GANNoC: A Framework for Automatic Generation of NoC Topologies using Generative Adversarial Networks

Maxime Mirka, Maxime France-Pillois, Gilles Sassatelli, Abdoulaye Gamatié
LIRMM, Université de Montpellier, CNRS
Montpellier, France
<name>.<surname>@lirmm.fr

ABSTRACT

We propose GANNoC, a framework for automatic generation of customized Network-on-Chip (NoC) topologies, which exploits generative adversarial networks (GANs) learning capabilities. We define the problem of NoC generation as a graph generation problem, and train a GAN to produce such graphs. We further present a Reward-WGAN (RWGAN) architecture, based on the Wasserstein GAN (WGAN). It is coupled to a reward network enabling to steer the resulting generative system towards topologies having desired properties. We illustrate this capability through a case study aimed at producing topologies with a specific number of physical connections. After training, the generative network produces unique topologies with a 36% improvement regarding the number of connections, when compared to those found in the training dataset. NoCs' performance assessment is carried out using the Ratatoskr 3D-NoC simulator with state-of-the-art characteristics. Results suggest interesting opportunities in learning correlations between intrinsic NoC features and resulting performance.

KEYWORDS

Generative Adversarial Network, Network-on-Chip, NoC Topology, Neural Networks

1 INTRODUCTION

Complex and heterogeneous Systems-on-Chip (SoCs) typically comprise hundreds of IPs with tight performance demand. This requires powerful Network-on-Chip (NoC) architectures [3], which have become the *de facto* communication infrastructures thanks to their performance scalability. The design of NoCs is a challenging task. Indeed, the characteristics of NoCs must be determined according to the SoC architecture and performance requirements. The performance of a NoC often depends on various factors. Some are levers in the hands of the NoC designers like the network topology, the router architecture and the routing algorithm. Others are application-dependent, e.g. the traffic pattern.

A large part of the existing literature on NoC design [3, 7, 21] has been devoted to the improvement of router architecture and routing algorithms. While NoC topologies are often assumed regular in these studies, here we rather consider both regular and irregular topologies for optimizing the NoC performances, e.g. the average packet delivery latency and the network saturation threshold. Then, we encode the NoC topology exploration issue as a graph generation problem, in which non-trivial graph properties can be identified w.r.t. given performance metrics. This is achieved by exploiting Machine Learning (ML) techniques.

The use of deep learning has grown exponentially in the scientific community, in a vast variety of fields, from basic data classification to medical image interpretation [26]. In particular, generative AI emerged in this decade, with impressive results in building accurate models through unsupervised learning. From this field of deep learning techniques, we distinguish two major architectures: variational autoencoder (VAE) [17] and generative adversarial network (GAN) [13]. The former is typically used to extract features from a set of data. The latter is used to generate new data and explore a dataset space. The GAN architecture has been proven effective in many application fields for generating data having similar characteristics as those of the dataset. Indeed, from the generation of photo-realistic images [16] to medical applications [26], GANs always show impressive learning capabilities.

In this work, we use GANs to generate NoC topologies that contribute in improving NoC performance. Given a traffic pattern and a routing policy as input design constraints, we first rely on a NoC simulator to produce and evaluate different NoC topologies. In particular, we use the Ratatoskr fast and cycle-accurate simulator [15]. The produced topologies make up the dataset based on which our GAN network is trained. Through this process, the network will learn the relevant topological features with a beneficial impact on NoC performance. Ultimately, the GAN is capable of generating novel NoC designs, i.e. not included in the training dataset, with higher performance scores.

The main contributions of this paper are as follows:

- an automated generation of customized NoC topologies (e.g. by targeting some specific router interconnections) through GAN training. For this purpose, we explore two kinds of GANs: the improved Wasserstein GAN (WGAN) presented in [14] and our proposed Reward Wasserstein GAN (RWGAN). The former is an upgraded architecture of GAN, with better convergence compared to conventional GANs [13]. The latter takes into account a reward function based on a specific objective function. To the best of our knowledge, this is the first application of GANs to the NoC design problem;
- a demonstration that both types of GANs can learn and produce novel relevant network topologies, i.e. 100% of the generated topologies do not belong to the training dataset. This opens an interesting perspective for NoC design space exploration (DSE) on top of such trained GANs;
- an illustration of the NoC average latency improvement by 37%, enabled by RWGAN over WGAN. This is obtained by considering the number of connections inside a NoC as the fitness function. Here, the RWGAN-based NoC generation improves the performance by producing NoC topologies where the number of router connections is increased by

36% in average. Note that the increase in the number of connections in itself is not enough, the positions of these connections within the NoCs also play an important role.

The remainder of the paper is organized as follows: Section 2 presents some background considerations on NoCs and GAN networks; Section 3 describes our methodology for generating suitable NoC topologies by leveraging GANs; Section 4 evaluates our proposal through some preliminary experimental results; Section 5 discusses some related work and finally Section 6 gives concluding remarks and indicates further research directions.

2 BACKGROUND NOTIONS

We first recall some basic notions about NoCs. Then, we discuss the NoC topology modeling as compact graph representation. Finally, we give an overview of the GAN network concept.

