2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) | 978-1-6654-6499-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/MCSOC57363.2022.00061

Turn-aware Application Mapping using Reinforcement Learning in Power Gating-enabled Network on Chip

Mohammadmehdi Shammasi, Mohammad Baharloo, Meisam Abdollahi, Amirali Baniasadi
Department of Electrical and Computer Engineering
University of Victoria
Victoria, Canada
m.shammasi@gmail.com, amirbaharloo@uvic.ca, meisam.abdolahi@ut.ac.ir, amiralib@ece.uvic.ca

***Abstract*—As the backbone for many-core chips, Network-on-chips (NoCs) consume a significant share of total chip power. As a result, decreasing the power consumption in these components can reduce the total chip's power significantly. NoC's routers can be powered down using power-gating, a promising technique for reducing static power consumption. In some advanced methods, routers are put in sleep mode and only wake up when they are needed to turn/inject packets. Since waking up the router takes several cycles to complete, packets will experience high latency. In this regard, application mapping significantly impacts the number of turns. This article proposes a reinforcement learning (RL) framework based on Actor-Critic architecture to optimize the application mapping problem to minimize the number of turn packets as well as communication cost. Our RL framework learns the heuristic of the mapping problem and outputs a near-optimal mapping. A 2-opt local search algorithm fine-tunes this strategy and provides an improved mapping. Our simulations show that the proposed RL framework can achieve better cost and algorithm run-time performance compared to other heuristic algorithms such as Simulated Annealing (SA) and Genetic Algorithm (GA).**

***Index Terms*—Network on Chip, Application mapping, Turn-aware, Power Gating, Reinforcement Learning**

I. Introduction

With the rise of multi/many-core processors, chips have become more complicated and consequently more power-consuming. In such complex chips, one of the components that consumes a significant portion of the chip's total power is Network-on-chip (NoC) [1]. NoC share communication infrastructure resources to provide high bandwidth to processing elements (PEs) in the architecture of chip multiprocessors (CMPs). Routers and interconnection wires are the basic components of an NoC. On-chip routers use packet switching to transfer data between PEs via interconnection wires. Among on-chip components, routers consume a considerable portion of the total power consumed by the NoC [2]. A chip's power consumption is the sum of switching, short-circuit, and static power [3].

Several approaches to reducing the power consumption of NoCs are broadly classified as dynamic and static power reduction strategies [3]. As previously stated, dynamic power is a function of the operating voltage and frequency; consequently, lowering these two factors leads to lower dynamic power consumption. This technique, known as dynamic voltage and frequency scaling (DVFS), allows the chip's dynamic power consumption to fall while incurring some performance penalty [4]. However, the impact of DVFS on total power consumption is expected to shrink as the relative contribution of dynamic power to total power consumption is decreasing as technology advances. This ratio reduction results from the fact that by scaling down the feature size, as the transistors' operating voltage reduces, the leakage current increases due to the internal structure of the MOSFET transistors [3]. Therefore, static power consumption has become the bottleneck for chip performance nowadays. As a result, static power reduction techniques are getting more attention comparing dynamic power reduction methods. Among static power reduction approaches, power-gating is a well-established one [1], [5]–[7].

Power-gating can be applied to routers in an on-chip network while they are idle for successive cycles. In power-gating, the routers can be placed in a sleep state while the power supply is cut off. In the sleep state, a transistor with a high threshold voltage disconnects the voltage rail, and the router consumes no power [8], [9]. Upon receiving a wake-up signal, the sleeping routers will be powered on.

Application mapping is a critical stage in NoC design. Every application consists of a series of tasks managed by the application's IP cores. To meet the application's requirements, the IP cores must exchange data. Proper placement of IP cores directly impacts NoC performance [10]. The application mapping optimization procedure maps the IP cores onto the NoC topology. In general, the metrics for application mapping optimization problems are communication performance (transmission delay) and bandwidth/throughput power/energy consumption, temperature and thermal management, life-time reliability and also multi-objective communication cost.

The main contribution of this paper are as follows:

- Proposing a reinforcement learning framework based on Actor-Critic architecture to optimize the application mapping in NoC to lower the number of turn packets and minimize the communication cost of IP cores.

2771-3075/22/$31.00 ©2022 Crown
DOI 10.1109/MCSoC57363.2022.00061

- Compared to the traditional heuristic algorithms such as SA and GA, the proposed RL-based method achieves lower computational cost and better algorithm run-time performance.
- Decreasing the performance penalty that arises from encountering asleep routers in the power gating technique.

II. Related Work

Placement of IP cores onto the mesh-based NoC is an NP-hard problem. Researchers have proposed several algorithms using mathematical programming and heuristics to solve the application mapping optimization problem. The authors of [11] proposed mixed-integer linear programming methods for the mapping optimization problem (MILP), which uses the clustering technique to reduce complexity. To optimize the application mapping problem, the authors in [12] have proposed a heuristic based on MILP relaxation and randomized rounding. This reduces programming complexity while also lowering solution quality. In [13], the authors formulated Integer Linear Programming (ILP) to obtain the energy-efficient IP core mapping.

