# Network-on-Chip Turn-aware application mapping optimization using Reinforcement Learning

By:

**Mohammadmehdi Shammasi**

A Report Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Engineering

In The Department of Electrical and Computer Engineering

University of Victoria

## Supervisory Committee

Dr. Amirali Baniasadi, **Supervisor**

(Department of Electrical and Computer Engineering)

Dr. Kin Fun Li, **Committee Member**

(Department of Electrical and Computer Engineering)

# Table of Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| CMOS: | Complementary Metal-Oxide Semiconductor |
| DVFS: | Dynamic Voltage and Frequency Scaling |
| FFN: | Feed Forward Network |
| GA: | Genetic Algorithm |
| GCNN: | Graph Convolutional Neural Network |
| ILP: | Integer Linear Programming |
| IP: | Intellectual Property |
| MHA: | Multi-Head Attention |
| ML: | Machine Learning |
| MOSFET: | Metal Oxide Semiconductor Field Effect Transistor |
| MPSoC: | Multiprocessor System-on-Chips |
| NoC: | Network on Chip |
| RL: | Reinforcement Learning |
| SA: | Simulated Annealing |
| SoC: | System on Chip |

# Acknowledgments

I would like to thank my supervisor, Dr. Amirali Baniasadi, for all his support, guidance and mentorship throughout my research and my program.

# Dedication

I dedicate this work to my parents who have always supported me with their love and guidance.

# Abstract

In todays advanced SoCs (System-on-Chip), power efficiency is a crucial concern. As chips get denser and more complicated, power consumption is becoming the bottleneck in further enhancing system's performance. Serving as the backbone for many-core chips, NoCs (Network-on-chip) consume a significant share of total chips power. As a result, decreasing the power consumption in these components can reduce the total chip's power significantly. Power-gating is a promising technique which can be used to reduce the static power consumption in NoC's routers. In this method routers are put in a sleep mode and only wake up when a turning packet needs to pass. Since the process of waking up the router takes several cycles to complete, turning packets will experience a high amount of latency. In this regard, application mapping has a significant impact on number of turns and latency between cores of an application. In this article we propose a Reinforcement Learning framework based on Actor-Critic architecture to optimize the application mapping problem for fewer number of turns as well as keeping the distance of the cores minimum. Our RL framework learns the heuristic of the mapping problem and outputs a sub-optimal mapping. A 2-opt local search algorithm fine-tunes this mapping and provides an improved mapping. The results of our simulations show that this RL framework could achieve better performance in terms of cost and algorithm run-time comparing to other heuristic algorithms such as Simulated Annealing (SA) and Genetic Algorithm (GA).

# 1 Introduction

The advancement of technology has brought newer and state-of-the-art chips and processors every year. As newer chips become smaller and smaller, transistor density has increased drastically. The progress of technology in reducing the transistor dimensions has made it possible to accommodate more and more transistors into one chip, leading to the chips' increased throughput and performance. However, 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 NoC [1].

To address the on-chip communication challenges in Multiprocessor System-on-Chips (MPSoCs), network-on-chip (NoC) architectures have been proposed [2] which share communication infrastructure resources to provide high bandwidths to processing elements (PEs). Routers, interconnection wires, and Intellectual Property (IP) cores or PEs are the basic components of a NoC. NoC routers use packet switching to transfer data between IP cores via interconnection wires.

Routers consume a considerable portion of the total power consumed by the NoC [3]. A chip's power consumption is the sum of switching, short-circuit, and static power, as shown in Equation 1: [4]

$$P = P_{switching} + P_{short-circuit} + P_{static}$$
$$= \alpha C_L f V^2 + I_{sc} V_{dd} + I_{leakage} V_{dd} \quad (1)$$

The dynamic power in this equation is made up of switching and short-circuit power. The switching power is influenced by the operating voltage ($V_{dd}$), clock frequency ($f$), load capacitance ($C_L$), and transition activity factor ($\alpha$). The main cause of short-circuit power in CMOS circuits is short-circuit current ($I_{sc}$). The leakage current ($I_{leakage}$) causes static power consumption (leakage power).

The static to dynamic power consumption ratio rises as chip sizes and complexity grow [5]. As a result, static power consumption becomes a limiting factor in further increasing processing power [6]. Consequently, power consumption inhibits the chip from performing at its peak performance [7]. The same applies to the power consumption of the on-chip network components. Previous research indicates that the portion of on-chip network power is considerable [8, 9, 10].

There are several approaches to reducing the power consumption of NoCs, which are broadly classified as dynamic and static power reduction strategies [4]. As previously stated, dynamic power is a function of transistor voltage and frequency; consequently, lowering these two factors leads in lower dynamic power consumption. This technique is known as dynamic voltage and frequency scaling (DVFS), and it allows a chip's dynamic power consumption to be drastically lowered while incurring some performance penalty [11, 12]. However, the impact of DVFS on total power consumption is minor because the contribution of dynamic power to total power consumption is decreasing as technology advances. This reduction results from the fact that by reducing the size, as the transistors' operating voltage reduces, the leakage current increases due to the internal structure of the MOSFET transistors [4]. As technology advances, static power consumption is becoming the bottleneck in chip performance. As a result, static power reduction techniques are getting more attention than dynamic power reduction methods. Among the static power reduction approaches, power-gating is a well-established one [13, 14, 15].

Power-gating can be used in power-consuming routers in an on-chip network while they are idle for successive cycles. In power-gating, the routers will be placed in a state where the power supply is cut off. In this state, a transistor with a high threshold voltage disconnects the voltage rail, and the router consumes no power [15, 16]. If a packet follows the straight path, it will pass through a bypass route without waking up the powered-off router. Upon receiving a wake-up signal, the sleeping routers will be powered on. When the wake-up signal arrives, the pull-up transistor begins to turn on, connecting the supply voltage to the router (Figure 1). The time it takes to fully switch on the transistor is referred to as "wake-up latency" which is the source of performance degradation in power-gating strategies. Because of this latency, packets that reach sleeping routers must wait a number of cycles before the router is woken up. As a result, on-chip network latency increases dramatically, potentially resulting in a loss of overall chip performance [17].

Application mapping is a critical stage in NoC design. Every application consists of a series of tasks, which are managed by the application's IP cores. To meet the application's requirements, the IP cores must exchange data among themselves. Proper placement of IP cores has a direct impact on NoC performance [18]. The application mapping optimization procedure maps the IP cores onto the NoC topology. In general, the metrics for application mapping optimization problems are communication energy, delay, bandwidth, and communication cost. [19].

In this article, a Reinforcement Learning based neural framework is proposed for application mapping in NoCs with power-gated routers to minimize the number of turns on the routes of packets. Consequently, the delay that arises from encountering turn routers (i.e., routers that packets must turn through them) can decrease.