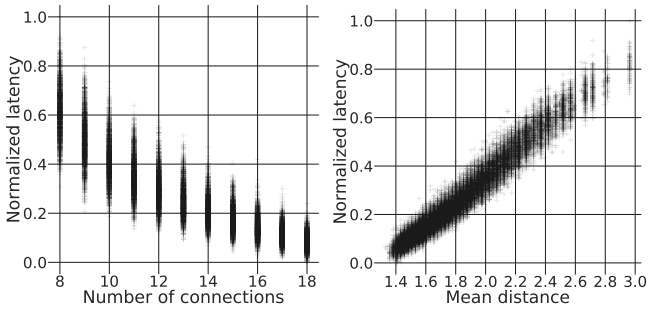
2.1 NoC Features and Performances

We focus our study on the topological attributes of NoCs. These attributes include the number of routers, the number of connections, the number of connections per router (i.e. the routers' degree). Furthermore, NoC performances are evaluated for static routing (see Section 4.1.2) and different traffics. A traffic is characterized by its pattern (e.g. uniform, hotspot) and its injection rate (IR) in percentage of flits per cycle (% flits/cycle). Various metrics can be used to evaluate NoC performances. Here, we consider the latency of a network, generally correlated to the throughput and bandwidth of the network. Hence, it is a relevant metric to evaluate NoC performances.

In Figure 1, several plots are depicted to demonstrate the influence of such attributes on the NoC performance. We present results for a dataset of NoCs with nine routers. They are evaluated with Ratatoskr [15], under both uniform traffic (see Figures 1a and 1b) and hotspot traffic (Figures 1c and 1d). Evaluations are conducted for an injection rate of 10%. Figures 1a and 1c show the latency according to the number of connections of the simulated NoC. In Figures 1b and 1d, are plotted the latency values according to the mean distance between routers. The mean distance of a NoC corresponds to the average number of routers a message has to travel through, before reaching its destination. Naturally, this value is also influenced by the routing method implemented.

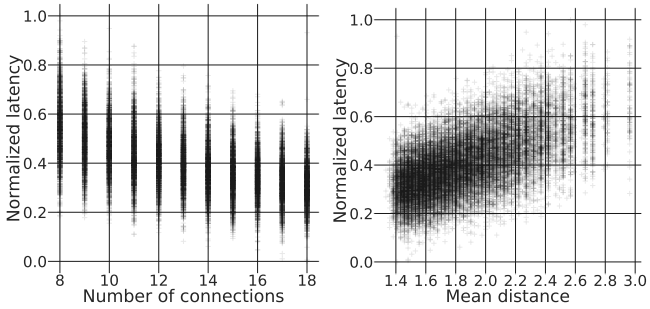
We notice the significant impact of the number of connections and the mean distance over the network performance, under a uniform traffic. Although we could expect similar results (i.e. the more connections, the more bandwidth and the smaller the mean distance, the lesser the latency), the same conclusion cannot be reached when the network is submitted to hotspot traffic. Indeed, despite a similar tendency, under hotspot traffic we notice a rather wide spread in latency, for the same feature value. Hence, we cannot confirm the direct impact of these features on the NoCs latency.

While the previous uniform traffic analysis is quite intuitive, we see that the hotspot case requires further study. Indeed, no obvious causality between the considered NoC topology features and NoC performance can be inferred. Moreover, we can assume that similar conclusions can be expected when dealing with more complex traffic patterns, especially considering the heterogeneous nature of SoCs. Hence, the idea to train a generative model that may learn non-intuitive correlations between intrinsic NoC parameters and performance.



(a) Number of connections impact. Traffic = uniform, IR=10%

(b) Mean distance impact. Traffic = uniform, IR=10%



(c) Number of connections impact. Traffic = hotspot, IR=10%

(d) Mean distance impact. Traffic = hotspot, IR=10%

Figure 1: Nine router NoC performance evaluation.

2.2 NoCs as Graphs

We describe a NoC topology as a set of routers and a list of connections between those routers. While this representation is the most standard form of NoC description, one can then add details to the design description like the routers design or the type of connections (bi-directional, uni-directional, etc.). In our work, we keep the simplest form, as we consider all routers and connections to be of the same type. This enables to reduce the NoC problem to a graph problem by direct analogy, i.e. routers are vertices and connections are edges. Then, as we implement routers with four external connections — typically referred to as North, South, East, West, plus the Local port — we set the maximum degree of our resulting graphs to four.

From this analogy, we can refer to several works on graph generation [10–12, 27]. For most, the chosen representation of a graph is its adjacency matrix. This matrix offers a formal, non-ambiguous, graph representation. Indeed, essential properties such as the number of edges and the vertices' degree can be directly extracted from this representation.

A NoC of n routers will be represented by a nxn adjacency matrix. Hence, the matrix is made of n^2 elements, where each element is a Boolean which encodes an existing connection between two routers. Furthermore, a characteristic of this matrix is that it is symmetric along the diagonal (i.e. top-left to bottom-right). This property results from our decision to consider only bidirectional connections

between routers. It makes it a particularly relevant choice for GAN training, as it is the first pattern the neural network may learn, before converging to more specific details.

2.3 Generative Adversarial Network

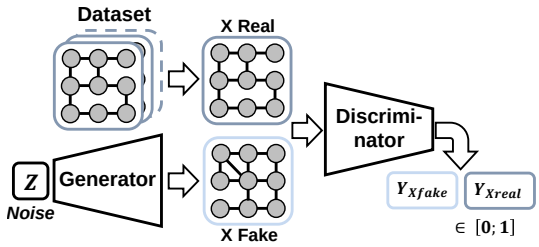


Figure 2: Generative Adversarial Network diagram.

