Reinforcement Learning for Application mapping on Network on chip

Project report submitted
In fulfilment of the requirements for

M. Tech Project 1 (EC57003)

for the degree of Masters of Technology in

Visual Information Processing & Embedded Systems

By

Arun Sammit Pandey
(17EC35006)

Under the supervision of
Prof. Sudipta Mahapatra

Department of Electronics & Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Declaration

I certify that

a) The work contained in this report has been done by me under the guidance of my supervisor

b) The work has not been submitted to any other Institute for any degree or diploma

c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: November 22, 2021
Place: Kharagpur

Arun Sammit Pandey

(17EC35006)

Department of Electronics & Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Certificate

This is to certify that the project report entitled "Reinforcement Learning for Application mapping on Network on chip" submitted by Arun Sammit Pandey (Roll No. 17EC35006) to Indian Institute of Technology Kharagpur towards the fulfilment of requirements of M. Tech Project 1 (EC57003) is a record of bona fide work carried out by him under my supervision and guidance during Autumn Semester, 2021-22

[illegible]

Prof. Sudipta Mahapatra
Department of Electronics & Electrical
Communication Engineering,
Indian Institute of Technology Kharagpur
Kharagpur – 721302, India

Date: November 22, 2020
Place: Kharagpur

Acknowledgment

I would like to thank my guide, Prof. Sudipta Mahapatra, for his exceptional guidance and support, without which this project would not have been possible. He has always motivated me to explore as much as I can. He has always supported me through whatever problems I faced during the project

I would also like to thank Mr. Sambangi Ramesh, research scholar in the Embedded Systems Lab, for his help throughout this work. He was the one who pointed me to specific research papers when I was a bit confused regarding the direction in which I should start working. He also gave specific and to the point advice in my weekly catch-up meetings whenever I got stuck in any place during my work

I would like to thank all of my batchmates and family members for supporting me morally during this journey

Contents

Abstract

The application mapping problem for Network-on-Chip (NoC) is an NP-hard problem is a combinatorial optimisation problem. Because of the computational complexity of this problem, it is impossible to get an optimal mapping by enumeration of all possible solutions. Heuristics-based methods like transformative heuristics and constructive heuristics, which are generally used to find the solutions to large scale application mapping problems, also get stuck in a local optimal solution sometimes and thus result in a sub-optimal mapping. Due to these reasons, a policy gradient-based reinforcement learning approach has been proposed in this report which uses the Neural Networks based architecture (Message Passing Neural Networks + Pointer Networks) to learn a near to optimal policy for a single given Core Graph. Further, a pretraining approach has also been suggested, which can reduce the time required for getting the optimal mapping

Chapter 1

Introduction

1.1 Network on Chip (NoC)

Network on Chip refers to the interconnection network which connects various IP cores, caches, memory and controllers. Network-on-Chip (NoC) is an alternative way of performing communication between IP cores, caches, memory etc. which has been traditionally done via buses. NoCs are also scalable because they provide scalable bandwidth at low power and low area. NoCs are quite effective with respect to their use of wiring and multiplexing of many traffic flows on the same channels improving the Quality of Service and bandwidth.

In an NoC, the Intellectual Property cores (known as IP cores) are interconnected via a set of shared links using a fabric of routers. Each core is assigned to a router as displayed in Fig 1.1. The topology of a NoC refers to the arrangement of these routers and channels connecting them. The 2D mesh topology is the most common topology used in NoCs, particularly because of the simplicity, regularity, scalability and ease of analysis. The router accepts and forwards each incoming packet to implement the routing policy. The switching policy and arbitration policy determines the manner in which multiple packets contending for the same link are forwarded through the network. Packet switching is mostly used in NoCs having a large number of cores.