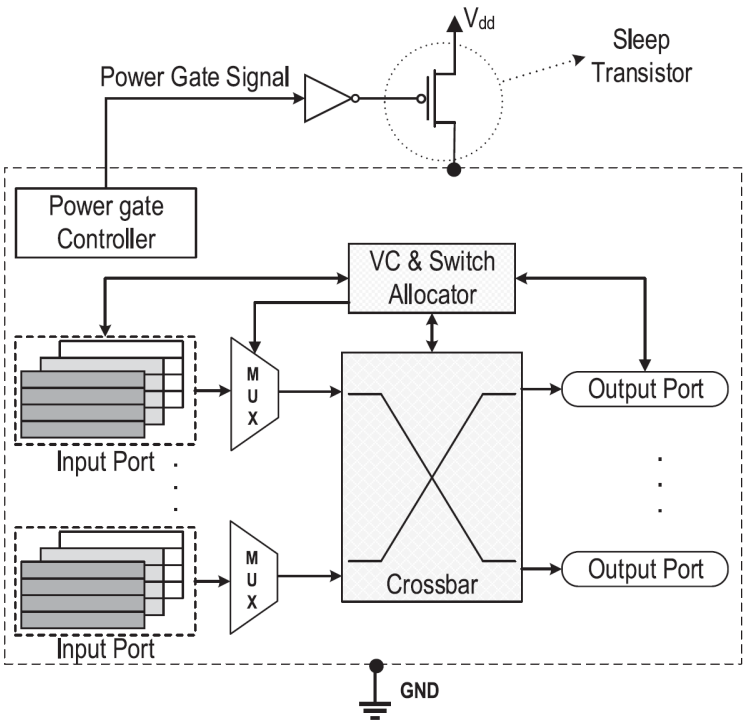



Figure 1. NoC router power-gating scheme [4]

## 2 Related work

Placement of IP cores onto the mesh-based NoC is a NP-hard problem. Researchers have proposed several algorithms using mathematical programming and heuristics to solve the application mapping optimization problem. The authors of [20] 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, [21] proposes a heuristic based on MILP relaxation and randomized rounding. It reduces programming complexity while also lowering solution quality. In [22], 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) [23], genetic algorithm (GA) [24], [25], and discrete particle swarm optimization (DPSO) [26] are examples of such efforts. In [27], simulated annealing (SA) algorithm was used to minimize the application execution time and energy consumption. From the research works reported in this literature, we can observe that these algorithms are very complex and require high computational power and time as the heuristic-based algorithm has to iteratively search the whole solution space. A human expert or a global application mapping model is required to guide and improve the performance of the heuristic to get an optimal solution. Another point is that while these methods may have a good performance in reducing power and latency in a general NoC architecture, they are not optimized for a NoC which leverages power-gating. In [4], the authors proposed a genetic algorithm (TAMA) specifically optimized for NoCs with power-gating. However, this method also requires a substantial amount of computation and time. With recent achievements in machine learning (ML), neural networks are being used to automatically extract features of the problem from data. So, ML methods have the potential to discover the heuristics of the application mapping problem.

Recently, several researchers have used neural networks to solve combinatorial optimization problems. In [28], the authors proposed pointer networks and RL based framework to solve the Traveling salesman Problem (TSP). The authors in [29], proposed a RL architecture to learn heuristics for TSP problem. This RL framework is based on Actor-Critic method and attention mechanism. In [30] authors have proposed the same Actor-Critic based RL algorithm of [29] to optimize the application mapping problem based on communication cost. Like the work of [30], we have also used the Actor-Critic model proposed in [29]. However, our model is optimized for a NoC architecture that utilizes power-gating.

# 3 Background

## 3.1 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 about 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, a numerical reward signal is also returned by the environment, 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 Figure 2. The notion of RL is focused on gradually improving the agent's behavior through trial and error, by maximizing the long-term expected reward.

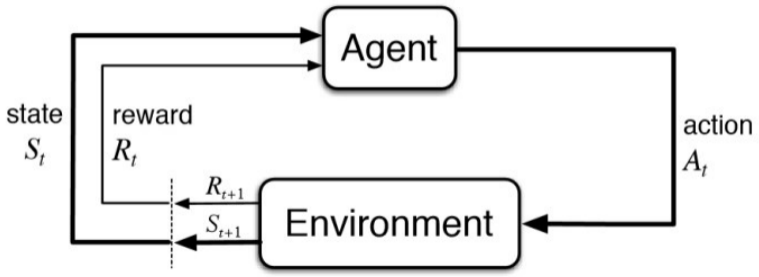



Figure 2. Reinforcement Learning diagram

## 3.2 Actor-Critic model

The main intuition behind the Actor-Critic method is that the actor takes as input the state and outputs the best action. It essentially controls how the agent behaves by learning the optimal policy (policy-based). The critic, on the other hand, evaluates the action by computing the value function (value-based). Those two models participate in a task where they both get better in their own 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 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 and it uses gradient descent to update both 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.

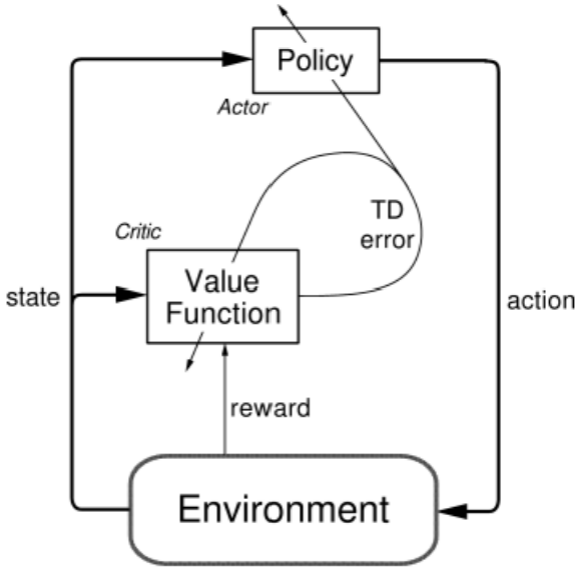



Figure 3. Diagram of the Actor-Critic method

# 4 Proposed architecture

As previously stated, the number of turns in the on-chip network can be reduced by using an appropriate mapping method. The mapping scheme should assign jobs to the appropriate location with the fewest number of turns. The method in [31] is an interesting mapping that greatly reduces the number of turns. However, fewer turns are achieved at the cost of dramatically increasing the distance of the tasks. As a result, the latency of several straight packets can be more than one turn packet. Supposing that we consider two cycles to pass a straight packet through a router, eight cycles for powering on the turn router, and four-stage router architecture. In this case, the latency of six straight packets is equal to the latency of one turn packet [4]. As a result, the distance between source and destination (hop count) must be considered. 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. The purpose of the mapping is to allocate each core to only one location to minimize the sum of turnings and distances, considering their weight (bandwidth).