The generative adversarial network is a neural network architecture first proposed in 2014 by Goodfellow et al. [13]. As shown in Figure 2, GANs consist of two main components: a generator and a discriminator. The generator is a generative neural network that learns to create new data-points from a prior dataset space. The discriminator is a discriminative neural network that learns to tell apart samples coming from the dataset and the generator's output. As depicted in Figure 2, the discriminator takes both real and fake data as input (respectively *X Real* and *X Fake*). Real data represent samples from the training dataset, and the fake ones are from the generator output. It outputs a probability value Y, as it behaves as a binary classifier. The closer the Y to 1, the more realistic the input, as seen by the discriminator. In Figure 2 we distinguish two outputs: Y_{real} and Y_{fake}, respectively the discriminator's output from real and fake input.

The two networks are trained simultaneously in two unique fashions. The discriminator follows a supervised learning, where the data from the dataset are labelled as real and the ones from the generator output are labelled as fake. On the other hand, the generator follows an unsupervised learning, where its sole goal is that the discriminator labels its output as real. Generator's input is a random vector z extracted from the latent space Z, called *Noise*. It learns to produce fake samples from it (here, NoC topologies).

To formalize it, let G and D represent the generator model and the discriminator model. Let x denote the real input. Both generator and discriminator have unique objectives, and can be seen as a two players game following the *minimax* rule detailed in Eq. 1 [13]:

$$\min_G \max_D \left(\mathop{\mathbb{E}}_{x \sim p_{data}(x)} [log(D(x))] + \mathop{\mathbb{E}}_{z \sim p_z(z)} [log(1 - D(G(z)))] \right) \quad (1)$$

where $\mathbb{E}[X]$ denotes the expected value of X, p_{data} is the dataset distribution, p_z is the input noise distribution and the notation $y \sim p_y(y)$ means a variable y of the probability distribution p_y. Thus, the generator learns to generate samples to fool the discriminator, and the discriminator learns to differentiate inputs correctly. Along training, this process eventually converges toward a zero-sum game, where each learning improvement of one network leads to a learning deterioration of the second.

Such neural network architecture appears extremely sensitive during the training and achieving convergence is often considered difficult. In [2], Arjovsky et al. propose the use of a Wasserstein loss (from the Wasserstein distance), under the name of Wasserstein GAN (WGAN), to improve the convergence of GAN model. With this model, the discriminator no longer acts as a binary classifier. Indeed, using the Wasserstein loss, the output of this block is no longer comprised in [0, 1] as in the conventional GAN, but within $[-\infty, +\infty]$. This new output can be interpreted as a better measure of "authenticity" or rather "fraudulentness" in the produced data. From this conceptual change, the discriminator network is renamed as the critic network, for a more consistent denomination, and will be denoted by C. Then, an improvement of this method is presented in [14]. The authors present WGAN-GP, where a technique of gradient penalty (GP) is included in the training. The loss function regarding the generator does not change from that of the WGAN, but the critic loss (L_C) is modified as follows:

$$L_C(x_i, G(z_i)) = \underbrace{C(G(z_i)) - C(x_i)}_{\text{original critic loss}} + \underbrace{\alpha(\|\nabla_{\hat{x}_i} C(\hat{x}_i)\|_2 - 1)^2}_{\text{gradient penalty}} \quad (2)$$

where G and C are the generator and critic component, ∇ is the usual gradient operator, x_i and z_i represent a single sample from the dataset and the latent space respectively and $\hat{x}_i = \epsilon x_i + (1-\epsilon)G(z_i)$ with $\epsilon \sim U[0, 1]$ a random number. The α coefficient value is set to 10, as in the original paper [14].

This improved GAN architecture is therefore used throughout our subsequent investigations.

3 THE GANNOC FRAMEWORK

We here describe the GAN network-oriented methodology for generating suitable NoC topologies. The resulting NoC generation framework is referred to as GANNoC.

3.1 Overview

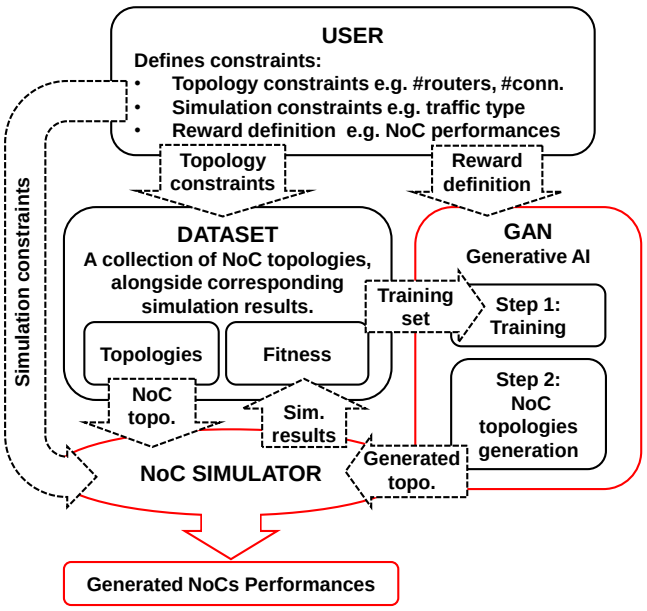


Figure 3: GANNoC framework.

Our framework is built on two essential parts: (1) a neural network that aims to learn how to generate NoC designs and (2) a NoC simulator which is used to evaluate NoC designs. As illustrated in Figure 3, the framework flow starts from user defined constraints. We distinguish three types of constraints. One concerns the NoC topology and is noted "Topology constraints". There are applied when creating the NoC dataset. In our work, we restrict those constraints to the number of routers. The second type of user defined constraints are related to the simulation settings. They define the traffic under which NoCs are evaluated. The third constraint is the definition of the reward. Indeed, with this framework, one could customize a reward to approximate a fitness function that matches NoC properties of interest such as high number of connections or a low latency (further details in 3.2). From the topology and simulation constraints, the dataset is built. To begin with, we create a set of NoC according to the topology constraints, and following the procedure detailed in 4.2. The devised dataset is a collection of NoC topologies combined with a score, also called "fitness". It depends on the metric one would like to optimize (here the number of connections enhancing the NoC average latency). The GAN is finally trained with the created dataset. After training, we leverage the NoC simulator to assess the generated NoCs' performances.