Nature-inspired heuristic algorithms have also been used to exploit the entire solution space of the application mapping problem. Ant colony algorithm (ACO) [14], genetic algorithm (GA) [15], [16], and discrete particle swarm optimization (DPSO) [17] are examples of such efforts. In [18], simulated annealing (SA) algorithm is employed to minimize the application execution time and energy consumption.

From the research works reported in this paper, we can observe that these algorithms are very complex and require high computational power and time as the heuristic-based algorithm has to search the whole solution space iteratively. A human expert or a global application mapping model is required to guide and improve the performance of the heuristic to find an optimal solution. Another point is that while these methods may perform well in reducing power and latency in a general NoC architecture, they are not optimized for an NoC that leverages power-gating. In [3], the authors proposed a genetic algorithm (TAMA) specifically optimized for NoCs equipped with power-gating. However, this method also requires a substantial amount of computation and time.

In [19], an energy consumption, task distribution profile, latency, and throughput analysis of deep learning AI applications on NoC architectures is presented. Based on queueing theory, the authros in [20] have proposed an analytical model for predicting latency. In order to improve the accuracy of the analytical model's latency prediction, a deep neural network has been used, called the latency prediction neural network (LPNet). In [21], the authors proposed pointer networks and RL based framework to solve the Traveling salesman Problem (TSP). The authors in [22], proposed an RL architecture to learn heuristics for the TSP problem. This RL framework is based on the Actor-Critic method and attention mechanism. In [23], the authors have proposed the same Actor-Critic based RL algorithm of [22] to optimize the application mapping problem based on communication cost. Similar to the work of [23], we use the Actor-Critic model proposed in [22]. To improve the reliability of heterogeneous NoCs, the author in [24] proposed a reliability-aware application mapping technique. This algorithm combines the Multi-Objective Particle Swarp Optimization (MOPSO) algorithm with Reinforcement Learning (RL). Choudhary et. al [25] proposed RAMAN approach which is an RL-inspired algorithm for mapping applications onto mesh-based NoC. This method is a modified version of Q-Learning that aims to minimize the on-chip communication costs. However, our model is optimized for an NoC architecture that utilizes power-gating.

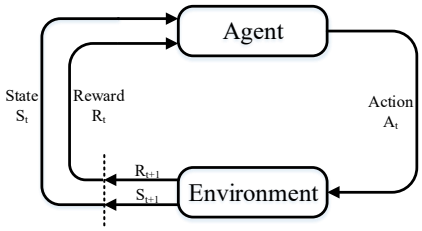


Fig. 1: Reinforcement Learning diagram

III. Background

In this section we explain reinforcement leaning along with Actor-Critic model which construct the core of of our proposed method to solve the mapping problem in NoCs with the objective of minimizing the communication cost as well as number of turn packets in the network.

A. Reinforcement learning

Reinforcement learning is a field of machine learning along with supervised and unsupervised learning. In reinforcement learning, an agent performs a specific task by interacting with an unknown environment. At each time step, the agent observes the environment state and decides the action to take in that state. According to the selected action and the dynamics of the environment model, a transition to a new state is performed. At the same time, the environment also returns a numerical reward signal, which expresses the goal of the task, reinforcing good decision making and penalizing bad decision making. A typical setting of the interaction between the agent and the environment is shown in Fig. 1. The notion of RL is focused on gradually improving the agent's behavior through trial and error by maximizing the long-term expected reward.

B. Actor-Critic model

The central intuition behind the Actor-Critic method is that the actor takes as input the state and outputs the best action (see Fig. 2). This essentially controls the agent's behavior by learning the optimal policy (policy-based). On the other hand, the critic evaluates the action by computing the value function (value-based). Those two models participate in a task where they both get better in their roles as time passes. The result is that the overall architecture will learn to do the task more efficiently than the two methods separately. The actor can be

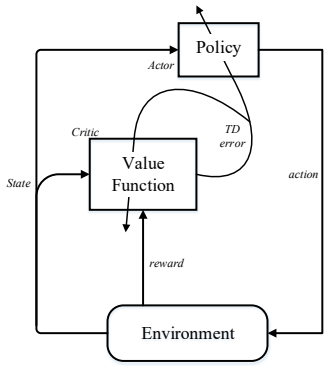


Fig. 2: Diagram of the Actor-Critic method

a function approximator like a neural network, and its task is to produce the best action for a given state. The critic is another function approximator, which receives as input the environment and the action by the actor, concatenates them, and outputs the action value for the given pair. The training of the two networks is performed separately, using gradient descent to update their weights. As time passes, the actor is learning to produce better and better actions (it is starting to learn the policy), and the critic is getting better and better at evaluating those actions.