The formulations for the application mapping problem are as follows:

- An application core graph 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 $X*Y$ number of locations to place the IP cores of application. 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 XY-routing algorithm minimum number of hops between two locations in mesh topology is given as:

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

  - Considering XY-routing algorithm minimum number of turns between two routers in mesh topology is given as:

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

  - 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.
  - Constant $\lambda$ is the ratio of the turn packet latency to the latency of straight packets. As mentioned earlier, the latency of 6 straight packets approximately equals the latency of one turn, so in this study $\lambda = 6$. It should be noted that this value is approximate, and the actual value depends on the physical characteristics of the fabricated chip.
  - Depending on the mapping sequence, the number of hops and turns between the cores mapped onto the mesh topology is varied which impacts the cost.

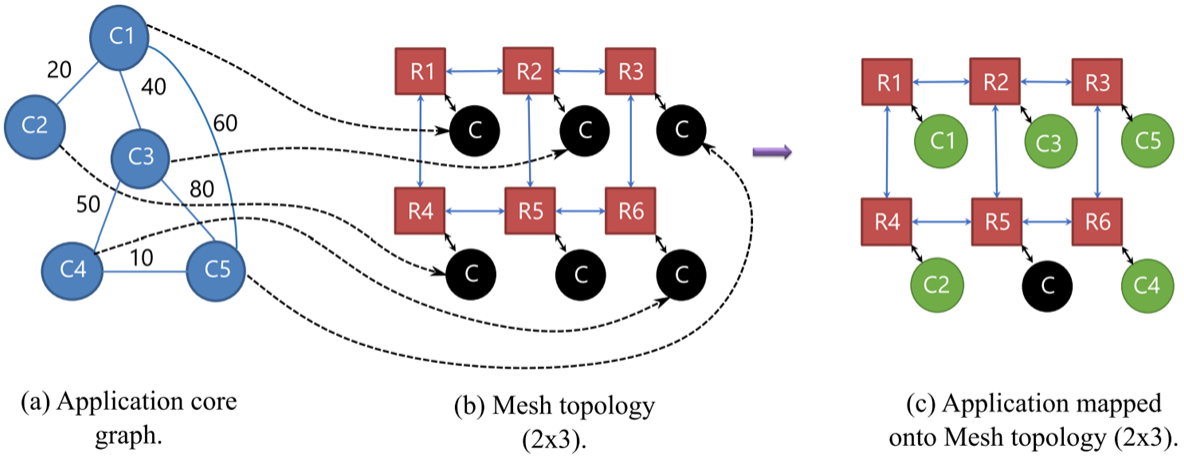
Based on the above definitions we define our cost function as follows:

$$Overall\ cost = \sum_{\forall\ Edges} Cost\ of\ edge = \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} + \lambda t_{ij}) bw_{ij} \quad (2)$$

Where $n$ is the total number of cores.

This cost function represents the sum of latencies between every two cores considering the weight of communication between them ($bw_{ij}$).

Figure 4 shows an application with 5 cores mapped into a mesh of size 2×3. Table 1 shows the mapping sequence and core placements.




(a) Application core graph.

(b) Mesh topology (2x3).

(c) Application mapped onto Mesh topology (2x3).

Figure 4. IP core mapping example. [30]

Table 1. Core assignment in 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 |

## 4.1 Reinforcement learning based Turn aware application mapping

The proposed reinforcement algorithm is based on actor and critic networks. To apply the actor-critic model to our mapping problem, we have used the model in [29] as a baseline for our problem. We have altered the input embedding layer and reward functions to work for 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. Architecture of the proposed network is shown in Figure 5.

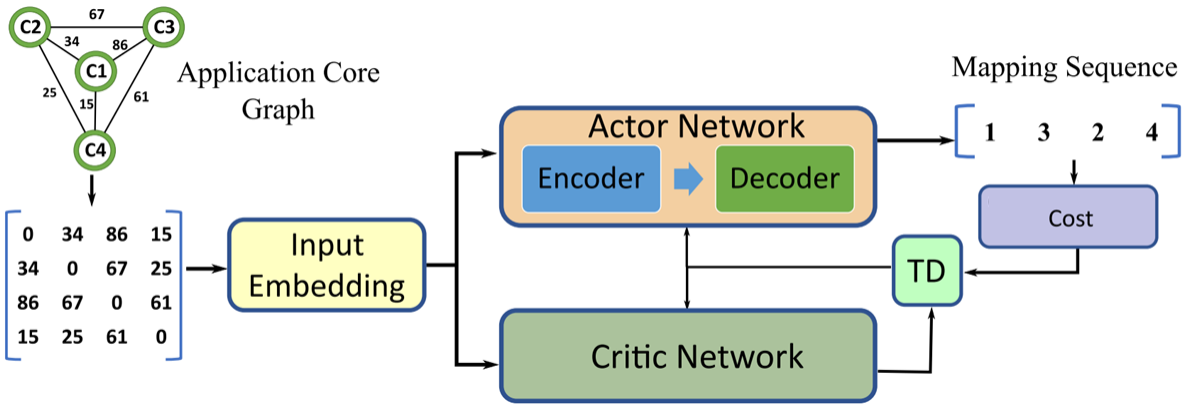



Figure 5. Reinforcement Learning framework

Considering we have a task graph with $n$ cores $C$, our application mapping problem consists in finding the minimum cost calculated by Equation 2 considering that each core is assigned to only one location. Our goal is to learn the parameters $\theta$ of a stochastic policy over mapping sequence permutations $p_\theta(M|C)$, using Neural Networks and Policy Gradient. The stochastic policy uses 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.

### 4.1.1 Input Embedding

The embedding layer was implemented as a simple Graph Convolutional Neural Network (GCNN) based on [32]. 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:

$$\text{Embed}(C) = ReLU(D^{-1}\hat{A}(ReLU\left(D^{-1}\hat{A}W_1\right)W_2) \quad (3)$$

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

### 4.1.2 Actor Network

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 autoregressive at each step meaning that It uses the previously generated symbols as an extra input when generating the next symbol.[29]

#### 4.1.2.1 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 embedded and batch normalized adjacency matrix of the core graph 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 Figure 6 [29]. It consists of a stack of N identical layers. Each layer is divided into two sublayers, namely Multi-head Attention (MHA) layer and Feed-Forward Network (FFN) layer.