3.2 Implemented GAN

The implemented GAN extends the WGAN principles described in Section 2.3. As illustrated in Figure 4, the GAN architecture we propose consists of three connected nets: a generator G, a critic C and a reward R. The two first are the basic WGAN blocks described before. The reward module aims at assessing given NoC properties.

The reward network R is trained prior to the WGAN training (i.e. G and C together) to approximate a chosen fitness function. Then it is included into the WGAN training as a generator learning guidance. This particular generator learning process is depicted in Figure 4. We call the final architecture Reward-WGAN (RWGAN).

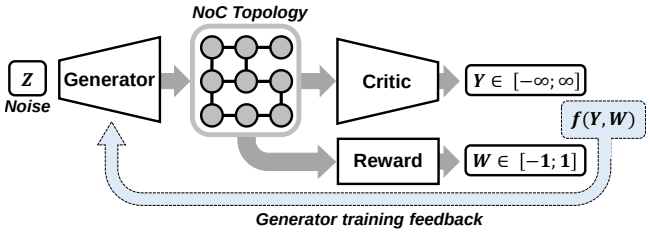


Figure 4: Reward-Wasserstein GAN diagram and its generator loss function $f(Y, W)$.

Generator Network. The architecture of the generator is a classic multi-layer perceptron (MLP). From the obtained empirical results, we found unnecessary to implement a more complex network for this block. The generator takes a sample from a random noise space (Z) and generates a NoC graph as an adjacency matrix. It is trained to create adjacency matrices of NoCs with user-defined characteristics. In this paper, the characteristics will be the number of routers and the bidirectional connections property on the one hand (*nxn* adjacency matrix), and the number of connections on the other hand. The former properties are learned through the basic GAN training. The latter is learned thanks to the inclusion of the reward output into the generator learning. The generator learns to maximize the output of the reward (i.e. W), which corresponds to the fitness value of generated NoCs.

The resulting new generator loss (L_G) is formulated in Eq. 3.

$$L_G(z_i) = (1 - \lambda)L_C(G(z_i)) + \lambda[\beta L_R(G(z_i))] \tag{3}$$

where G denotes the generator, z_i represents a single sample from the latent space, λ is the ratio between the reward loss (L_R) and critic loss (L_C) and β is a weight coefficient to balance both reward and critic losses. Indeed, while the critic loss is in theory not bounded, the reward loss has bounds. Hence, depending on how the critic loss converges, the coefficient β has to be tuned (in this work we set it to 3, but it remains effective within 1 to 5).

Thus, the generator loss is a linear combination of the loss from the critic output and the one from the reward output (i.e. $f(Y, W)$ in Figure 4). To smooth the training from learning general NoC features to specific performances, the linear combination ratio λ is slowly increased from 0 to 0.1 until reaching 10% of reward loss and 90% of critic loss.

Critic Network. The critic loss remains the same as the one in the WGAN-GP (see Eq. 2). The critic network is made of convolutional layers. It can learn patterns from 2-dimensional inputs such as adjacency matrices. We further exploit the rows and columns order of the adjacency matrix to retain router IDs information.

Reward Network. The reward network reproduces the same architecture as the critic network. It is used as a guide through the generator training. Indeed, while the critic network helps global convergence to learn how to generate valid NoC designs, the reward network narrows this learning to induce a characteristic to the generated NoCs. In other words, the generator and critic network flow enables to learn general NoC topology attributes from the training dataset. The generator and reward flow encourages the learning of a specific attribute of interest, corresponding to the fitness function. As mentioned before, the training dataset is made of NoC adjacency matrices alongside fitness values. Note that any arbitrary fitness function can be used for the reward training. We here choose to focus on an easily measurable parameter, i.e. the number of connections, so as to demonstrate the capability of steering the generative process. Other non-obvious cost function like latency, bandwidth or power consumption values can also be retrieved from the simulator and fed as features for training the Reward. From these properties data, the reward network is used to predict a score regarding the fitness of the corresponding generated NoC. By including this information in the generator training feedback loop, we can make it generate NoCs with custom characteristics.

4 EXPERIMENTAL RESULTS

This section evaluates GANNoC. The experimental setup is first described. Then, some NoCs are generated and evaluated.

4.1 The Ratatoskr Simulator

We use Ratatoskr [15] to evaluate the performance of the considered NoCs under specific traffics. Ratatoskr is a flexible, fast and

cycle-accurate NoC simulator, that makes it possible to assess NoC performance within state-of-the-art [5] accuracy. It is an open-source project that allows one to conduct various NoC simulations, with a large scalability regarding input parameters. Among others, users can easily customize the NoC topology and some architecture features (e.g. buffers depth, virtual channels, etc.). Various traffics are already implemented, such as the uniform and hotspot, but pre-recorded traces can also be injected. Then, the injection rate is also scalable, which provides an extensive range of possibilities. Hence the interest in using this simulator, which opens broad perspectives for the diversity of simulation parameters, while ensuring a relatively small simulation time (approx. 5s for a 100k cycles simulation of a 3x3 mesh under a uniform traffic of 10% IR and XY-routing, on an Intel E3-1225 at 3.2 GHz).

