on-chip communication. In *Proceedings of ACM/IEEE Design Automation Conference (DAC'08)*. 760–765. https://doi.org/10.1145/1391469.1391664

[33] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila. 2013. Smart hill climbing for agile dynamic mapping in many-core systems. In *Proceedings of Design Automation Conference (DAC'13)*. 1–6.

[34] Q. Fettes, M. Clark, R. Bunescu, A. Karanth, and A. Louri. 2019. Dynamic voltage and frequency scaling in NoCs with supervised and reinforcement learning techniques. *IEEE Trans. Comput.* 68, 3 (March 2019), 375–389. https://doi.org/10.1109/TC.2018.2875476

[35] Jose Flich and Davide Bertozzi. 2010. *Designing Network On-Chip Architectures in the Nanoscale Era*. Chapman & Hall/CRC.

[36] Christopher J. Glass and Lionel M. Ni. 1992. The turn model for adaptive routing. *SIGARCH Comput. Archit. News* 20, 2 (April 1992), 278–287. https://doi.org/10.1145/146628.140384

[37] R. Ho, K. W. Mai, and M. A. Horowitz. 2001. The future of wires. *Proc. IEEE* 89, 4 (April 2001), 490–504. https://doi.org/10.1109/5.920580

[38] Simon Holmbacka, Mohammad Fattah, Wictor Lund, Amir-Mohammad Rahmani, Sébastien Lafond, and Johan Lilius. 2014. A task migration mechanism for distributed many-core operating systems. *The Journal of Supercomputing* 68, 3 (June 2014), 1141–1162. https://doi.org/10.1007/s11227-014-1144-7

[39] M. Horowitz. 2014. 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'14)*. 10–14. https://doi.org/10.1109/ISSCC.2014.6757323

[40] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. 2007. A 5-GHz mesh interconnect for a teraflops processor. *IEEE Micro* 27, 5 (Sept. 2007), 51–61. https://doi.org/10.1109/MM.2007.4378783

[41] Mohaddeseh Hoveida, Fatemeh Aghaaliakbari, Ramin Bashizade, Mohammad Arjomand, and Hamid Sarbazi-Azad. 2017. Efficient mapping of applications for future chip-multiprocessors in dark silicon era. *ACM Trans. Des. Autom. Electron. Syst.* 22, 4, Article 70 (June 2017), 26 pages. https://doi.org/10.1145/3055202

[42] Chung-Hsing Hsu and Ulrich Kremer. 2003. Dynamic voltage and frequency scaling for scientific applications. In *Proceedings of the 14th International Conference on Languages and Compilers for Parallel Computing (LCPC'01)*. Springer-Verlag, Berlin, 86–99. http://dl.acm.org/citation.cfm?id=1769331.1769337.

[43] Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan, Victor Zyuban, Hans Jacobson, and Pradip Bose. 2004. Microarchitectural techniques for power gating of execution units. In *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'04)*. 32–37. https://doi.org/10.1109/LPE.2004.1349303

[44] Biresh Kumar Joardar, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2018. Hybrid on-chip communication architectures for heterogeneous manycore systems. In *Proceedings of International Conference on Computer-Aided Design (ICCAD'18)*. Article 62, 6 pages. https://doi.org/10.1145/3240765.3243480

[45] D. C. Juan, Huapeng Zhou, D. Marculescu, and Xin Li. 2012. A learning-based autoregressive model for fast transient thermal analysis of chip-multiprocessors. In *17th Asia and South Pacific Design Automation Conference*. 597–602. https://doi.org/10.1109/ASPDAC.2012.6165027

[46] E. Kakoulli, V. Soteriou, and T. Theocharides. 2012. Intelligent hotspot prediction for network-on-chip-based multicore systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 3 (March 2012), 418–431. https://doi.org/10.1109/TCAD.2011.2170568

[47] A. Kanduri, M. Haghbayan, A. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen. 2015. Dark silicon aware runtime mapping for many-core systems: A patterning approach. In *Proceedings of International Conference on Computer Design (ICCD'15)*. 573–580. https://doi.org/10.1109/ICCD.2015.7357167

[48] Miray Kas. 2012. Toward on-chip datacenters: A perspective on general trends and on-chip particulars. *J. Supercomput.* 62, 1 (2012), 214–226. https://doi.org/10.1007/s11227-011-0703-4

[49] Daya S. Khudia, Babak Zamirai, Mehrzad Samadi, and Scott Mahlke. 2015. Rumba: An online quality management system for approximate computing. In *Proceedings of International Symposium on Computer Architecture (ISCA'15)*. 554–566.