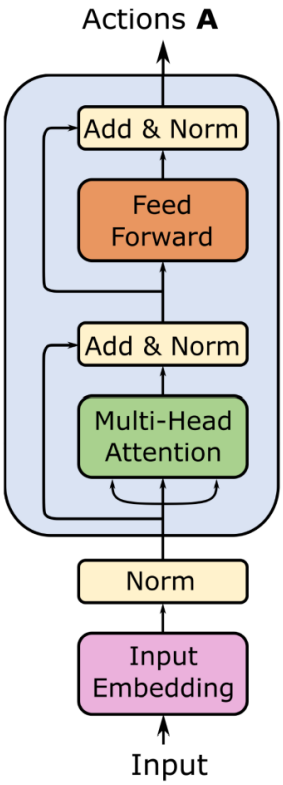



Figure 6. Encoder Architecture

**Multi-Head Attention**: The attention mechanism of MHA provides interaction between queries and key-value pairs. Figure 7 shows the structure of 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:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \qquad (4)$$

Where $Q = [q_1, \ldots, q_n]$, $K = [k_1, \ldots, k_n]$, $V = [v_1, \ldots, v_n]$ and $h$ is the number of independent subspaces. MHA sublayer outputs a new representation for each core based on the weighted sum of core values. The affinity function defines the weights based on the relation between cores, queries, and keys. The queries and key-value pairs are linearly mapped to $h$ independently trained subspaces. The attention mechanism is applied to every new set of representations to get each core output value. These values are fed to the FFN Layer.

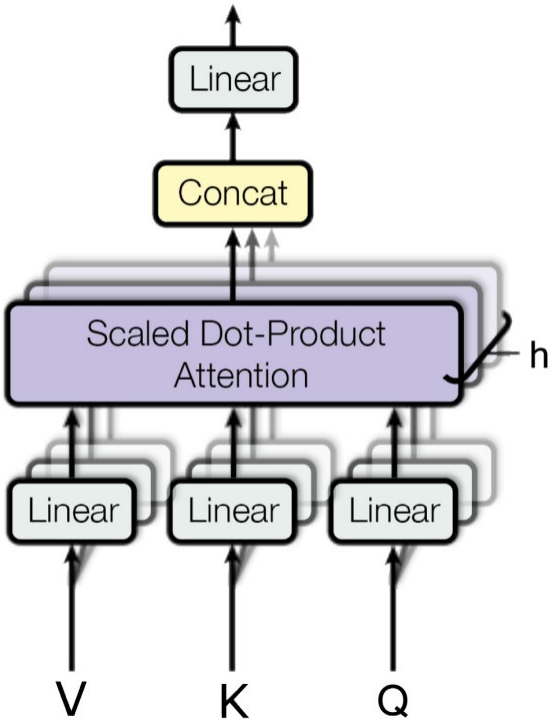



Figure 7. Multi-Head attention [30]

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

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)\,W_2 + b_2 \tag{5}$$

Where $x$ is the input, $W_1$ and $W_2$ are weights of the network, and $b_1$ and $b_2$ are biases of network. The input of the sublayer is added to the output of the sublayer and sent to the normalization layer.

#### 4.1.2.2 Decoder

The proposed decoder network architecture uses the chain rule to factorize the probability of the mapping sequence as

$$p_\theta(M|C) = \prod_{t=1}^{n} p_\theta(M(t)|M(<t), C) \tag{6}$$

Each term on the right-hand side of Equation 6 is computed sequentially with softmax modules. This model explicitly forgets the previous actions after $K = 3$ steps, dispensing with LSTM networks. At each output time $t$, the three last sampled actions are mapped to the following query vector:

$$q_t = \mathrm{ReLU}(W_1 a_{M(t-1)} + W_2 a_{M(t-2)} + W_3 a_{M(t-3)}) \tag{7}$$

Figure 8 shows the neural decoder structure [29]. 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.

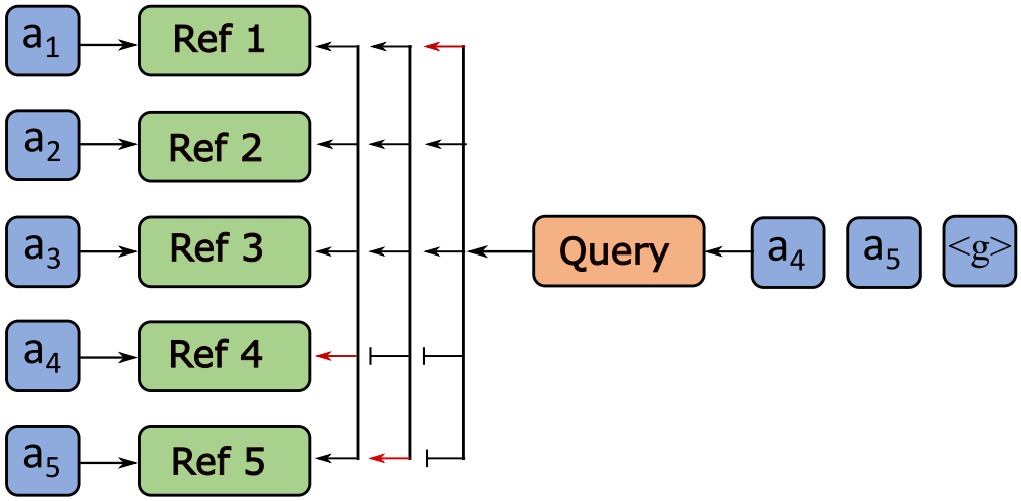



Figure 8. Decoder architecture

**Pointing Mechanism**: Pointing mechanism is used to predict 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:

$$\forall i \leq n, u_i^t = \begin{cases} v^T \tanh(w_{ref} a_i + W_q q_t) & if \ \ i \notin M(0), M(1), \dots, M(t-1) \\ -\infty & otherwise \end{cases} \quad (8)$$

$$p_\theta(M(t)|M(<t), C) = \text{softmax}(L \tanh(u^t/T)) \quad (9)$$

$p_\theta(M(t)|M(<t), C)$ predicts a distribution over the set of n action vectors, given a query $q_t$ . A mask is used to set the logits (aka log-probabilities) of cores that already appeared in the mapping sequence to $-\infty$, as shown in Equation 8. 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.

### 4.1.3 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 $A = (a_1, \ldots, a_n)$.

$$gl_c = \sum_{i=1}^{n} p_\phi(C)_i a_i \qquad (10)$$

The glimpse vector $gl_c$ is fed to a 2 fully connected layers with ReLU activations. 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.

### 4.1.4 2-opt local search

2-opt local search [34] is used to refine the solution obtained by the neural network. Figure 9 shows the pseudocode 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 the no further improvement is observed in the cost value or maximum number of iterations is achieved.