4.1.1 NoC parameters. In this section, we list the global NoC parameters used in the Ratatoskr simulation. We only exploit the network performance outputs from Ratatoskr, which do not require to specify the technology used (this information is however mandatory for area and power estimations). Ratatoskr uses wormhole packet switching. Here, we simulate NoCs with 32 flits per packet, 4-flit deep FIFO buffers, a single virtual channel and a 1 GHz clock.

4.1.2 Routing algorithm. To evaluate NoCs, we must set the routing algorithm. As we consider a vast range of NoC topologies, we need to implement a universal routing algorithm within Ratatoskr to enable routing any NoC topologies. Thus, we decide to implement the routing method presented in [25]. It provides an effective static deadlock-free routing methodology. In this work we consider the routing algorithm as a constraint and not as a lever. Thus, this routing method has been chosen for its simplicity of implementation while providing effective universal routing. We take advantage of using a simulator to rely on routing tables. Further investigations may consider more complex routing methods such as table reduction or algorithmic routing techniques.

4.2 Datasets

The dataset we use to train the GAN model is a set of adjacency matrices corresponding to NoC topologies satisfying the following criteria: i) the connections of the NoC form a path between any pair of routers, i.e. a *connected set* of pair-wise router connections; ii) each router r must have less than five neighbour connections within a set C of all connections of the NoC, i.e. $degree(r, C) \leq 4$; and iii) all connections are bidirectional. Algorithm 1 describes a simple procedure to generate an adjacency matrix M of a NoC.

We implement the above algorithm in Python. The choice of designing a homogeneous dataset w.r.t the number of NoC connections comes from the fact that the topology of a NoC is known to be a prime factor on latency, confirmed by the analysis of uniform traffic. Each NoC in the dataset is simulated in Ratatoskr to collect its performance (i.e. latency), and the final dataset consists of a list of NoCs with their number of connections and their mean latency for analytic purpose. Then, a dataset corresponds to a defined traffic.

Algorithm 1: Creation of a NoC adjacency matrix M

Data: nR the number of routers, nC the desired number of connections, $\mathcal{P}$ the set of all possible NoC connections $c = [r_0, r_1]$ where r_0 and r_1 are routers, $MaxTry$ the maximum number of consecutive unsuccessful constructions of a connected set C.
Result: The adjacency matrix M of the NoC to create.

```
1  C ← {} %initially empty set of NoC connections% ;
2  try ← 1 %first try% ;
3  while True do
4      for nC iterations do
5          if ∃c = [r0, r1] ∈ P, such that (degree(r0, C)< 4) and
             (degree(r1, C)< 4) then
6              select c ;
7              add c to C  %this increases the degrees
                 of both r0 and r1 by 1 in the set C% ;
8          else goto line 13  %restart the procedure% ;
9      if C is connected then
10         Create M from the set of connections C ;
11         return M ;
12     else
13         try++ ;
14         if try < MaxTry  then
15             C ← {} ;
16         else return;
```

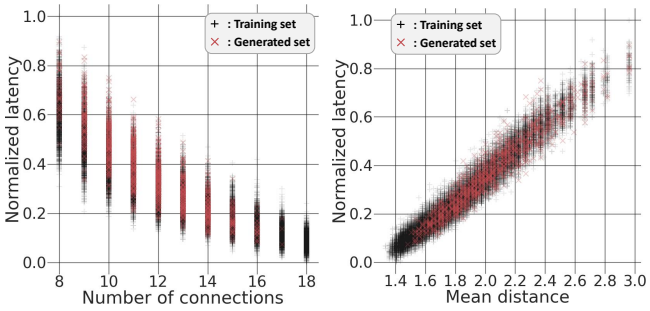
4.3 Results

In this section, we present various results illustrating the performance of our framework to help one to design customized NoC topologies, with specific characteristics.

4.3.1 Training dataset. The training dataset consists of a collection of NoCs of 9 routers. Those NoCs have between 8 and 18 connections, with 10000 unique samples for each class (number of connections). Thus, the dataset is homogeneous w.r.t. this feature. We consider here to analyze NoCs with a uniform traffic of 10% IR.

In this work, we propose to train the reward to evaluate the number of connections of an input topology. Indeed, as we showed in Section 2.1, under uniform traffic, the NoC latency is directly correlated with the number of connections. Hence, the fitness function approximated by the reward is a function that, given an input adjacency matrix, outputs a fitness value corresponding to the number of connections. This fitness value is a normalization of this number.

4.3.2 Neural networks architecture. We build both WGAN and RWGAN from the same blocks. The generator is a 3-layers MLP of $\{162, 162, 81\}$ hidden units respectively, with $tanh$ as activation function. The last layer is eventually reshaped to match the two-dimensionality of the generated matrices (here, 9x9 matrices). It takes a 100-units random array as input. Both critic and reward are made of four layers, using the LeakyReLU activation function with a 0.2 slope for negative values. The first and the second ones are convolutional layers of respectively $\{64, 128\}$ units using $\{9x9, 3x3\}$ filters and strides of $\{1, 2\}$. The two following ones are fully-connected



(a) Comparison w.r.t. the number of connections **(b) Comparison w.r.t. the mean distance between routers**

Figure 5: Comparisons between the original dataset and the WGAN generated samples. Latency values are evaluated for a uniform traffic at 10% injection rate.

layers of $\{512, 1\}$ units. The training uses the RMSprop optimizer with a 5×10^{-5} learning rate, as recommended in [14]. The neural networks are implemented in Python, through Keras API [6] with Tensorflow [1] backend.