[50] R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu. 2018. Machine learning and manycore systems design: A serendipitous symbiosis. *Computer* 51, 7 (July 2018), 66–77. https://doi.org/10.1109/MC.2018.3011040

[51] Michel A. Kinsy, Shreeya Khadka, and Mihailo Isakov. 2017. PreNoc: Neural network based predictive routing for network-on-chip architectures. In *Proceedings of the on Great Lakes Symposium on VLSI 2017 (GLSVLSI'17)*. ACM, New York, NY, USA, 65–70. https://doi.org/10.1145/3060403.3060406

[52] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel. 2011. DistRM: Distributed resource management for on-chip many-core systems. In *Proceedings of IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'11)*. 119–128. https://doi.org/10.1145/2039370.2039392

[53] Haldun Kufluoglu and Muhammad Ashraful Alam. 2004. A computational model of NBTI and hot carrier injection time-exponents for MOSFET reliability. *Journal of Computational Electronics* 3, 3 (01 Oct. 2004), 165–169. https://doi.org/10.1007/s10825-004-7038-9

[54] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan. 2005. Heterogeneous chip multiprocessors. *IEEE Computer* 38, 11 (Nov. 2005), 32–38. https://doi.org/10.1109/MC.2005.379

[55] R. Kumar, V. Zyuban, and D. M. Tullsen. 2005. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *32nd International Symposium on Computer Architecture (ISCA'05)*. 408–419. https://doi.org/10.1109/ISCA.2005.34

[56] Junghee Lee, Chrysostomos Nicopoulos, Hyung Gyu LEE, and Jongman Kim. 2013. TornadoNoC: A lightweight and scalable on-chip network architecture for the many-core era. *ACM Trans. Archit. Code Optim.* 10, 4, Article 56 (Dec. 2013), 30 pages. https://doi.org/10.1145/2541228.2555312

[57] Hui Li, Huaxi Gu, Yintang Yang, and Xiaoshan Yu. 2013. A hybrid packet-circuit switched router for optical network on chip. *Computers & Electrical Engineering* 39, 7 (2013), 2197–2206. https://doi.org/10.1016/j.compeleceng.2013.08.006

[58] G. De Micheli, C. Seiculescu, S. Murali, L. Benini, F. Angiolini, and A. Pullini. 2010. Networks on chips: From research to products. In *Proceedings of ACM/IEEE Design Automation Conference (DAC'10)*. 300–305. https://doi.org/10.1145/1837274.1837352

[59] M. Modarressi, H. Sarbazi-Azad, and M. Arjomand. 2009. A hybrid packet-circuit switched on-chip network based on SDM. In *Proceedings of Design, Automation Test in Europe Conference Exhibition*. 566–569. https://doi.org/10.1109/DATE.2009.5090728

[60] Thomas Moscibroda and Onur Mutlu. 2009. A case for bufferless routing in on-chip networks. *SIGARCH Comput. Archit. News* 37, 3 (June 2009), 196–207. https://doi.org/10.1145/1555815.1555781

[61] U. Y. Ogras and R. Marculescu. 2006. "It's a small world after all": NoC performance optimization via long-range link insertion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14, 7 (July 2006), 693–706. https://doi.org/10.1109/TVLSI.2006.878263

[62] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu. 2007. Voltage-frequency island partitioning for GALS-based networks-on-chip. In *2007 44th ACM/IEEE Design Automation Conference*. 110–115.

[63] Maurizio Palesi and Masoud Daneshtalab. 2013. *Routing Algorithms in Networks-on-Chip*. Springer Publishing Company, Incorporated.

[64] Partha Pratim Pande, Amlan Ganguly, Kevin Chang, and Christof Teuscher. 2009. Hybrid wireless network on chip: A new paradigm in multi-core design. In *Proceedings of the 2nd International Workshop on Network on Chip Architectures (NoCArc'09)*. ACM, 71–76. https://doi.org/10.1145/1645213.1645230

[65] Partha Pratim Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. 2005. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Trans. Comput.* 54, 8 (Aug. 2005), 1025–1040. https://doi.org/10.1109/TC.2005.134

[66] Vasilis F. Pavlidis and Eby G. Friedman. 2007. 3-D topologies for networks-on-chip. *IEEE Trans. Very Large Scale Integr. Syst.* 15, 10 (Oct. 2007), 1081–1090. https://doi.org/10.1109/TVLSI.2007.893649

[67] Z. Qian, D. C. Juan, P. Bogdan, C. Y. Tsui, D. Marculescu, and R. Marculescu. 2013. SVR-NoC: A performance analysis tool for network-on-chips using learning-based support vector regression model. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE'13)*. 354–357. https://doi.org/10.7873/DATE.2013.083

[68] M. F. Reza. 2020. Reinforcement learning based dynamic link configuration for energy-efficient NoC. In *Proceedings of IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS'20)*. 468–473.