**ALGORITHM :** 2- *opt* local search algorithm

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

Initialize *iter i*;
$M \leftarrow (M_{RL})$
$Cost_{best} \leftarrow$ Calculate cost $(M_{RL}, C)$
**for** $iter \leftarrow 0$ *to* $i$ **do**
    $M_{new} \leftarrow 2 - opt(M_{RL})$
    $Cost_{new} \leftarrow$ Calculate cost $(M_{new}, C)$
    **if** $Cost_{new} < Cost_{best}$ **then**
        $M \leftarrow (M_{new})$
        $Cost_{best} \leftarrow Cost_{new}$
    **end**
**end**

Figure 9. 2-opt algorithm pseudocode

### 4.1.5 Training the Model

Supervised learning for NP-hard problems like application task mapping is infeasible as the search space is incredibly large and the amount of required labeled data is huge. By contrast, reinforcement learning provides an appropriate and simple paradigm for training a Neural Network. An RL agent explores different mappings and observes their corresponding 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 open-source tool called Task Graphs for Free (TGFF) to generate our random application core graphs. Different graphs with number of cores ranging from 6 to 64 was generated. For the purpose of this work, a mesh size of 8x8 was considered. A batch size of 256 was selected and the network was trained with 128000000 task graphs of different sizes. Stochastic Gradient Descent (SGD) with Adam optimizer ($\beta1 = 0.9$, $\beta2 = 0.99$ and $\epsilon = 10{-}9$) is used. The model was implemented in Python language using TensorFlow library.

The training and testing of the algorithms was performed on a high-performance machine with a 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.

# 5 Results and Discussion

The RL-based application mapping framework is tested against multiple benchmarks. As mentioned earlier, the mapping sequence obtained by the network is further improved with use of 2-opt local search. Figure 10 illustrate the core graphs for the benchmarks used in our test.

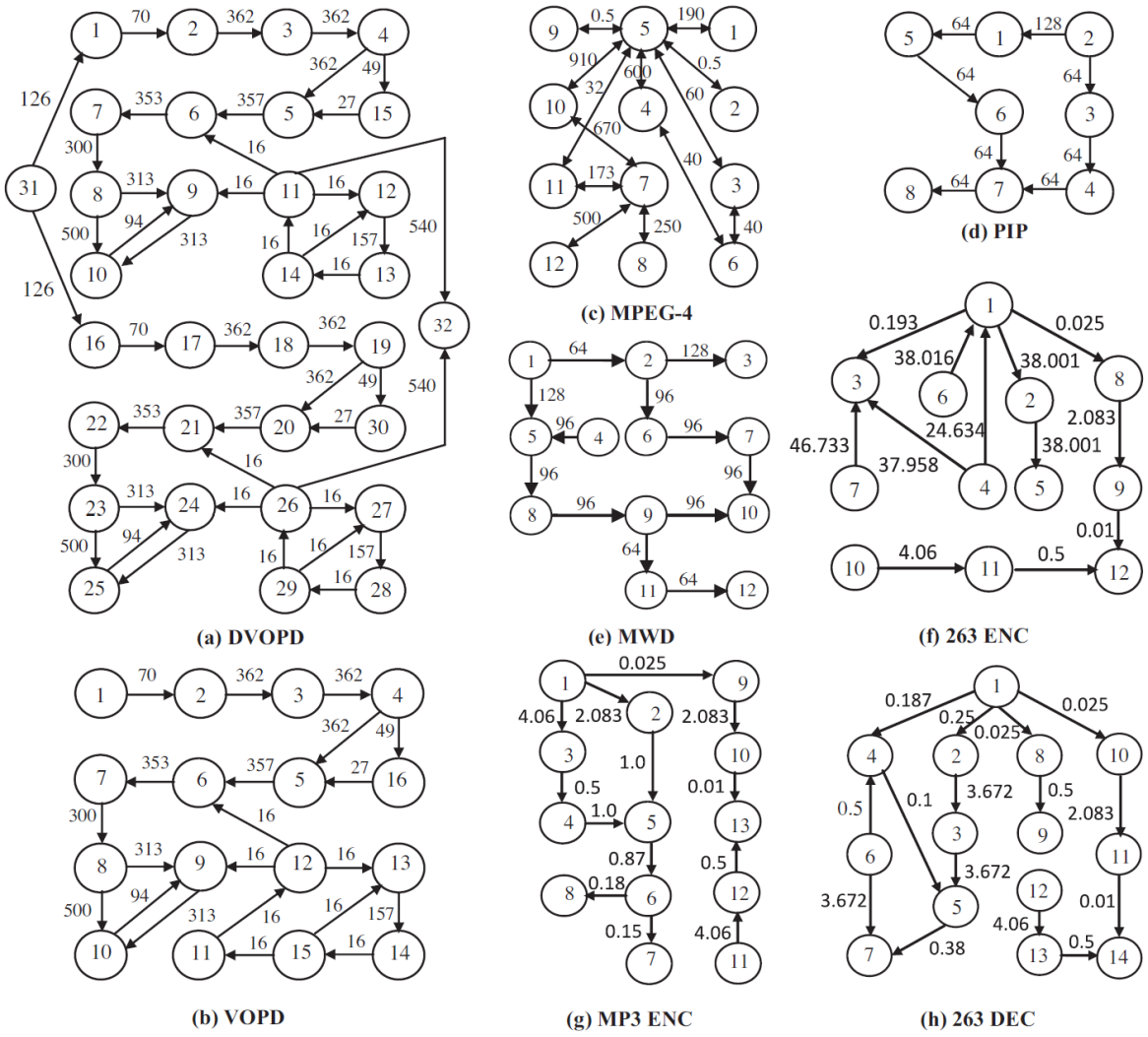



Figure 10. Application core graphs of the benchmarks [35]

Table 2 demonstrate the value of cost function for the RL algorithm and the cost obtained after applying 2-opt local search. The run-time of the algorithms is also presented in that table. We have

run the algorithm 10 times and the results presented in this table are the average of 10 runs. In the rest of this report, the cost value is rounded to the nearest integer value.

Table 2. Comparison of our model's 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 |
| VOPD | 16 | 8106 | 0.4 | 4730 | 12.93 |
| DVOPD | 32 | 20194 | 0.4 | 14046 | 42.03 |

As we can see the mapping obtained from the RL algorithm is not optimal. However, using 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 different number of cores as the network architecture does not depend on this parameter. On the other hand, the run-time of the algorithm leveraging 2 opt search has increased substantially as the number of cores have increased. The reason is that as the complexity of the problem increases with 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 also 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 3.

Table 3. 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 | 42.52 | 37038 | 82.98 |

Figure 11 illustrate the cost of different algorithms normalized to value of cost obtained from our model. Figure 12 also shows the run-times normalized to the run-time of our method.

