Memories - Materials, Devices, Circuits and Systems 4 (2023) 100059

Contents lists available at ScienceDirect

Memories - Materials, Devices, Circuits and Systems

journal homepage: www.elsevier.com/locate/memori

A performance-centric ML-based multi-application mapping technique for regular Network-on-Chip

Jitesh Choudhary [a,b,*], Chitrapu Sai Sudarsan [a], Soumya J. [a]

[a] BITS-Pilani, Hyderabad Campus, Hyderabad, India
[b] Centre for Development of Advanced Computing, India

ARTICLE INFO

Keywords:
Fault-tolerant
Neural network
Network-on-Chip
Machine Learning
Multi-application mapping

ABSTRACT

This research article discusses the challenges faced by the Network-on-Chip (NoC) architecture due to increased integration density and proposes a novel fault-tolerant multi-application mapping approach called "FANC." The approach is based on Machine Learning (ML) and can provide solutions for unseen graphs and topologies without prior training. The proposed technique uses an ML-based model to extract relevant information from the search data and incorporate it into the search process. This results in a more robust model with a higher convergence rate and solution quality. The approach is evaluated using a variety of simulation parameters, such as communication cost, network latency, throughput, and power usage. Static simulations are performed in a Python programming environment, while dynamic simulations are performed with a SystemC-based cycle-accurate NoC simulator and the Orion2.0 Power tool. The results show that FANC reduces communication costs by 266%. It also improves network latency by 9%, throughput by 1%, and power consumption by 7%. The approach also simplifies and minimizes the search area in the design exploration process and can be used as an auxiliary component for other optimization algorithms.

1. Introduction

System-on-Chip (SoC) has become ubiquitous in modern electronics, powering everything from smartphones and tablets to smart homes and autonomous vehicles. SoC is an integrated circuit that incorporates all essential processing components onto a single chip, providing a complete system solution [1]. SoC designs typically include a processor unit, input/output ports, internal memory, and input/output blocks, among other components. SoCs are programmable and integrate multiple applications on the same chip. These multi-application systems share many hardware components and have varying communication needs and design constraints [2]. As the number of cores on the chip increases, communication between them becomes increasingly crucial for system performance. Traditional communication structures which cannot meet today's applications' concurrent communication, and high-throughput demands [3] lead to a communication-centric paradigm known as Network-on-Chip (NoC).

An NoC represents the communication framework of an SoC, utilizing router-based communication between its Processing Elements (PEs). A typical NoC architecture is shown in Fig. 1. The essential components of an NoC include PEs, Routers (R), and Network Interfaces (NI). NoC can be designed using global parameters, such as topology, network size, mapping, and local parameters, such as link width, buffer configuration, flow control, routing technique, and arbitration mechanism. The NoC's configuration significantly impacts the cost and performance of the entire SoC [4]. Determining the optimal global solution is a challenge, as the ultimate structure of the NoC must satisfy the demands of all SoC applications. Designing the NoC for a specific application may not adequately address the needs of other applications [4].

Application mapping is an important stage in the design of NoC-based SoCs. During this stage, NoC routers are assigned to each core of the application graph [3]. Application mapping is a quadratic assignment problem, and as the network size grows, the search space grows exponentially. For instance, if an application has 32 cores that can be mapped onto 32 routers in the network topology, the search space consists of 32! possible solutions, equivalent to 2.63×10^{35}. If it takes 0.001 ms to obtain one solution, exploring all possible solutions would take 8.33×10^{12} years. Therefore, many researchers proposed different application mapping techniques to reduce design time and enhance NoC performance. It is also observed that an optimal application mapping solution can improve NoC performance by up to 60% [4].

As shown in Fig. 2, existing application mapping techniques can be categorized as dynamic or static based on the time at which the application mapping takes place [6]. In static mapping, application mapping is done during the system design phase and remains fixed throughout the execution. This minimizes computational overhead and energy loss, as the application mapping is completed before execution.