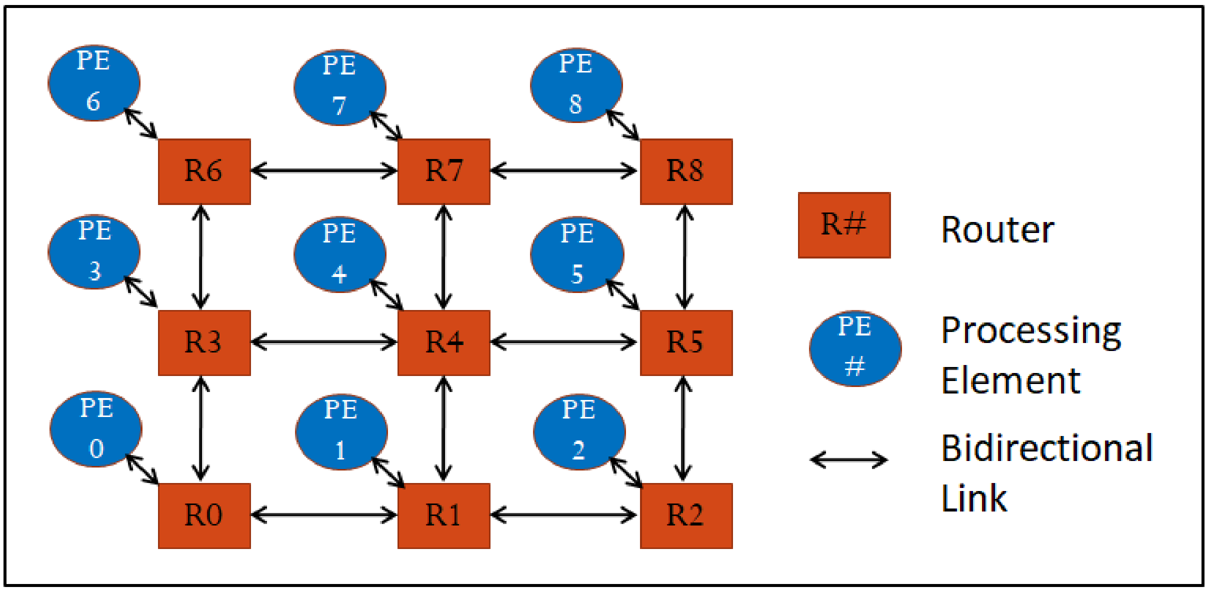

Figure 1.1 A mesh-topology based Network on Chip Architecture

1.2 Application Mapping Problem in NoC

Application mapping problem involves choosing the nodes (which contains routers) of the given NoC topology corresponding to each core of the given application. This process of selection is done in such a way that a specific metric corresponding to NoC can be optimised. There are various metrics which are important for the design of application mapping algorithms for NoC. Some of these are communication cost, latency, communication energy etc. This problem is Non-Polynomial (NP) hard since corresponding to n IP cores and n NoC nodes, we can have a total of $n!$ possible mappings.

Chapter 2

Literature Survey

The latest survey for application mapping on Network on Chip can be found in [1]. The application mapping methods have gathered considerable Attention from the research community. These methods can be classified into three categories, i.e. exact methods, transformative heuristics methods, constructive heuristics methods

2.1 Exact Methods

There are a few mathematical programming-based algorithms that solve the application mapping problem exactly. In one of these methods, the application mapping problem is transformed into Integer Programming Problem (ILP) or the Mixed Integer Programming Problem (MILP). In another such algorithm named branch and bound algorithm [2], a systematic search procedure is adopted, which is more efficient than a full, exhaustive search. However, these solutions can only be used for solving Application mapping problems with very few cores.

2.2 Transformative Heuristics Methods

These algorithms generally adopt or modify methods based on PSO (Particle Swarm Optimization), Genetic Algorithms (GA) or Ant Colony Algorithm. In [3] The authors use a discrete particle swarm-based strategy to map applications on both 2D and 3D meshes using communication cost as the optimisation metric. In [4], the authors have used a genetic-based hyper-heuristic algorithm (GHA) for application mapping while optimising energy consumption and communication latency.

2.3 Constructive Heuristics Methods

Such algorithms are based on the construction of a prior solution using some model or method, or assumptions, after which the preceding solution is improved using an iterative procedure. One example of this kind of method is the work mentioned in [9]. In this work the authors has proposed an algorithm which adopts an ordering of edges for constructively creating a solution. The most recent works for this class of algorithms is [5], [6]. In [5], the authors propose a probability model based on a neural network (NNMM) which is based on MPNN (Message Passing Neural Networks) [7] and Pointer Networks [8]. This model is then used to sample multiple candidate solutions obtaining a preliminary solution with the desired best NoC metric. Then that prior solution is further improved using the 2-Opt local search heuristics algorithm. The NoC metric used by authors in this work was communication cost. In [6], the authors use the same probability model but train it using reinforcement learning instead of training it with supervised learning. In addition to that, they use PSMAP [10] and GA (Genetic Algorithm) for improving the preliminary solution instead of the 2-Opt algorithm.

Chapter 3

Problem Formulation

3.1 Mathematical Description

We can mathematically describe the Application mapping problem using core graph $G(C, E)$ and topology graph $T(R, L)$. The core graph, $G(C, E)$, is a directed and weighted. The topology graph $T(R, L)$ is an undirected unweighted graph. Let's assume that there are n cores to which the tasks corresponding to our application has been mapped. The nodes of the core graph $c_i, i = \{1, \dots, n\}$ represents the cores corresponding to tasks of our application. The value (weight) of the directed edges $e_{i,j}$ which connects the cores i and j is represented by w_{ij}. w_{ij} represents the communication bandwidth requirement between the cores i and j. The nodes of the topology graph r_i $i = 1, \dots, n$ represents the nodes of network topology and the edge l_{ij} represents that the nodes r_i and r_j in the NoC topology are connected through which packets of data can be exchanged.