IV. Proposed Architecture

As previously stated, in the novel NoC architectures like TooT [26], Changesub [27] and TAMA [3], equipped with power gating techniques, power gated routers need to be woken up just in case of receiving turn packets or injection phase. So turn packets play a critical role in the power gating technique's power efficiency. We cannot do anything with the number of packets that must be injected into the network because the injection rate is an inherent feature of the applications, but the adopted mapping scheme can be influenced the number of turn packets. So by adopting an appropriate turn-aware mapping scheme, we can reduce the number of turn packets and consequently avoid turning on the asleep routers. In this case, the latency incurred due to the turn packets in an NoC with a power gating technique can be dramatically dropped. For this objective, the mapping scheme must assign tasks to the proper location so that the number of turn packets reduces across the network.

Another point we should pay attention to is that if we adopt the turn-aware mapping method, we may map the tasks far away from each other to reduce the number of turns. Deploying tasks far from each other may increase the communication cost and decrease the performance of the on-chip network, which contradicts the spirit of mapping schemes. To solve this challenge, we need to calculate the cost of turn packets in the sense of network performance or latency. Supposing that we consider two cycles to pass a straight packet through a router, eight cycles for powering on an asleep router, and four-stage router architecture. In this case, the latency of

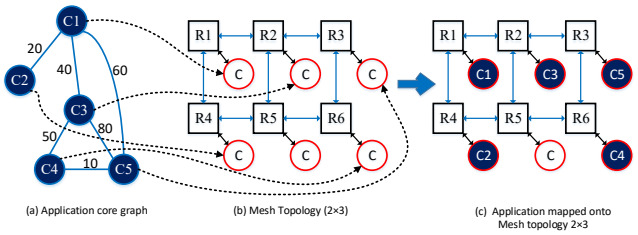


Fig. 3: IP core mapping example [23]

six straight packets is equal to the latency of a turn packet [3]. As a result, the distance between source and destination (hop count) and the number of turns must be considered as decision aspects of our mapping scheme. In short, to handle the described problem, two factors must be considered: (1) the number of turns and (2) Hop counts. Consequently, a two-objective optimization function is required.

An application core graph specifies the interconnection between the cores and the bandwidth required to handle the tasks. Our proposed mapping scheme aims to allocate each core to only one location to minimize the sum of turns and hop count (to minimize the communication cost), considering their weight (bandwidth). To this end the formulations of our application mapping problem are as follows:

- An application core graph (See Fig. 3(a)) is defined as directed graph $G(C; A)$, where each vertex $c_i \in C$ denotes an application core, and the edge $e_{ij} \in E$ denotes the link between the application cores c_i to c_j. The weight of the edge e_{ij}, represents the communication bandwidth (bw_{ij}).
- Mesh topology-based NoC of size $X \times Y$ has $X \times Y$ number of locations to place the IP cores of an application (See Fig. 3(b). Each location can accommodate one core. $r \in R$ is a set of locations in the mesh topology. $r_i(x_i, y_i)$ and $r_j(x_j, y_j)$ are the locations to which the cores of edge e_{ij} are mapped.
  Considering the XY-routing algorithm, the minimum number of hops between two locations in mesh topology is as follows:

$$Number\ of\ hops\ (d_{ij}) = |x_j - x_i| + |y_j - y_i| \quad (1)$$

  Also, under the XY-routing scheme, the minimum number of turns between two routers in mesh topology is given as:

$$Number\ of\ turns\ (t_{ij}) = \begin{cases} 0 & if\ x_j = x_i\ or\ y_j = y_i \\ 1 & otherwise \end{cases} \quad (2)$$

- The mapping sequence provides information about the connection between the locations in the NoC and the cores of the application. Based on the mapping sequence, the cores of the application graph are assigned to the

locations in the mesh topology in a zigzag order from top-left to bottom-right (See Fig. 3(c) and Table I).

- We have defined a constant λ, the ratio of the turn cost to hop-count cost. As mentioned earlier, the cost of six straight packets is approximately equal to one turn, so in our study $\lambda = 6$.
- Depending on the mapping sequence, the number of hops and turns between the cores mapped onto the mesh topology varies, which impacts the cost.

Based on the above definitions and assuming n is the total number of cores in our application core graph, we define our cost function as follows:

$$Overall\,cost = \sum_{\forall\, edges} Cost = \sum_{i=1}^{n}\sum_{j=1}^{n}(d_{ij} + \lambda t_{ij})bw_{ij} \quad (3)$$

Fig. 3 shows an application with 5 cores mapped into a mesh of size 2×3 and Table I shows the mapping sequence and core placements.