4.3.3 WGAN. We first focus on the WGAN training. The global training converges correctly. Indeed, once trained the generator is able to create NoCs with the standard features (number of routers, maximum degree, connected graph) in up to 82% of the cases.

Figure 5 presents a comparison of the training set performances with a set of generated NoCs. All generated NoC topologies are unique, which emphasizes the creativity potential of the generator. We see that latencies of the generated NoCs have a similar distribution as those of the training set, regarding both the number of connections (i.e. Figure 5a) and the routers' mean distance (i.e. Figure 5b). Thus, we can conclude the generator learns effectively the global NoC characteristics from the dataset.

However, we notice the generator has difficulties to produce NoCs topologies with extreme numbers of connections i.e. 8 and 18. This is caused by the learning process on its own. Indeed, during training, the generator will learn to generate data likely to be included in the training set. This makes it learns to converge around the mean of the training dataset space. As this space is homogeneous regarding the number of connections, the mean of this space is around 13 connections. Added to this typical learning behaviour, topologies generated with a high number of connections are more likely to not meet the constraints to have a maximum router degree of four. Hence the lack of 18-connections NoCs. On the other side, a generated topology of eight connections is less likely to be connected. Hence the lack of 8-connections NoCs.

Results: The implemented WGAN is able to learn global NoC topology features. It generates valid topologies at up to 82% of the generations. All of these generated topologies are unique i.e. do not belong to the training dataset.

4.3.4 RWGAN. As noticed before in Section 2.1, under uniform traffic, superior performance directly relates to a high number of connections i.e. densely connected NoCs. Thanks to the reward network, we train the generator to produce topologies with a greater

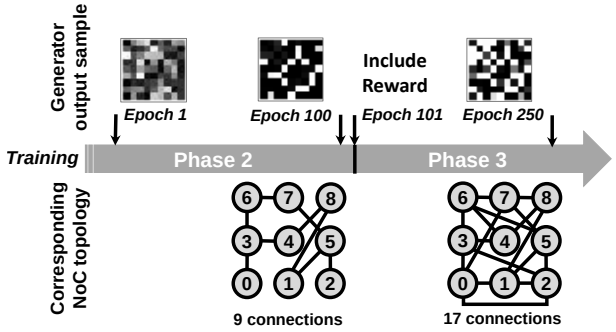


Figure 6: Impact of the reward in the RWGAN training.

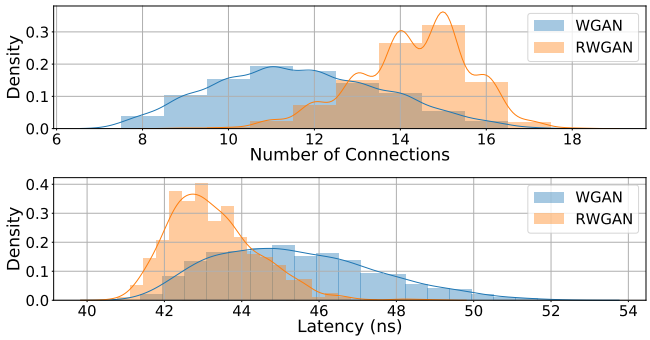


Figure 7: WGAN vs. RWGAN comparison. The distributions of generated NoC topologies according to the number of connections (top) and the packet latency of the NoCs (bottom).

number of connections. Hence, topologies generated by the RWGAN should exhibit better performances than by WGAN.

The experiments of the complete RWGAN architecture consist of three distinct phases: (1) the reward network is trained alone to detect the number of connections in a given NoC topology i.e. adjacency matrix. (2) the RWGAN is trained without the reward (i.e. $\lambda = 0$), like a WGAN training, until it stabilizes. This allows the generator to first learn the basic characteristics of a NoC, through the critic feedback. (3) the reward output is progressively included into the generator training loop (see Eq. 3). It is important to emphasize that the reward network is no longer trained, since step (1). In Figure 6 is illustrated the impact of the reward on the generator training (from phase (2) to (3)). Phase (2) corresponds to the period from epoch 0 to epoch 100, and Phase (3) is from epoch 101 to the end (epoch 250). During this time, the breakdown between the critic feedback and reward feedback to the generator is progressively tailored from 0% and 100% (respectively the reward and critic proportions), to 10% and 90%. Given the same input, the generator learns to increase the number of connections when the reward feedback is included into the RWGAN training.

In Figure 7, we compare the NoC topologies generated by both the WGAN trained alone (i.e. no reward net), and the RWGAN, after a 250 epochs training. We first analyze the number of connections. As expected, the mean number of connections is increased by 36% (from 11 to 15), which represents as well an increase of 36% regarding the min/max possibilities (between 8 and 18). Then,

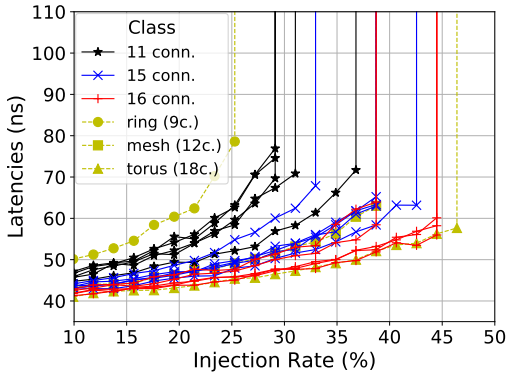


Figure 8: Saturation curves of RWGAN generated NoCs and classic topologies. Generated NoCs are distinguished according to three different numbers of connections: 11, 15 and 16.