With the help of definitions given above, the application mapping process can be represented as the function map: $C \rightarrow R$, such that $\forall\ c_i \ \in C, \exists\ r_j \in R$. Also, each core c_i must be mapped to a unique topology node r_j. The previous condition also implies that this problem is only defined when $|C| \leq |R|$ i.e., the number of cores should be less than the number of NoC topology nodes. If the number of cores is less than the number of NoC topology nodes, then extra dummy cores are added to the core graph, and these cores are assumed to have no connection (or communication requirement) with other nodes. Thus, we can assume that $|C| = |R|$ without loss of generalisation. The final mapping solution can be represented as a permutation of first n natural numbers. We call this mapping sequence/solution as $\boldsymbol{m}$. The i^{th} number in the sequence m_i represents that the m_i^{th} core is mapped to i^{th} NoC topology node.

3.2 Optimisation metric

We use communication cost as the NoC metric we are going to optimise, i.e., through our algorithm, we will try to find out the mapping with the least communication cost. Let's assume that the core c_i has been mapped to topology node r_k and it has an edge e_{ij} to a core c_j which has been mapped to topology node r_l. Then we can define communication cost corresponding to edge e_{ij} as

$$comm_{ij} = w_{ij} \times dist(k, l)$$

Where $dist(k, l)$ gives the distance (or number of hops in our case since we are assuming topology graph $T(R, L)$ to be unweighted) between the nodes r_k and r_l in the topology graph. This distance can easily be calculated $\forall\ k, l = 1, \dots, n$ using any of the All-pair shortest path algorithms like Floyd-Warshall algorithm. Denoting the communication cost $comm_{ij}$ corresponding to p^{th} edge of core graph, e_{ij} as $comm_p$ where $p = 1, \dots, |E|$, we can define the total communication cost as

$$communication\ cost = \sum_{p=1}^{|E|} comm_p$$

3.3 Contributions

The work's main contribution is the proposal of a new reinforcement learning-based method to solve the application mapping problem described in the previous section. In this work, a model similar to that used in [6] has been taken after some modifications, after which the policy parameters are optimised for a given single input core graph using reinforcement learning techniques. The technique used has been inspired by the Active Search algorithm mentioned in [11]. A reinforcement learning based method for the application mapping problem has been proposed previously by the authors in [6]. In that work, the authors also use a reinforcement learning approach to train a generic stochastic policy for the entire class of application mapping problems, which is then used to sample a few candidate solutions for a given specific problem. After the sampling is done, a transformative heuristic algorithm like DPSO (discrete particle swarm optimisation) or GA (Genetic Algorithm) generates the final solution. Only a part of the initial population is taken from the previous probability model/policy. Thus, their method does not optimise the policy parameters for a single input core graph and use the policy with the same parameters for every input core graph they evaluate. Also, they use the reinforcement learning approach to generate the initial population and not for the whole optimisation/search process. In this work, we try to build upon their work by performing the complete optimisation/searching process using reinforcement learning based methods. Also, our method doesn’t rely on any heuristic search algorithms. Another advantage of our approach is that it can be combined with the learning approach mentioned in [6] to generate a pre-trained model. Thus, we can initialise the policy/model parameters to that of the pre-trained model when performing the optimisation process for a particular core graph. This procedure will enable us to reduce the time required for the estimation of optimal mapping by our algorithm.

The organisation of the remaining report is as follows. The upcoming chapter 4 describes the reinforcement learning approach, models used and the training process used. Then Chapter 5 describes the results obtained and the conclusions, and Chapter 6 describes the work to be conducted in the future and further possible extensions of our approach.

Chapter 4

Reinforcement Learning Approach

In the below paragraph, we try to formulate the application mapping problem as a reinforcement learning problem.

In Reinforcement Learning, particularly in policy gradient methods, we try to find the parameters of optimal stochastic policy $\pi(m|s,\theta)$ which gives the probability of performing an action m given that the agent is in a state s It can be defined mathematically as

$$\pi(m|s,\boldsymbol{\theta}) = \Pr\{A_t = m|S_t = s, \theta_t = \boldsymbol{\theta}\}$$

In the context of the application mapping problem, the actions represent the process of selecting a particular mapping out of the $n!$ possible mappings, whereas the state represent the various possible input core graph $G(C,E)$. A reinforcement learning task also involves rewards L_{t+1} which is generally a function of the previous state, $S_t = s$ and the previous action taken $A_t = m$. In the context of the application mapping problem, we can use the negative of the NoC metric that we want to minimise as the reward. We use the notation $L(m|s)$ to denote the communication cost when we perform mapping m for a given core graph $G(C,E)$. Thus, the reward would be equal to $-L(m|s)$ in this case.

4.1 Model

The application mapping problem is a combinatorial optimisation problem in which we are trying to learn to output a sequence. In the literature, many scholars have tried to solve combinatorial problems like TSP (travelling salesman problem) using Reinforcement Learning techniques. For example, in [12], the authors try to solve TSP, VRP (Vehicle Routing Problem) and Orienteering Problem (OP) using models trained by reinforcement learning techniques. A complete survey of reinforcement learning methods for combinatorial optimisation problems is present in [13].