[69] Md Farhadur Reza. 2021. Machine learning for design and optimization challenges in multi/many-core network-on-chip. In *Proceedings of the 14th International Workshop on Network on Chip Architectures (NoCArc'21)*. ACM, 29–34. https://doi.org/10.1145/3477231.3490427

[70] Md Farhadur Reza. 2022. Deep reinforcement learning enabled self-configurable networks-on-chip for high-performance and energy-efficient computing systems. *IEEE Access* 10 (2022), 65339–65354. https://doi.org/10.1109/ACCESS.2022.3182500

[71] Md Farhadur Reza and Tung Thanh Le. 2021. Reinforcement learning enabled routing for high-performance networks-on-chip. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'21)*. 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401790

[72] M. F. Reza, T. T. Le, B. De, M. Bayoumi, and D. Zhao. 2018. Neuro-NoC: Energy optimization in heterogeneous many-core NoC using neural networks in dark silicon era. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'18)*. 1–5. https://doi.org/10.1109/ISCAS.2018.8351580

[73] M. F. Reza, D. Zhao, and M. Bayoumi. 2018. Power- thermal aware balanced task-resource co-allocation in heterogeneous many CPU-GPU cores NoC in dark silicon era. In *Proceedings of 31st IEEE International System-on-Chip Conference (SOCC'18)*. 260–265. https://doi.org/10.1109/SOCC.2018.8618557

[74] M. F. Reza, D. Zhao, Hongyi Wu, and M. Bayoumi. 2018. HotSpot-aware task-resource co-allocation for heterogeneous many-core networks-on-chip. *Computers & Electrical Engineering* 68, C (2018), 581–602.

[75] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. Neurocomputing: Foundations of research. MIT Press, Chapter Learning Representations by Back-propagating Errors, 696–699. http://dl.acm.org/citation.cfm?id=65669.104451.

[76] Andreas G. Savva, Theocharis Theocharides, and Vassos Soteriou. 2012. Intelligent on/off dynamic link management for on-chip networks. *Journal of Electrical and Computer Engineering (JECE)*, Article 107821 (2012), 12 pages. https://doi.org/10.1155/2012/107821

[77] Muhammad Shafique, Siddharth Garg, Jörg Henkel, and Diana Marculescu. 2014. The EDA challenges in the dark silicon era: Temperature, reliability, and variability perspectives. In *Proceedings of ACM/IEEE Design Automation Conference (DAC'14)*. Article 185, 6 pages. https://doi.org/10.1145/2593069.2593229

[78] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. 2004. The impact of technology scaling on lifetime reliability. In *International Conference on Dependable Systems and Networks*. 177–186. https://doi.org/10.1109/DSN.2004.1311888

[79] M. B. Taylor. 2013. A landscape of the new dark silicon design regime. *IEEE Micro* 33, 5 (Sept. 2013), 8–19. https://doi.org/10.1109/MM.2013.90

[80] Michael Bedford Taylor et al. 2002. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro* 22, 2 (March 2002), 25–35. https://doi.org/10.1109/MM.2002.997877

[81] K. Wang, A. Louri, A. Karanth, and R. Bunescu. 2019. High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE'19)*. 1166–1171. https://doi.org/10.23919/DATE.2019.8714869

[82] Ke Wang, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. 2019. IntelliNoC: A holistic design framework for energy-efficient and reliable on-chip communication for manycores. In *Proceedings of International Symposium on Computer Architecture*. 589–600. https://doi.org/10.1145/3307650.3322274

[83] Yao Xiao, Shahin Nazarian, and Paul Bogdan. 2019. Self-optimizing and self-programming computing systems: A combined compiler, complex networks, and machine learning approach. *IEEE TVLSI* 27, 6 (2019), 1416–1427. https://doi.org/10.1109/TVLSI.2019.2897650

[84] Y. Xu, B. Zhao, Y. Zhang, and J. Yang. 2010. Simple virtual channel allocation for high throughput and high frequency on-chip routers. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. 1–11. https://doi.org/10.1109/HPCA.2010.5416640

[85] Y. Yao and Z. Lu. 2016. Memory-access aware DVFS for network-on-chip in CMPs. In *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE'16)*. 1433–1436.

[86] Jieming Yin, Yasuko Eckert, Shuai Che, Mark Oskin, and Gabriel H. Loh. 2018. Toward more efficient NoC arbitration: A deep reinforcement learning approach. In *International Workshop on AI-assisted Design for Architecture (AIDArc), Held in Conjunction with ISCA'18*.

[87] Yuan Zhou, Hanyu Wang, Jieming Yin, and Zhiru Zhang. 2021. Distilling arbitration logic from traces using machine learning: A case study on NoC. In *2021 58th ACM/IEEE Design Automation Conference (DAC'21)*. 55–60. https://doi.org/10.1109/DAC18074.2021.9586301

Received 1 March 2022; revised 30 December 2022; accepted 13 March 2023