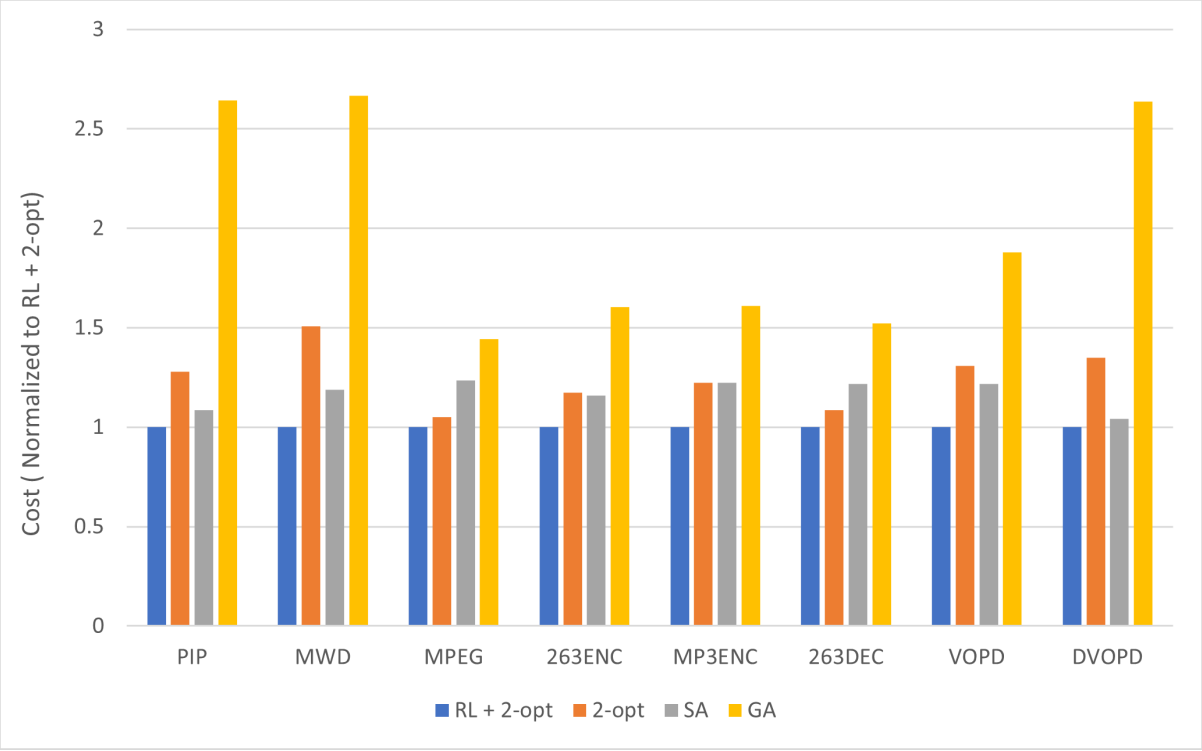



Figure 11. Cost of different algorithms normalized to RL+2-opt

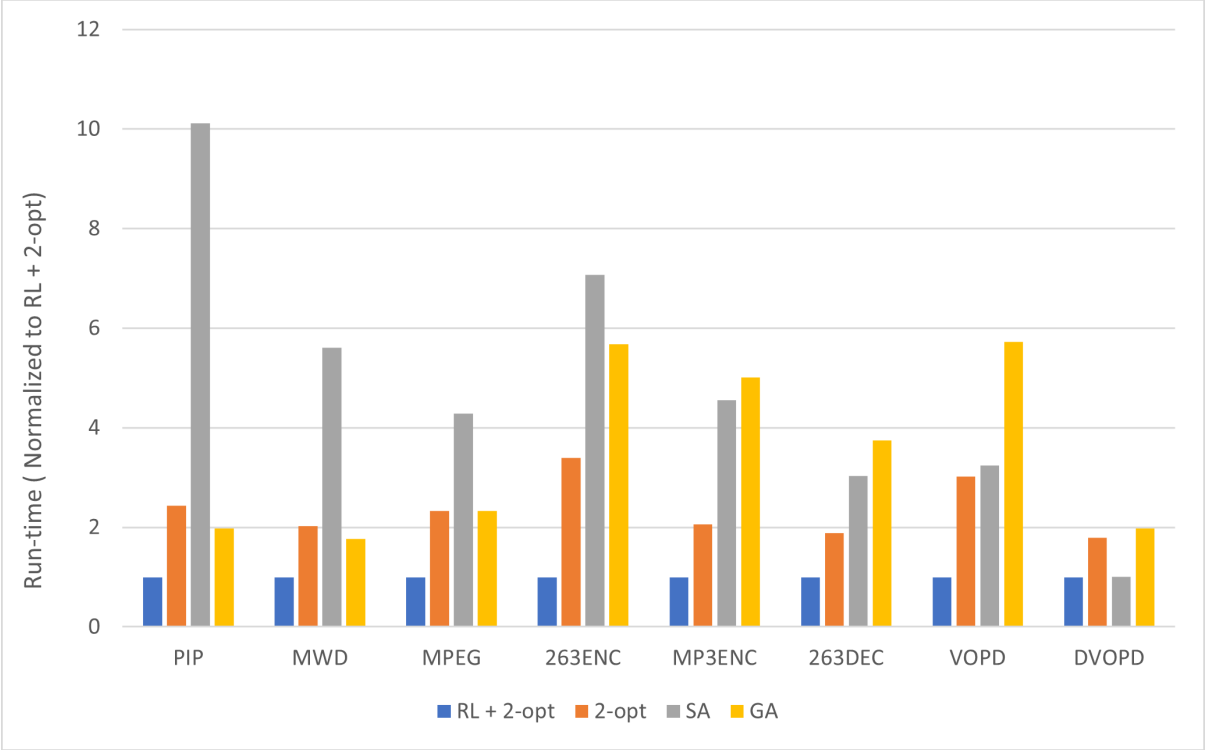



Figure 12. Algorithm's run-time normalized to RL+2-opt

From figure 11 we can observe 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. According to Figure 12 we can also see that the run-time of our algorithm is also lower than the 2-opt algorithm alone. The reason is that our RL network was able to provide a good estimate of the optimum mapping to the 2-opt algorithm. As a result, the 2-opt algorithm finds the optimum mapping in fewer number of iterations.

Figure 13 the number of turns per each benchmark application for different algorithms. For most of the benchmarks our method results in fewer turns.