The model which we use in this work is called MPNN-Ptr. It is a slightly modified version of the model with the same name as present in [5]. The model consists of two parts, namely MPNN (message passing neural networks)[7] & PN (Pointer Networks)[8]. Firstly, the input core graph $G(C, E)$ containing n nodes is pre-processed by the message passing neural networks (MPNN) to generate node embeddings $x_i \in \mathbb{R}^p, \forall i \in \{1, \dots, n\}$ corresponding to each node in the core graph $G(C, E)$. After that, all of these p-dimensional embeddings are fed to the pointer-networks. The pointer network's structure has been taken from [11] with minor modifications. The primary function of pointer networks is to take the sequence of input embeddings (with length = n) and output a tuple of a sequence and a probability. The sequence consists of permutation of first n natural numbers denoting the sampled mapping. The corresponding probability represents the probability of the sampled sequence. Thus, with the help of Pointer Networks & MPNN, we are successfully able to model the stochastic policy $\pi(m|s, \boldsymbol{\theta})$ from which we can sample the mapping sequences m along with its probability. The vector $\boldsymbol{\theta}$ denotes the learnable parameters of neural networks present in MPNN and Pointer Networks. In the below sections of this report, a brief explanation of MPNN and Pointer Networks is done.

MPNN: The full form of MPNN is message passing neural networks. It can be considered to be a generalisation of convolution operator to irregular domains. MPNN can be described using two stages. The first stage is message passing (repeated k number of times as k layers), and the second stage is the readout stage. Each message passing layer consists of three sub-stages. Let's assume that x_i^k denotes the features of i^{th} node in the k^{th} message passing layer and $e_{j,i}$ denotes the features of directed edge from j^{th} node to i^{th} node. Then the message passing for getting the node features in the k^{th} layer can be represented as:

$$x_i^{(k)} = \gamma^{(k)}(x_i^{(k-1)}, op_{j\in\mathcal{N}(i)}\phi^{(k)}\left(x_i^{(k-1)}, x_j^{(k-1)}, e_{j,i}\right))$$

In the above equation $\gamma^{(k)}$ and $\phi^{(k)}$ represents the node update function and message function for k^{th} layer respectively. These are generally implemented as MLPs (Multi-Layer Perceptrons). op represents one of the message aggregation schemes such as mean, add or max. After all the K message passing stages has been completed, the node embeddings can be aggregated to output a feature vector for the whole graph $G(C,E)$. The below equation can represent this stage

$$\hat{g} = rout(\{x_v^{(k)} | v \in G\}$$

Here $\hat{g}$ represents the feature vector for the whole graph.

The exact architecture of MPNN, which we used in our model, can be described as follows:

The node features, w_v of core c_v present in the core graph $G(C,E)$ are initialised by the bandwidth requirement of that core with other cores. w_{ij} denotes the weight of the edge e_{ij} as mentioned previously

We use one message passing layer directly on the input node features to calculate one set of embeddings ξ_v corresponding to each node v (we will directly use this in subsequent message passing layers to calculate the final node embeddings)

$$\xi_v = relu(\theta_3\left[\frac{1}{|N(v)|}\sum_{u\in N(v)} relu(\theta_2[w_{uv}, w_u])\ , |N(v)|\right])$$

Here, $\theta_2 \in \mathbb{R}^{m+1\times p-1}$,$\theta_3 \in \mathbb{R}^{p\times p}$

We initialise the embedding vector using

$$x_v^0 = relu(\theta_1 w_v)$$

Here $w_v \in \mathbb{R}^m$ and $\theta_1 \in \mathbb{R}^{m\times p}$

After this K rounds of message-passing is performed through K layers which can be represented by:

$$m_v^{k+1} = relu(\theta_{4,k}\left[\frac{1}{N(v)}\sum_{u\in N(v)} w_{uv}x_u^k\,, \xi_v\right])$$

$$x_v^{k+1} = relu(\theta_{5,k}[x_v^k, m_v^{k+1}])$$

In the above equations, $\theta_{4,k}$ and $\theta_{5,k}$ belongs to $\mathbb{R}^{2p\times n}$. After all the K rounds of message-passing have been completed, the final embeddings are calculated via the equations,

$$s_v = \theta_7[relu\left(\theta_6\frac{1}{|C|}\sum_{u\in C} x_u^K\right), \mu_v^K\ ]$$

Here, $\theta_6 \in \mathbb{R}^{p\times p}$ and $\theta_7 \in \mathbb{R}^{2p\times p}$

Thus, after this step, we obtain the required embeddings vectors, s_v corresponding to each core v in the core graph $G(C, E)$. These embeddings are then fed into Pointer Networks to model the stochastic policy $\pi(m|s)$. Here s represents the sequence of embeddings s_v. The Pointer Networks are briefly described below:

Pointer Networks:

The pointer networks consist of three sub-modules: an Encoder, a Decoder and an Attention module. Encoder and Decoder consist of Long Short-Term Memory (LSTM) cells. The encoder reads the input embeddings $s_v \in \mathbb{R}^p$ one at a time to generate outputs e_v $v \in \{1, \dots, n\}$ corresponding to each input embedding. The Decoder takes the last hidden state, h_n and cell state, c_n of the encoder as its initial hidden and cell state and a vector $\langle g \rangle$ as its input which we set as a learnable parameter of the model. The inputs to the later LSTM cells of the Decoder are the input embedding, selected according to the probability distribution generated by the Attention module. The Attention module generates this probability distribution with the help of encoder outputs and previous decoder output. The exact calculation steps are described below

The input to attention module at i^{th} step of Decoder is the output d_i of current Decoder, and the whole set of encoder outputs $\{e_1, e_2, \dots, e_n\}$. The attention module then calculates an integer u_j corresponding to each encoder output e_j as follows,

$$u_j = \begin{cases} v^T \cdot \tanh\left(W_e e_j + W_d d_i\right), & if\ j \neq m(k)\ \forall\ k < j \\ -\infty & otherwise \end{cases}$$

Here, $m(k)$ represents the k^{th} value in the mapping vector m which denotes the mapping calculated by k^{th} decoder unit.

After this, the probability of mapping j^{th} core with i^{th} topology node, $p(m(i) = j|m(< i), s)$ is calculated as,

$$p(m(i) = j|m(< i), s) = softmax(C \tanh(u))_j$$

Where u represents the vector consisting of all the u_j calculated earlier. The $\tanh u$ hyperbolic function with scaling factor C is used for clipping the logits to the range $[-C, C]$ to encourage the model to perform more exploration of possible mappings by restricting its entropy [11].

Thus, according to the above probability, the output of i^{th} decoder step is sampled. Let's assume after sampling, j^{th} input/core has been chosen. Then the input to the next decoder LSTM cell would be the j^{th} input embedding s_j

Hence, we can calculate the optimal policy, $\pi(m|s,\boldsymbol{\theta})$ by the help of the below equation,

$$\pi(m|s,\boldsymbol{\theta}) = \prod_{i\in\{1,\ldots,n\}} p(m(i)|m(<i),s)$$

4.2 Training Process

The algorithm used to find the parameters of MPNN and Pointer Networks for getting the optimal policy $\pi_{optim}(m|s,\boldsymbol{\theta})$ is based on the Active Search algorithm mentioned in [11], which comes under the general class of algorithms known as "Policy Gradient Methods" [17]. It has been described in the below paragraph:

Proposed Algorithm:

We consider the expected communication cost of the policy $\pi(m|s,\boldsymbol{\theta})$ as the training objective which we want to minimise. Thus, the training objective can be defined mathematically as:

$$J(\boldsymbol{\theta} \mid s) = \mathbb{E}_{\pi(m|s,\,\theta)}(L(m|S))$$

The parameters, $\boldsymbol{\theta}$ can be optimised by using policy gradient methods. We estimate the gradient of J using the REINFORCE algorithm [14] as:

$$\nabla_\theta J(\theta|s) = \mathbb{E}_{\pi(m|s,\,\theta)}\left[\left(L(m|s) - b(s)\right)\nabla_\theta \ln\left(\pi(m|s)\right)\right]$$

For our purposes of finding the optimal mapping for a given core graph $G(C, E)$, the above expression can be approximated via sampling n mappings from the model as follows:

$$\nabla_\theta J(\theta|s) = \frac{1}{n} \sum_{i \in \{1,\dots n\}} \left(\left(L(m_i|s) - b(s)\right) \nabla_\theta \ln\left(\pi(m_i|s)\right)\right)$$

The baseline can be estimated as the exponential moving average of the communication cost obtained. Thus, we can now formulate our active search algorithm as follows

ALGORITHM 1: ACTIVE SEARCH FOR APPLICATION MAPPING

```
    Input: $\theta, n_{epochs}, n$
    Output: $m_{best}, L_{best}$
1   for i = 1 to $n_{epochs}$ do
2       $\boldsymbol{m} \leftarrow$ SAMPLEMAPPINGS($\pi_\theta(m|s), n$)
3       $\boldsymbol{m_{best}}, \boldsymbol{L_{best}} \leftarrow$ SELECTBEST( $m_{best}, L_{best}, \boldsymbol{m}$)
4       if i == 1 then
5           $b \leftarrow \left(\frac{1}{B}\sum_{i=1}^{B} L_i\right)$
6       else
7           $b \leftarrow \alpha \times b + (1-\alpha) \times \left(\frac{1}{B}\sum_{i=1}^{n} L_i\right)$
8       end if
9       $\nabla_\theta J(\theta|s) \leftarrow \left(\frac{1}{n}\sum_{i=1}^{n}(L(m_i|s) - b)\nabla_\theta \ln\left(\pi(m_i|s)\right)\right)$
10      $\theta \leftarrow Adam\left(\theta, \nabla_\theta J(\theta|s)\right)$
11  end for
```