V. Reinforcement Learning Based Turn-Aware Application Mapping

The proposed reinforcement algorithm in this paper is based on actor and critic networks. To apply the actor-critic model to our mapping problem, we have used the model in [28] as a baseline for our problem. We have altered the input embedding layer and reward functions to work on our application mapping problem. The embedding layer accepts the application core graph as input and transforms it for further processing. The actor takes the embedding layer output as input and produces the mapping probabilities. The mapping sequence (M) is obtained from these probabilities, and the cost is calculated using Equation 2. The temporal difference between the cost of the mapping sequence obtained from the actor and the baseline value obtained from the critic is used to train the actor and the critic. The architecture of the proposed network is shown in Fig. 4.

Considering we have a task graph with n cores C, our application mapping problem consists in finding the minimum cost calculated by Equation 3, considering that each core is assigned to only one location. Using Neural Networks and Policy Gradient, we aim to learn the parameters θ of a stochastic policy over mapping sequence permutations $p_\theta(M|C)$. The stochastic policy uses the Adam optimizer to tune the network parameters to minimize the cost. Given an input application core graph of cores C, the key idea is to assign a higher probability to "*good*" mapping sequences, which result in lower costs, and a lower probability to "*undesirable*" mapping sequences with higher costs.

TABLE I: Core assignment in the mapping of an application graph to a mesh

NoC Location	R1	R2	R3	R4	R5	R6
Mapping sequence	1	3	5	2	-	4
Cores	C1	C3	C5	C2	-	C4

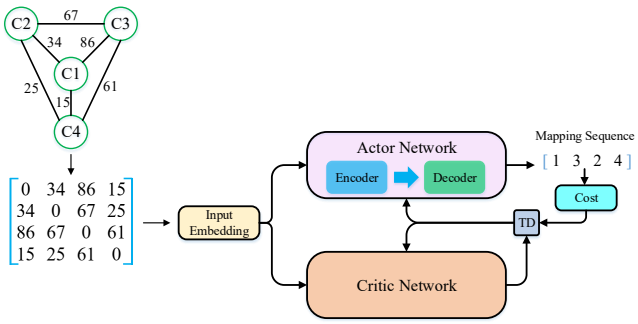


Fig. 4: Reinforcement learning framework

A. Input Embedding

We implement the embedding layer as a simple Graph Convolutional Neural Network (GCNN) based on [29]. The idea is to represent the features of each node (core) of the application graph as a vector in a d-dimensional space. The embedding layer is defined as:

$$Embed(C) = ReLU(D^{-1}\hat{A}(ReLU(D^{-1}\hat{A}W_1)W_2) \quad (4)$$

Where A is the adjacency matrix of the application graph, $\hat{A} = A + 1$ and I is the unity matrix. D is the diagonal node degree matrix of A. W_1 and W_2 are the learnable weights of the network.

B. Actor Network

The Actor network is a typical encoder-decoder. The encoder maps the input set $I = (i_1, ..., i_n)$ into a set of continuous representations $Z = (z_1, ..., z_n)$. The decoder then takes Z and generates an output sequence $O = (o_1, ..., o_n)$ of symbols one element at a time. The model is auto-regressive

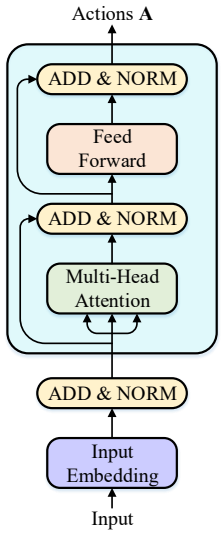


Fig. 5: Encoder Architecture

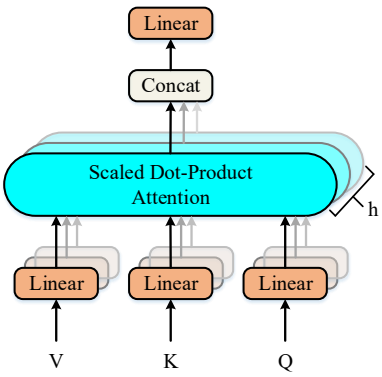


Fig. 6: Multi-Head Attention [28]

at each step, meaning it uses the previously generated symbols as an extra input when generating the next symbol [28].

(a) Encoder The purpose of the encoder is to obtain the representation of each core in the form of actions. An action represents a core. The encoder takes the core graph's embedded and batch normalized adjacency matrix and constructs a set of action vectors (mapping sequences) $A = (a_1, ..., a_n)$, each representing a core. The structure of the encoder is shown in Fig. 5. It consists of a stack of N identical layers. Each layer is divided into two sublayers: The Multi-head Attention (MHA) layer and Feed-Forward Network (FFN) layer.