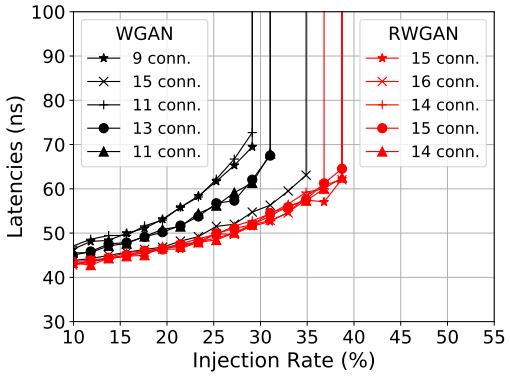


Figure 9: Saturation curves of NoCs generated by both RWGAN (red) and WGAN (black), after a 250-epoch training.

latency distributions are compared in the bottom plot of Figure 7. The mean packet latency is decreased from 45.4ns to 43.3ns. When normalized according to the ensemble of possibilities (from about 40ns to 54ns), the normalized latency is decreased from 0.38 to 0.24, hence an improvement of 37% of the mean NoC packet latency.

We now analyze the saturation plots of the generated NoCs under uniform traffic, regarding packet latency. Those plots are obtained from Ratatoskr simulations, by sweeping through the entire range of injection rates, until the saturation threshold.

First, in Figure 8, we propose to compare the performances of three classes of NoCs. These classes differ in fitness function value, here the number of connections. Hence, we present five different saturation plots of five different generated NoCs, for each class. The three classes are respectively for NoCs of 11 connections (in black), 15 connections (in blue) and 16 connections (in red). To compare with existing regular topologies, we provide network saturation plots of a nine-router ring, mesh and torus (respectively 9, 12 and 18 connections). It can be observed that a NoC with a higher number of connections performs overall better, though a significant overlap exists. Indeed, we observe a 11-connections NoC with a higher saturation threshold than a 15-connections NoC. Same applies to a 15-connections NoC performing better than a 16-connections NoC and the 18-connections torus with similar performance as a 16-connections NoC. This highlights the fact that the number of connections is not the sole parameter impacting performance. It also suggests that a reward network trained to approximate a refined fitness function (e.g. latency) can enable to generate NoCs with optimized performances. For instance, the generator could produce NoCs performing lower latency, for a similar number of connections.

Finally, in Figure 9, we propose to compare saturation results from NoC topologies generated by both the WGAN (i.e. black plots) and the RWGAN (i.e. red plots) after a training of 250 epochs. To obtain these curves, we give to both the WGAN and RWGAN the same 5 inputs. On Figure 9, one type of marker represent one input. The resulting generated topologies are then simulated through Ratatoskr and the results are plotted.

From these results, we see that, despite identical inputs, the generators output NoCs topologies with different performances and number of connections. In particular, the RWGAN outputs NoCs with better performances than those produced by the WGAN. It illustrates the effectiveness of the reward. In addition, we recognize the broad range of performances from the NoCs generated by the WGAN, as the training leads it to mimic the dataset space.

Results: The proposed architecture demonstrates significant improvements of the generated NoCs. While the number of valid generations is reduced by the impact of the reward, the quality in terms of expected performances of the valid generated topologies is increased. Indeed, as the generator learns to increase the number of connections, it degrades its ability to assert the constraint of the maximum degree of 4, and we get up to a 54% probability to obtain a valid NoC (against 82% without including the reward i.e. WGAN alone). However, we obtain a 36% improvement of the NoCs number of connections (i.e. reward cost function). This significant improvement proves the ability of the Reward to speed up the generator learning process to converge toward the generation of NoC topologies with desired characteristics.

5 RELATED WORK

In this work, we consider a GAN network to generate optimized NoC topologies. To do so, we formulated the NoC design problem as a graph structure learning problem. Graph generation with deep learning approach has been investigated in previous work, typically for graph pattern learning. Authors in [10] focus on pattern identification in large graph structures such as social networks. An implementation of a GAN is proposed, based on Long Short-Term Memory (LSTM) networks.

In [27], the authors perform graph generation using an adjacency matrix representation. They define a neural network that learns how to produce the neighbour connections of a given node within a graph. In [12], the authors use Wasserstein GAN to produce labelled graphs. They obtained some promising results regarding the complexity of the generate data. Indeed, their GAN is able to generate not only an adjacency matrix, but also a label matrix. Their work is inspired by the MolGAN framework [4], where labelled graphs are also generated. In MolGAN graphs represent molecules. The

labels feature the type of atoms and bonds. The MolGAN approach presents a GAN architecture with a third network, referred to as Reward network. This third network is implemented to guide the learning generator for converging towards a sub-space of solutions matching the user constraints. The Reward network relies on reinforcement learning (RL), by invoking an external software during the training process. Contrary to MolGAN, our third network does not use RL but rather a CNN that can be trained beforehand with the same dataset used to train the GAN critic module. Therefore, it enables a faster global training as it does not require an external software module (e.g. a NoC simulator) during the training process. Regarding the design of guided GANs, we can also cite the work of Lee and Seok in [18, 19]. They first proposed a controllable GAN [18], inspired by conditional GAN [20], but using a third network as a classifier. They extended this work by adding a fourth network [19], which helps the generator to produce more diversified and high-quality data, i.e. inception score [24].

Regarding NoC design with machine learning (ML), the MLNoC approach has been proposed by Rao et al. in [22]. The authors show that ML can be efficient in predicting general NoC designs such as the type of topology (i.e. mesh, torus, etc.) or the arbitration policy, according to SoCs features. While MLNoC only explores supervised learning techniques, no existing work specifically addresses the NoC topology generation using generative deep neural networks. Finally, in [23], Reza et al. dealt with the problem of designing heterogeneous energy-efficient NoCs by exploring dynamic control solutions through online learning, to adapt NoC configuration at runtime. Our design approach rather operates at design time as it generate physical topologies.