The input to the above algorithm is a randomly initialised set of parameters θ for our model $\pi(m|s, \boldsymbol{\theta})$, number of epochs n_{epochs} and the number of solutions n which we should sample during each epoch. $\boldsymbol{m}$ contains all the n sampled mappings. m_{best} denotes the mapping with the least communication cost and L_{best} denotes the corresponding communication cost seen across all iterations of our algorithm. b refers to the baseline, which is updated using the coefficient of exponential averaging α. Finally, $Adam$ refers to the optimiser described in [15]

Chapter 5

Results

5.1 Dataset Generation

Directed Erdős-Rényi graphs of various nodes (42, 49, 56, 64) were generated using NetworkX library[1]. The probability of edge creation was kept as .3 for graphs of all sizes. The communication bandwidth requirement for each directed edge was also generated randomly using a lognormal distribution with parameters $\mu = 1$ & $\sigma = 3$. The network topology was assumed to be a 2D mesh. The number of rows and columns of mesh for a given generated core graph of size x was calculated using the formulas:

$$n = \lceil \sqrt{x} \rceil$$

$$m = \left\lceil \frac{x}{n} \right\rceil$$

Here n denotes the number of rows and m denotes the number of columns in the mesh topology. Finally, the distance matrix corresponding to the topology was calculated using Floyd-Warshall algorithm, and it was fed to the training algorithm along with the generated core graph. The generated core graphs are present in the link[2]

5.2 Hyperparameters & Training

[1] https://networkx.org/
[2] https://github.com/arunsammit/MPNN-Ptr/tree/master/data

The embedding dimension p of the message passing neural networks were taken as 72. Three rounds of message passing were conducted, i.e., $K = 3$ The dimension of hidden units of LSTM were taken as 80. 4 layers of LSTM units were stacked together. The dropout was taken to be 0. For each core-graphs of size larger than or equal to 49, number of epochs, n_{epochs} were taken to be 4000 and for core-graphs of size less than 49 number of epochs, n_{epochs} was taken to be 2500. The number of application mapping solutions sampled in each epoch was taken to be 128. The exponential averaging factor α for updating baseline was taken to be 0.99. Finally, the learning rate was taken to be 0.001. The training was done on Google Colab with Tesla P100 GPU and 12.69 GB RAM. The code[3] was written using the Pytorch-geometric[4] library (used for message passing neural networks) and Pytorch framework. Figures 5.1 and 5.2 represent the typical plot obtained when the algorithm is run for the core graph of sizes 49 and 56, respectively. Here y-axis represents the least obtained communication cost among all mappings sampled during an epoch. The x-axis represents the epoch number. It took somewhere between 15-30 minutes (depending on the size of the core graph) to get the optimal mapping/solution for each core graph

[3] https://github.com/arunsammit/MPNN-Ptr
[4] https://pytorch-geometric.readthedocs.io/