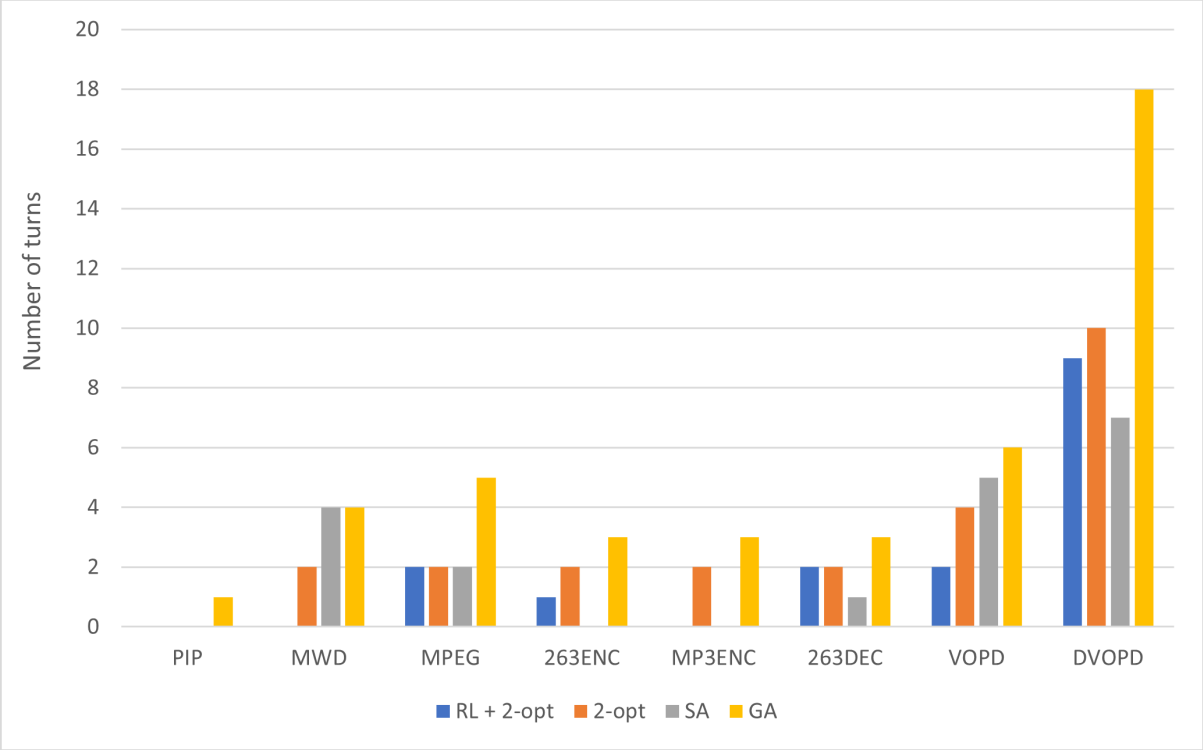



Figure 13. Number of turns for different algorithms

As mentioned earlier our goal was to optimize the mapping algorithm for fewer turns as well as reducing the distance between mapped cores. As can be seen from figure 14 our method also outperforms the other methods in terms of average number of hops between cores.

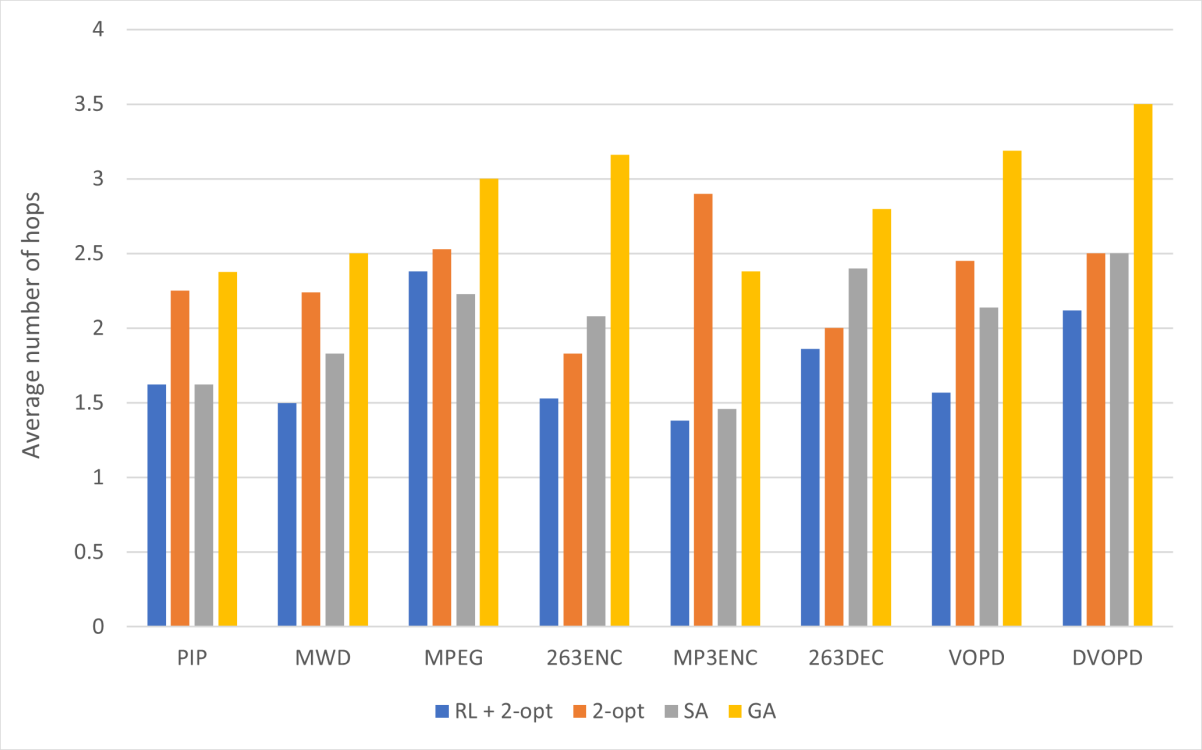



Figure 14. Average number of hops for different algorithms

# 6 Conclusion

With the advancement of technology newer chips are becoming more compact and power-hungry. As Static power is becoming the dominant source of power consumption in advanced chips, Static power reduction methods are getting more attention. Power-gating is a promising technique to reduce the static power of routers in NoCs which are an important component in MPSoCs. However, power-gating can cause a substantial performance degradation in terms of network latency. In this article 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. The results show that using the RL framework with the 2-opt local search has the best performance in terms of overall cost. The reason is that our RL network was able to predict the heuristics of the problem and provide a good estimate of the optimum mapping to the 2-opt algorithm. The overall cost of mapping sequences obtained by our method was reduced 24%, 17% and 100% comparing to generic 2-opt, SA and GA algorithms.

# 7 Future work

As we can see from the results, the combination of the RL framework with the 2-opt search has a good performance in terms of overall cost. However, we can see that there is still a considerable gap between the cost achieved by the RL network and the cost after applying 2-opt search. In other words, our RL network has only achieved an approximation of the optimum mapping which was later fine-tuned by the 2-opt local search. The reason behind this might be related to the input embedding layer we have used in our network. As mentioned earlier this embedding layer is a very simple graph convolutional neural network and consequently the node representations provided by this layer may not be differentiating enough. As a scope for feature work, deploying more complex graph embedding architectures can be considered to further enhance the mappings obtained by the RL network.

# 8 References


[1] Emmanuel Ofori-Attah and Michael Opoku Agyeman. 2018. A survey of power-aware Network-on-Chip design techniques. In 13th International Multi-Conference on Computing in Global Information Technology. IARIA.

[2] L. Benini and G. De Micheli. 2002. Networks on chips: a new SoC paradigm. 35, 1 (2002).