6 CONCLUSION

This paper presents GANNoC, a framework for automatic generation of customized Network-on-Chip (NoC) topologies, using a generative adversarial network (GAN). We propose the RWGAN architecture which is able to generate unique and optimized NoC topologies according to specific performance criteria. In particular, we achieve a 37% improvement of the average generated NoC latency, enabled by RWGAN over WGAN. This is obtained by considering the number of connections inside a NoC as the specific objective function. Here, the generated NoCs performance is improved by producing NoC topologies where the number of router connections is increased by 36% in average. Generated topologies are irregular and may incur place and route or timing closure difficulties for large networks, however our framework makes it possible to control such topological features through the definition of a suitable objective function. GANNoC opens perspectives of building an automatic design space exploration (DSE), where the ability to produce unique NoC topologies under an arbitrary optimization goal is key.

Perspectives: Future work will consist in expending those results to a variety of NoC characteristics. For instance, previous studies on roundabout routers showed the strong impact of their graph topology on the corresponding NoC performances [8, 9]. GANs could be very helpful when used as generators of candidate topologies in this context. Among other, the energy consumption might be the first feature of interest to study next.

REFERENCES

[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. arXiv:stat.ML/1701.07875

[3] L. Benini and G. De Micheli. 2002. Networks on chips: a new SoC paradigm. *Computer* 35, 1 (2002), 70–78. https://doi.org/10.1109/2.976921

[4] Nicola De Cao and Thomas Kipf. 2018. MolGAN: An implicit generative model for small molecular graphs. arXiv:stat.ML/1805.11973

[5] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. 2016. Cycle-Accurate Network on Chip Simulation with Noxim. *ACM Trans. Model. Comput. Simul.* 27, 1, Article 4 (Aug. 2016), 25 pages.

[6] François Chollet et al. 2015. Keras. https://keras.io.

[7] William J. Dally and Brian Towles. 2001. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*. ACM, 684–689.

[8] Charles Effiong, Gilles Sassatelli, and Abdoulaye Gamatié. 2017. Distributed and Dynamic Shared-Buffer Router for High-Performance Interconnect. In *Proc. of the IEEE/ACM Int'l Symp. on Networks-on-Chip, NOCS 2017*. 2:1–2:8.

[9] Charles Effiong, Gilles Sassatelli, and Abdoulaye Gamatié. 2017. Scalable and Power-Efficient Implementation of an Asynchronous Router with Buffer Sharing. In *Euromicro Conference on Digital System Design, DSD*. 171–178.

[10] Aleksandar Bojchevski et al. 2018. NetGAN: Generating Graphs via Random Walks *(Proceedings of Machine Learning Research)*, Vol. 80. PMLR, 610–619.

[11] Tinghao Guo et al. 2019. Circuit synthesis using generative adversarial networks (Gans). In *AIAA Scitech 2019 Forum*. https://doi.org/10.2514/6.2019-2350

[12] Shuangfei Fan and Bert Huang. 2019. Labeled Graph Generative Adversarial Networks. arXiv:cs.LG/1906.03220

[13] Ian J. Goodfellow et al. 2014. Generative Adversarial Networks. arXiv:stat.ML/1406.2661

[14] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved Training of Wasserstein GANs. arXiv:cs.LG/1704.00028

[15] Jan Moritz Joseph et al. 2020. Ratatoskr: An open-source framework for in-depth power, performance and area analysis in 3D NoCs. arXiv:cs.AR/1912.05670

[16] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *CoRR* abs/1710.10196 (2017). arXiv:1710.10196 http://arxiv.org/abs/1710.10196

[17] Diederik P. Kingma and Max Welling. 2019. An Introduction to Variational Autoencoders. *Foundations and Trends® in Machine Learning* 12, 4 (2019), 307–392.

[18] Minhyeok Lee and Junhee Seok. 2019. Controllable Generative Adversarial Network. arXiv:cs.LG/1708.00598

[19] Minhyeok Lee and Junhee Seok. 2020. Score-Guided Generative Adversarial Networks. arXiv:cs.LG/2004.04396

[20] Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. arXiv:cs.LG/1411.1784

[21] E. Ofori-Attah and M. O. Agyeman. 2017. A survey of recent contributions on low power NoC architectures. In *2017 Computing Conference*. 1086–1090.

[22] N. Rao, A. Ramachandran, and A. Shah. 2018. MLNoC: A Machine Learning Based Approach to NoC Design. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. 1–8.

[23] M. F. Reza, T. T. Le, B. De, M. Bayoumi, and D. Zhao. 2018. Neuro-NoC: Energy Optimization in Heterogeneous Many-Core NoC using Neural Networks in Dark Silicon Era. In *2018 IEEE Int'l Symp. on Circuits and Systems (ISCAS)*. 1–5.

[24] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved Techniques for Training GANs. arXiv:cs.LG/1606.03498

[25] José Carlos Sancho, Antonio Robles, and José Duato. 2000. A New Methodology to Compute Deadlock-Free Routing Tables for Irregular Networks. In *Network-Based Parallel Computing. Communication, Architecture, and Applications*, Babak Falsafi and Mario Lauria (Eds.). Springer Berlin Heidelberg, 45–60.

[26] Nripendra Kumar Singh and Khalid Raza. 2020. Medical Image Generation using Generative Adversarial Networks. arXiv:eess.IV/2005.10687

[27] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. arXiv:cs.LG/1802.08773