- *Multi-Head Attention:* The attention mechanism of MHA provides interaction between queries and key-value pairs. Fig. 6 shows the structure of the MHA layer. For the application mapping, queries (q_i) and key-value pairs (k_i ,v_i) are obtained by linearly transforming each core (c_i) and applying ReLU nonlinearity. The output of the embedding layer is given as input to the MHA layer. Attention mechanism [33] is defined as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V \quad (5)$$

- *Feed Forward Network:* In addition to the MHA sublayer, the encoder contains a fully connected Feed-Forward network layer. It includes two position-wise linear transformations with a ReLU activation function between them as follows:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \quad (6)$$

Where x is the input, W_1 and W_2 are the network weights, and $b-1$ and b_2 are biases of the network. The input of the sublayer is added to the output of the sublayer and sent to the normalization layer.

(b) Decoder The proposed decoder network architecture uses the chain rule to factorize the probability of the mapping sequence as foloows:

$$p_\theta(M|C) = \prod_{t=1}^{n} p_\theta(m(t)|M(<t), C) \quad (7)$$

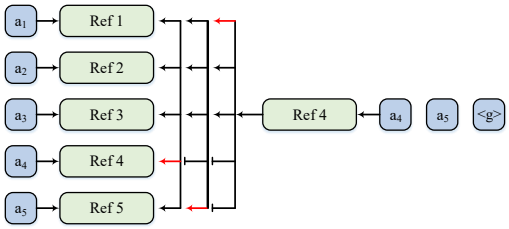


Fig. 7: Decoder architecture

Each term on the right-hand side of Equation 7 is computed sequentially with *softmax* modules. This model explicitly forgets the previous actions after $K = 3$ steps, dispensing with Long Short-Term Memory (LSTM) networks. At each output time t, the three last sampled actions are mapped to the following query vector:

$$q_t = ReLU(W_1 a_{M(t-1)} + W_2 a_{M(t-2)} + W_3 a_{M(t-3)}) \quad (8)$$

Fig. 7 shows the neural decoder structure [28]. The interaction of query vector q_t with a set of n action vectors defines the pointing distribution over the action space and maps a core to a location. Whenever a core is mapped, the selected action vector is updated in the trajectory q_{t+1}, and the process continues until the mapping is completed. The completed mapping receives the overall cost as the reward.

- *Pointing mechanism:* The pointing mechanism predicts the distribution over IP cores based on encoded actions and a state representation (query vector). The proposed pointing mechanism is parameterized by two attention matrices W_{ref}, W_q, and an attention vector v as follows:

$$\forall i \leq n, u_i^t = \begin{cases} v^T \tanh(W_{ref} a_i + W_q q_t) & ,i \notin M(0), M(1), ..., M(t-1) \\ -\infty & ,\text{otherwise} \end{cases} \quad (9)$$

$$p_\theta(M(t)|M(<t), C) = softmax(L \tanh(u^t/T)) \quad (10)$$

In Equation 10 the expression $p_\theta(M(t)|M(<\ t), C)$ predicts a distribution over the set of n action vectors, given a query q_t. A mask is used to set the logits (a.k.a. log-probabilities) of cores that already appeared in the mapping sequence to $-\infty$, as shown in Equation 9. This ensures that the model gives a valid permutation of the input. Then, the logits are clipped in $[-L, +L]$ to control the entropy. T is a temperature hyper-parameter used to control the certainty of the sampling. $T = 1$ during training and $T > 1$ during inference.

Algorithm 1: 2-opt local search algorithm

Input: Core graph (C), Mapping sequence from RL (M_{RL})
Output: Best mapping sequence (M)

```
Initialize iter i;
M ← (M_RL)
Cost_best ← Calculate Cost(M_RL, C)
for iter ← 0 to i do
    M_new ← 2 − opt(M_RL)
    Cost_new ← Calculate Cost(M_new, C)
    if Cost_new < Cost_best then
        M ← (M_new
        Cost_best ← Cost_new
```

C. Critic Network

The proposed critic network uses the same encoder as the actor network. The critic network uses pointing distribution over the cores $p_\phi(C)$. It specifies a glimpse vector gl_c, which is calculated as a weighted sum of the action vectors $A = (a_1, ..., a_n)$ as follows:

$$gl_c = \sum_{i=1}^{n} p_\phi(C)_i a_i \tag{11}$$

The glimpse vector gl_c is fed to two fully connected layers with ReLU activation. The output of the critic network is a prediction of the cost. The critic is trained by minimizing the Mean Square Error between its predictions and the actor rewards.

D. 2-Opt Local Search

2-opt local search [30] is employed to refine the solution obtained by the neural network. Algorithm 1 shows the pseudo-code for the 2-opt local search. The algorithm swaps the positions of the cores in the mapping sequence obtained by the neural network and generates a new solution. If the overall cost of the new mapping is less than the previous solution, the best solution is replaced with the new solution. This process is repeated until no further improvement is observed in the cost value or the maximum number of iterations is achieved.

VI. Results and Discussion

A. Training the Model

Supervised learning for NP-hard problems like application task mapping is infeasible as the search space is huge and the amount of required labeled data is enormous. In contrast, reinforcement learning provides an appropriate and straightforward paradigm for training a Neural Network. A reinforcement learning agent explores different mappings and observes their corresponding learning rewards. The Neural Network is trained by Policy Gradient using the reinforcement learning rule. The cost function defined earlier is used as reward $r(M|C)$.