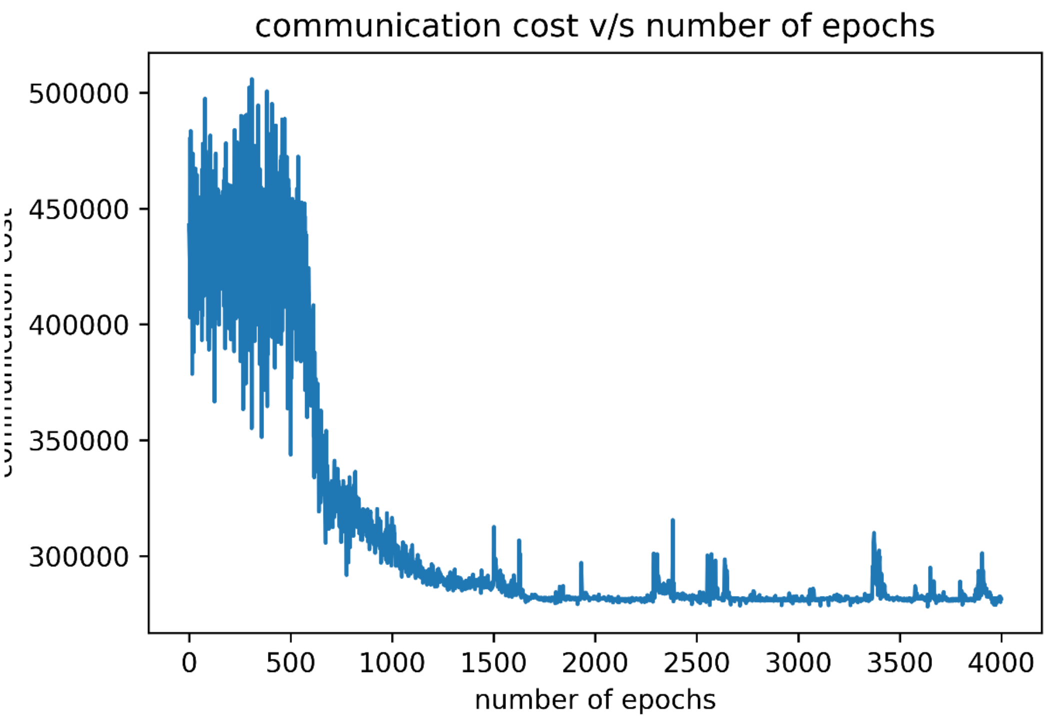

Figure 5.1: Core Graph size: 49

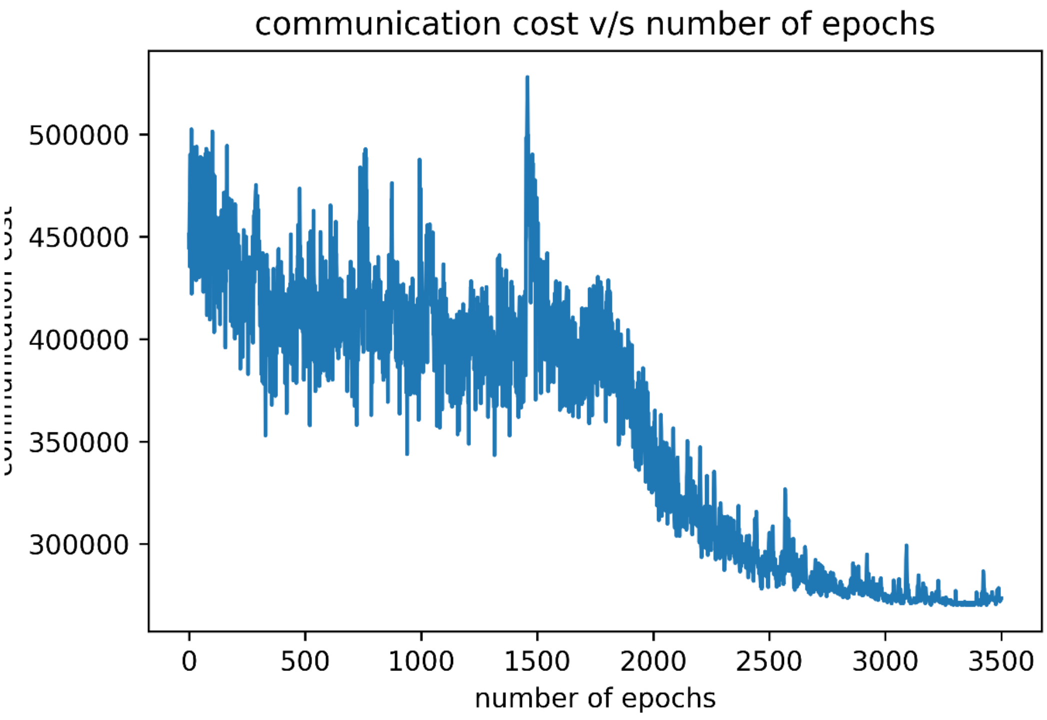

Figure 5.2: Core Graph size: 56

5.3 Output & Comparison

For our proposed algorithm, the optimal communication cost was obtained after the training procedure given above. We also run a modified version of the DPSO algorithm taken from [16] on the generated core graphs for comparison purposes. This version of DPSO uses a doping approach for updating the particles depending on their fitness value (communication cost). The DPSO algorithm was run for 2000 generations, and the population size of particles in each generation was kept as 100. The table below contains the communication cost corresponding to the most optimal mapping found by the two algorithms.

Core Graph Size	Communication Cost (DPSO)	Communication Cost (Proposed Algorithm)	% Decrease in comm. Cost
42	187374	162393	13.33
49	322199	267977	16.82
56	360706	272138	24.55%
64	607548	406593	33.07%

Table 5.1: Comparision of our algorithm with DPSO

5.4 Conclusion

From the above table 5.1 it is clear that the mappings found out by our proposed reinforcement learning approach is better than the mappings found out by the best available heuristic methods like the modified DPSO. We also observe that the percentage reduction in communication cost increases as the size of the core graph increases. It may be happening because of the fact that as the size of the core graph increases, the number of possible mappings blow up exponentially, which can introduce several local minima in the solution space, thus making it difficult for completely randomised heuristics algorithms like DPSO to search for mappings with lower costs as they could get stuck in local minima.

Chapter 6

Extensibility of the Work

The work can be extended in several ways. Below we mention three possible ways of extending this work.