[3] Mohamed S. Abdelfattah and Vaughn Betz. 2013. The power of communication: Energy-efficient NOCS for FPGAS. In 23rd International Conference on Field-programmable Logic and Applications. IEEE, 1–8.

[4] R. Aligholipour, M. Baharloo, B. Farzaneh, M. Abdollahi, and A. Khonsari, "TAMA," ACM Transactions on Embedded Computing Systems, vol. 20, no. 5, pp. 1–24, Sep. 2021

[5] Lizhong Chen and Timothy M. Pinkston. 2012. NoRD: Node-router decoupling for effective power-gating of on-chip routers. In 45th IEEE/ACM International Symposium on Microarchitecture. IEEE, 270–281.

[6] Mohammad Sadrosadati, Seyed Borna Ehsani, Hajar Falahati, Rachata Ausavarungnirun, Arash Tavakkol, Mojtaba Abaee, Lois Orosa, YaohuaWang, Hamid Sarbazi-Azad, and Onur Mutlu. 2019. ITAP: Idle-time-aware power management for GPU execution units. ACM Trans. Archit. Code Optim. 16, 1 (2019), 1–26.

[7] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In 38th International Symposium on Computer Architecture (ISCA). IEEE, 365–376.

[8] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, et al. 2008. Tile64-processor: A 64-core soc with mesh interconnect. In IEEE International Solid-state Circuits Conference–Digest of Technical Papers. IEEE, 88–598.

[9] Praveen Salihundam, Shailendra Jain, Tiju Jacob, Shasi Kumar, Vasantha Erraguntla, Yatin Hoskote, Sriram Vangal, Gregory Ruhl, and Nitin Borkar. 2011. A 2 Tb/s 6*4 mesh network for a single-chip cloud computer with DVFS in 45 nm CMOS. IEEE J. Solid-state Circ. 46, 4 (2011), 757–766.

[10] Michael Bedford Taylor, Jason Kim, Jason Miller, DavidWentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee,Walter Lee, et al. 2002. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. IEEE Micro 22, 2 (2002), 25–35.

[11] Mario R. Casu and Paolo Giaccone. 2017. Power-performance assessment of different DVFS control policies in NoCs. J. Parallel Distrib. Comput. 109 (2017), 193–207.

[12] Yuan Yao and Zhonghai Lu. 2016. DVFS for NoNs in CMPs: A thread voting approach. In IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 309–320.

[13] Lizhong Chen, Di Zhu,Massoud Pedram, and Timothy M. Pinkston. 2015. Power punch: Towards non-blocking powergating of NoC routers. In IEEE 21st International Symposium on High Performance Computer Architecture (HPCA). IEEE, 378–389.

[14] Reetuparna Das, Satish Narayanasamy, Sudhir K. Satpathy, and Ronald G. Dreslinski. 2013. Catnap: Energy proportional multiple network-on-chip. ACM SIGARCH Comput. Archit. News 41, 3, 320–331.

[15] Dominic DiTomaso, Ashif Sikder, Avinash Kodi, and Ahmed Louri. 2017. Machine learning enabled power-aware network-on-chip design. In Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 1354–1359.

[16] Davide Zoni andWilliam Fornaciari. 2015. Modeling DVFS and power-gating actuators for cycle-accurate NoC-based simulators. ACM J. Emerg. Technol. Comput. Syst. 12, 3 (2015), 1–24.

[17] Lizhong Chen, Di Zhu, Massoud Pedram, and Timothy M. Pinkston. 2016. Simulation of NoC power-gating: Requirements, optimizations, and the agate simulator. J. Parallel Distrib. Comput. 95 (2016), 69–78.

[18] Radu Marculescu, Jingcao Hu, and Umit Y. Ogras. 2005. Key research problems in NoC design: a holistic perspective. In 2005 Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'05).

[19] Pradip Kumar Sahu and Santanu Chattopadhyay. 2013. A survey on application mapping strategies for Network-on-Chip design. 59, 1 (2013)

[20] K. Srinivasan, K.S. Chatha, and G. Konjevod. 2006. Linear-programming-based techniques for synthesis of network-on-chip architectures. 14, 4 (2006), 407 - 420.

[21] Pavel Ghosh, Arunabha Sen, and Alexander Hall. 2009. Energy efficient application mapping to NoC processing elements operating at multiple voltage levels. In 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip. 80-85.

[22] Suleyman Tosun, Ozcan Ozturk, and Meltem Ozen. 2009. An ILP formulation for application mapping onto Network-on-Chips. In 2009 International Conference on Application of Information and Communication Technologies. IEEE, Baku, Azerbaijan, 1-5.

[23 Max Farias, Edna Barros, Abel Filho, André Araújo, André Silva, and João Melo. 2013. An ant colony metaheuristic for energy aware application mapping on NoCs. In Proceedings of the IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS'13). 365–368.

[24] Changqing Xu, Yi Liu, Peng Li, and YinTang Yang. 2018. Unified multi-objective mapping for network-on-chip using genetic-based hyper-heuristic algorithms. IET Computers & Digital Techniques 12, 4 (2018), 158–166.

[25] Lei Guo, Yifan Ge, Weigang Hou, Pengxing Guo, Qing Cai, and Jingjing Wu. 2018. A novel IP-core mapping algorithm in reliable 3D optical network-on-chips. Optical Switching and Networking 27 (2018), 50–57.

[26] Pradip Kumar Sahu, Tapan Shah, Kanchan Manna, and Santanu Chattopadhyay. 2014. Application mapping onto mesh-based network-on-chip using discrete particle swarm optimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 22, 2 (2014), 300–312.

[27] C. Marcon, A. Borin, A. Susin, L. Carro, and F. Wagner. 2005. Time and energy efficient mapping of embedded applications onto NoCs. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'05), Vol. 1. 33–38.

[28] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural combinatorial optimization with reinforcement learning. arXiv abs/1611.09940.

[29] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. 2018. Learning heuristics for the TSP by policy gradient. In Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Willem-Jan van Hoeve (Ed.). Springer International Publishing, Cham, 170–181.

[30] S. Jagadheesh, P. V. Bhanu, and S. J., "NoC Application Mapping Optimization Using Reinforcement Learning," ACM Transactions on Design Automation of Electronic Systems, vol. 27, no. 6, pp. 1–16, Nov. 2022

[31] Samira Saeidi, Ahmad Khademzadeh, and Fatemeh Vardi. 2009. Crinkle: A heuristic mapping algorithm for network on chip. IEICE Electron. Exp. 6, 24 (2009), 1737–1744.

[32] N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," arXiv:1609.02907 [cs, stat], Feb. 2017,

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[34] Matthias Englert, Heiko Röglin, and Berthold Vöcking. 2014. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. Algorithmica 68, 1 (2014), 190–264.

[35] 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, Aug. 2014,