We have used a free and open-source tool called Task Graphs for Free (TGFF) to generate our random application core graphs. Different graphs with a different number of cores ranging from 6 to 64 were generated. For this work, a mesh size of 8x8 was considered. A batch size of 256 was selected, and the network was trained with 128,000,000 task graphs of different sizes. Also, Stochastic Gradient Descent (SGD) with Adam optimizer ($\beta_1 = 0.9, \beta_2 = 0.99, and\, \epsilon = 10^{-9}$) is used. The model was implemented in Python language using the TensorFlow library.

The training and testing of the algorithms were performed on a high-performance machine with an Intel(R) Xeon(R) Silver 4114 @ 2.20GHz CPU and 32 GB of RAM. The system also had 2 GPUs installed (GeForce RTX 2080 and Quadro K620). It took approximately 120 hours for the training process to complete.

B. Analyzing the Simulation Results

In this work, we test the RL-based application mapping framework against multiple benchmarks. As mentioned earlier, the mapping sequence obtained by the network is further improved by employing a 2-opt local search. Eight task graph of real world applications (i.e., DVOPD, VOPD, MPEG4, PIP, MWD, 263 ENC MP3 DEC, MP3 ENC MP3 DEC, 263 DEC MP3 DEC) have been applied as the benchmarks [31].

Table II demonstrates the value of the cost function for the RL algorithm and the cost obtained after applying the 2-opt local search. The run-time of the algorithms is also presented in that table. We have run the algorithm ten times, and the results presented in this table are the average of ten runs.

As we can see, the mapping obtained from the RL algorithm is not optimal. However, using the 2-opt local search algorithm, we can see a significant improvement in terms of cost. We can also notice that the run-time of the RL algorithm is the same for a different number of cores as the network architecture does not depend on this parameter. On the other hand, the algorithm's run-time leveraging the 2-opt search has increased substantially as the number of cores has increased. The reason is that as the complexity of the problem increases with a higher number of cores, the 2-opt algorithm needs more iterations to find the optimum mapping.

The result of our algorithm is also compared to using the 2-opt algorithm initialized randomly and two other generic heuristic optimization algorithms; Simulated Annealing (SA) and Genetic algorithm (GA). The values of cost and the run-time of the algorithms are presented in Table III.

Fig. 8a illustrates the cost of different algorithms normalized to the cost obtained from our model. Fig. 8b also shows the run-times normalized to the run-time of our proposed method.

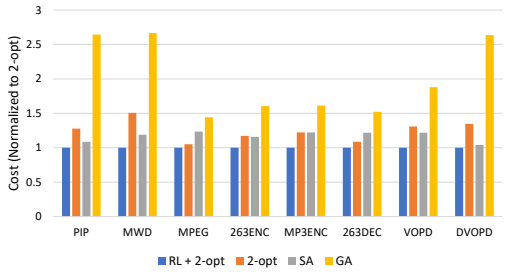
Fig. 8a shows that our RL network has efficiently learned the heuristics for the application mapping problem during the training. As a result, the cost value obtained from utilizing the RL network before the 2-opt search is substantially lower than using 2-opt alone with a random initial mapping. It is also clear that our method outperforms the other heuristic methods in terms of cost such that the 2-opt, SA, and GA methods result in 25%, 17%, and 100% higher costs. According to Fig. 8b, we can also see that the run-time of our algorithm is also

TABLE II: Comparison of our models cost before and after applying 2-opt local search

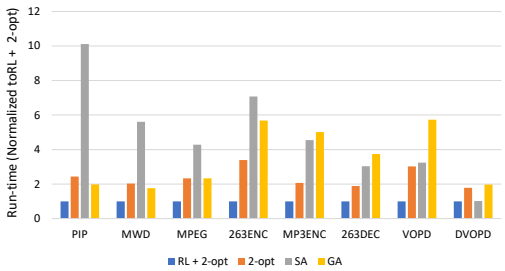
Application	Number of Cores	RL Algorithm		RL + 2-opt search	
		Cost	Run-time (s)	Cost	Run-time (s)
PIP	8	896	0.38	806	4.14
MWD	12	6694	0.39	3084	7.55
MPEG	12	11007	0.4	8462	9.87
263ENC	12	444	0.39	283	5.92
MP3ENC	13	25	0.4	18	9.28
263DEC	14	34	0.4	23	13.83
VODP	16	8106	0.4	4730	12.93
VODPD	32	20194	0.4	14046	42.03