6.1 Pretraining for initial model parameters

To reduce the time we take to get the optimal mapping for a single given core graph, we can pre-train our model using the approach mentioned in [6]. The pretraining can be done using datasets containing core graphs of various sizes, which we have already generated[5]. The training process, particularly the baseline estimation, would be different for the pretraining part. Reinforcement learning methods like Actor-Critic methods, which require a separate model to estimate the baseline (as the expected communication cost of the solutions generated by our modal), should be more suitable for the pretraining part. The pretraining process may also require changing the model we are currently using to make sure that the model works for core graphs of different sizes. This problem is further explored in section 6.3. We are currently evaluating this technique and comparing its performance/running time with other algorithms.

6.2 Optimising Network Latency

Network latency can be defined as the time required for a packet to traverse the NoC from input to output port. It is proportional to the number of hops times the bandwidth requirement only in cases where there is no congestion in the network. This assumption

[5] https://github.com/arunsammit/MPNN-Ptr/blob/master/data/data_81.pt

holds only if the injection rate is negligible. The authors in [16] have used Deep Neural Networks to develop a latency prediction model taking network congestion into account. We can use that model with our reinforcement learning approach to optimise latency directly instead of indirectly doing it via communication cost.

6.3 Using NoC Topology Graph as features

The model which has been used in our work only taken Core Graph $G(C, E)$ as input vectors for modelling the stochastic policy $\pi(m|s, \boldsymbol{\theta})$.This works well when finding the optimal mapping corresponding to only one particular Core Graph. However, for the pretraining part mentioned in section 6.1, the policy must be trained for core graphs of multiple sizes. This means that the corresponding NoC topology to which those core graphs are mapped are of different sizes (different numbers of rows and columns in case of a 2D mesh NoC topology). Thus we must also consider the NoC topology graphs as a part of our policy's state. This implies that the modal should be changed to accept NoC topology graphs as inputs along with core graphs. This formulation will also enable us to use different topologies like 3D mesh etc., for the same core graph with the pretraining algorithm.

Bibliography

[1] W. Amin et al., “Performance evaluation of application mapping approaches for network-on-chip designs,” IEEE Access, vol. 8, pp. 63607–63631, 2020

[2] S. Khan, S. Anjum, U. A. Gulzari, M. K. Afzal, T. Umer, and F. Ishmanov, “An efficient algorithm for mapping real time embedded applications on NoC architecture,” IEEE Access, vol. 6, pp. 16324–16335, 2018.

[3] P. K. Sahu, T. Shah, K. Manna, and S. Chattopadhyay, “Application mapping onto mesh-based network-on-chip using discrete particle swarm optimisation,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 22, no. 2, pp. 300–312, Feb. 2014.

[4] C. Xu, Y. Liu, P. Li, and Y. Yang, “Unified multi-objective mapping for network-on-chip using genetic-based hyper-heuristic algorithms,” IET Comput. Digit. Techn., vol. 12, no. 4, pp. 158–166, Jul. 2018.

[5] Q. Chen, W. Huang, Y. Zhang, and Y. Huang, “An IP core mapping algorithm based on neural networks,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 29, no. 1, pp. 189–202, Jan. 2021.

[6] Q. Chen, W. Huang, Y. Zhang, and Y. Huang “A Reinforcement Learning-Based Framework for Solving the IP Mapping Problem” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol 29, no. 9 pp 1638-1651, 2021

[7] J. Gilmer, S. S. Schoenholz, P. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in Proc. Adv. Neural Inf. Process. Syst., pp. 1263–1272, 2017.

[8] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 28, pp. 2692–2700, 2015.

[9] P. K. Sahu, K. Manna, T. Shah, and S. Chattopadhyay, “A constructive heuristic for application mapping onto mesh based network-on-chip,” J. Circuits, Syst. Comput., vol. 24, no. 8, Art. no. 1550126, 2015.

[10] P.K. Sahu, P. Venkatesh, S. Gollapalli, S. Chattopadhyay, “Application mapping onto mesh structured Network-on-Chip using particle swarm optimisation”, IEEE International symposium on VLSI (ISVLSI), 2011, pp. 335–336

[11] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural combinatorial optimisation with reinforcement learning,” 2016, arXiv:1611.09940.

[12] W. Kool, H. Van Hoof., M. Welling., “Attention, learn to solve routing problems!”, 2019, arXiv:1803.08475

[13] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev “Reinforcement learning for combinatorial optimisation: A survey”, Computers & Operations Research, Volume 134, 2021, 105400

[14] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Machine Learning, vol. 8, nos. 3–4, pp. 229–256, 1992.

[15] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimisation,” 2014, arXiv:1412.6980

[16] R. Sambangi, H. Manghnani, S. Chattopadhyay, “LPNet: A DNN based latency prediction technique for application mapping in Network-on-Chip design”, Microprocessors and Microsystems, vol. 87, 104370, 2021

[17] Sutton, R., Bach, F. and Barto, A., 2018. Reinforcement Learning. 2nd ed. Massachusetts: MIT Press Ltd, ch 13, p.321.