TABLE III: Comparison of our model with other generic heuristic methods

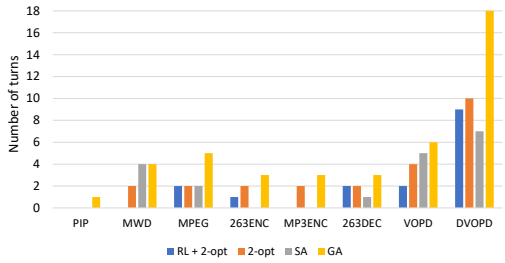
Application	Number of Cores	RL + 2-opt search		2-opt		SA		GA	
		cost	Run-time (s)	Cost	Run-time (s)	Cost	Run-time (s)	Cost	Run-time (s)
PIP	8	806	4.14	1030	10.1	876	41.86	2131	8.2
MWD	12	3084	7.55	4646	15.31	3667	42.3	8224	13.32
MPEG	12	8462	9.87	8895	22.96	10448	42.3	12215	23.03
263ENC	12	283	5.92	332	20.1	328	41.89	454	33.63
MP3ENC	13	18	9.28	22	19.13	22	42.23	29	46.52
263DEC	14	23	13.83	25	26.11	28	41.93	35	51.73
VOPD	16	4730	12.93	6186	39.12	5757	41.86	8892	74.08
DVOPD	32	14046	42.03	18932	75.04	14627	45.52	37038	82.98



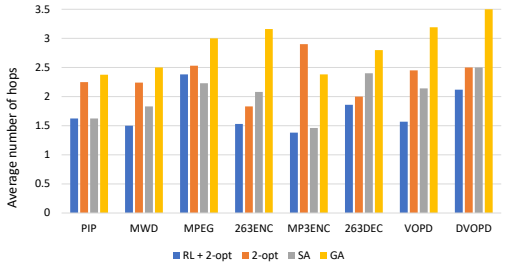
(a) Cost of different algorithms normalized to RL+2-opt



(b) Algorithm's run-time normalized to RL+2-opt



(c) Number of turns for different algorithms



(d) Average number of hops for different algorithms

Fig. 8: Performnce evaluation of the proposed method against 2-opt search, SA, and GA algorithms

lower than the 2-opt algorithm alone. The reason is that our RL network was able to provide a reasonable estimate of the optimum mapping to the 2-opt algorithm. As a result, the 2-opt algorithm finds the optimum mapping in fewer iterations. On average, the run-time of the 2-opt, SA, and GA methods are $2.37\times$, $4.86\times$, $3.52\times$ times of our proposed method.

Fig. 8c shows the number of turns per each benchmark application for different algorithms. For most of the benchmarks, our method results in fewer turns. Among the three methods compared with our proposed method, the SA method shows the best performance regarding the number of turns than the 2-opt local search and GA. On average, in the SA, 2-opt local search, and GA, the number of turns is 18.75%, 50%, and 168.75% more than our proposed method.

As mentioned earlier, our goal was to optimize the mapping algorithm for fewer turns and reduce the distance between mapped cores. As shown in fig. 8d, our proposed method outperforms the other methods regarding the average number of hop-count. On average, the SA method obtained the best result compared to the 2-opt search and GA methods such

that in the SA method, the hop count is 17.6% higher than our proposed method, while in the 2-opt local search and GA methods, this rise is 38.17% and 67.1% respectively.

VII. Conclusion

Although power-gating is a promising technique to reduce the static power of routers in NoCs, it can cause substantial network performance degradation. One of the sources of performance loss in power gating techniques is wake-up latency incurred while turning the asleep components. In some advanced power gating methods in NoC architecture with a minor architectural modification, there is no need to turn on the power gated routes while packets must go straight through them. In this paper, we proposed a reinforcement learning-based turn-aware application mapping to minimize the number of turns between application cores while maintaining their distance as low as possible. Compared to some state-of-the-art approaches, our proposed method combined with the 2-opt local search shows the best overall performance. The reason is that our method can predict the heuristics of the problem and provide a reasonable estimate of the optimum mapping to the 2-opt algorithm. Our method's overall cost of mapping sequences was reduced by 24%, 17%, and 100% compared to generic 2-opt, SA, and GA algorithms.

References

[1] E. Ofori-Attah and M. O. Agyeman, "A survey of power-aware network-on-chip design techniques," in *30th International Multi-Conference on Computing in Global Information Technology (IARIA)*, 2018.

[2] M. Baharloo and A. Khonsari, "A low-power wireless-assisted multiple network-on-chip," *Microprocessors and Microsystems*, vol. 63, pp. 104–115, 2018.

[3] R. Aligholipour, M. Baharloo, B. Farzaneh, M. Abdollahi, and A. Khonsari, "Tama: turn-aware mapping and architecture–a power-efficient network-on-chip approach," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5, pp. 1–24, 2021.

[4] C. G. d. A. Gewehr, "Design space exploration of hybrid topologies and dvfs in on-chip communication networks," 2021.

[5] L. Chen, D. Zhu, M. Pedram, and T. M. Pinkston, "Power punch: Towards non-blocking power-gating of noc routers," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 378–389.

[6] R. Das, S. Narayanasamy, S. K. Satpathy, and R. G. Dreslinski, "Catnap: Energy proportional multiple network-on-chip," in *Proceedings of the 40th annual international symposium on Computer architecture*, 2013, pp. 320–331.

[7] A. Namazi and M. Abdollahi, "Pcg: Partially clock-gating approach to reduce the power consumption of fault-tolerant register files," in *2017 Euromicro Conference on Digital System Design (DSD)*. IEEE, 2017, pp. 323–328.

[8] D. DiTomaso, A. Sikder, A. Kodi, and A. Louri, "Machine learning enabled power-aware network-on-chip design," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1354–1359.

[9] D. Zoni and W. Fornaciari, "Modeling dvfs and power-gating actuators for cycle-accurate noc-based simulators," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 3, pp. 1–24, 2015.

[10] U. Y. Ogras, J. Hu, and R. Marculescu, "Key research problems in noc design: a holistic perspective," in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2005, pp. 69–74.

[11] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, pp. 407–420, 2006.

[12] P. Ghosh, A. Sen, and A. Hall, "Energy efficient application mapping to noc processing elements operating at multiple voltage levels," in *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*. IEEE, 2009, pp. 80–85.

[13] S. Tosun, O. Ozturk, and M. Ozen, "An ilp formulation for application mapping onto network-on-chips," in *2009 International Conference on Application of Information and Communication Technologies*. IEEE, 2009, pp. 1–5.

[14] M. Farias, E. Barros, A. Araújo, A. Silva, J. Melo *et al.*, "An ant colony metaheuristic for energy aware application mapping on nocs," in *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, 2013, pp. 365–368.

[15] C. Xu, Y. Liu, P. Li, and Y. Yang, "Unified multi-objective mapping for network-on-chip using genetic-based hyper-heuristic algorithms," *IET Computers & Digital Techniques*, vol. 12, no. 4, pp. 158–166, 2018.

[16] L. Guo, Y. Ge, W. Hou, P. Guo, Q. Cai, and J. Wu, "A novel ip-core mapping algorithm in reliable 3d optical network-on-chips," *Optical Switching and Networking*, vol. 27, pp. 50–57, 2018.

[17] P. K. Sahu, T. Shah, K. Manna, and S. Chattopadhyay, "Application mapping onto mesh-based network-on-chip using discrete particle swarm optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 2, pp. 300–312, 2013.

[18] C. Marcon, A. Borin, A. Susin, L. Carro, and F. Wagner, "Time and energy efficient mapping of embedded applications onto nocs," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, 2005, pp. 33–38.

[19] Z. A. Khan, U. Abbasi, and S. W. Kim, "An efficient algorithm for mapping deep learning applications on the noc architecture," *Applied Sciences*, vol. 12, no. 6, p. 3163, 2022.

[20] R. Sambangi, H. Manghnani, and S. Chattopadhyay, "Lpnet: A dnn based latency prediction technique for application mapping in network-on-chip design," *Microprocessors and Microsystems*, vol. 87, p. 104370, 2021.

[21] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[22] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the tsp by policy gradient," in *International conference on the integration of constraint programming, artificial intelligence, and operations research*. Springer, 2018, pp. 170–181.

[23] S. Jagadheesh and P. V. Bhanu, "Noc application mapping optimization using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2022.

[24] N. Kadri, A. Chenine, Z. Laib, and M. Koudil, "Reliability-aware intelligent mapping based on reinforcement learning for networks-on-chips," *The Journal of Supercomputing*, pp. 1–36, 2022.

[25] J. Choudhary, J. Soumya, and L. R. Cenkeramaddi, "Raman: Reinforcement learning inspired algorithm for mapping applications onto mesh network-on-chip," in *2021 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*. IEEE, 2021, pp. 52–58.

[26] H. Farrokhbakht, M. Taram, B. Khaleghi, and S. Hessabi, "Toot: an efficient and scalable power-gating method for noc routers," in *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2016, pp. 1–8.

[27] M. Baharloo, R. Aligholipour, M. Abdollahi, and A. Khonsari, "Changesub: A power efficient multiple network-on-chip architecture," *Computers & Electrical Engineering*, vol. 83, p. 106578, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790619317288

[28] W.-J. Van Hoeve, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings*. Springer, 2018, vol. 10848.

[29] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[30] M. Englert, H. Röglin, and B. Vöcking, "Worst case and probabilistic analysis of the 2-opt algorithm for the tsp," *Algorithmica*, vol. 68, no. 1, pp. 190–264, 2014.

[31] P. K. Sahu, K. Manna, N. Shah, and S. Chattopadhyay, "Extending kernighan–lin partitioning heuristic for application mapping onto network-on-chip," *Journal of Systems Architecture*, vol. 60, no. 7, pp. 562–578, 2014.