* Corresponding author at: BITS-Pilani, Hyderabad Campus, Hyderabad, India.
E-mail address: jiteshchoudhary@gmail.com (J. Choudhary).

https://doi.org/10.1016/j.memori.2023.100059
Received 30 November 2022; Received in revised form 13 May 2023; Accepted 13 May 2023

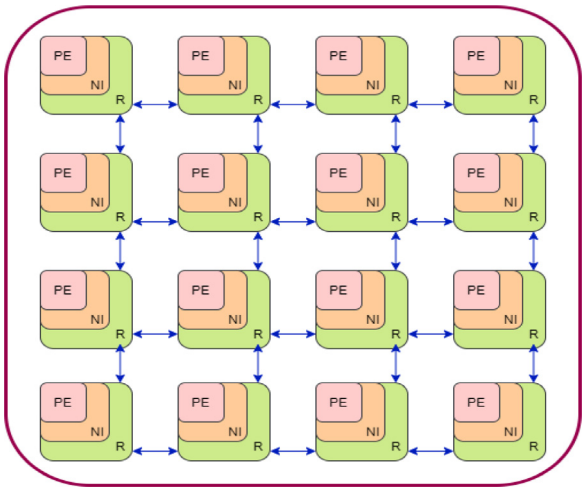

Fig. 1. NoC architecture [5].

On the contrary, dynamic mapping assigns cores to routers during run-time. The approach proposed in this article is a static application mapping approach. The choice of application mapping technique and network topology in an NoC design can impact several performance parameters, such as communication cost, network latency, low power consumption, high efficiency, and design throughput.

Every NoC topology has unique characteristics, such as the different numbers of global and local ports, bisection width, routing strategies, etc., and the solution to the same application mapping problem must be treated differently if the topology is different [7]. Due to this, it is necessary to select the right NoC topology for application mapping. Both regular and irregular topologies can be used in the NoC design. Regular topologies, such as mesh, torus, and spidergon (see Fig. 3), are more straightforward in structure, easier to implement, and reusable for various applications than irregular topologies.

The application mapping techniques in the literature mainly employ mathematical formulations, heuristics, and metaheuristic approaches. Although mathematical programming is the most accurate method, it becomes increasingly time-consuming as the network size increases. On the other hand, the heuristic approach executes in the shortest amount of time but is prone to get trapped in local minima. The metaheuristic method has limited space exploration, slow convergence, sensitivity to starting conditions, trouble dealing with constraints, no theoretical guarantees, and limited scalability for large problems. These limitations might lead to inferior solutions in time-critical or multi-constraint optimization problems. The application mapping process produces substantial data during the search process. These data contain characteristics of good and bad solutions, the effectiveness of various operators at various stages, the precedence of search operators, etc. Incorporating this knowledge can enhance the search process, solution quality, convergence rate, and robustness. Based on the above discussion, we investigated fault-tolerant multi-application mapping using Machine Learning (ML) techniques to overcome the limitations of current approaches and seek novel solutions.

Current SoC fabrication technologies make the NoC more susceptible to transient, intermittent, and permanent faults, such as crosstalk, electromagnetic interference, alpha-neutron particle strikes, and power supply disturbances. These faults are dealt with using fault-tolerant techniques that can be applied at various abstraction levels, from circuit to system [9]. Assuming that the most communicative core is highly susceptible to failure, we focus on permanent application core failure. Most approaches proposed in the literature consider the same assumption for core failure and use additional cores to improve system reliability in the event of core failure [9]. This article continued our earlier work on multi-application mapping on mesh NoC using Artificial Neural Networks. We extended the scope to fault-tolerant multi-application NoC mapping. The following are novel contributions of our work in this article:

- To investigate fault-tolerant multi-application mapping approaches and develop ML-based multi-application mapping approach for regular NoC topologies.
- To find how different design parameters, such as core failures (single and multiple), the number of cores in applications, and the size and type of NoC topology affect the performance of the proposed approach by evaluating communication cost, latency, throughput, and power consumption.

This manuscript presents related work in Section 2. Section 3 discusses some relevant concepts on the topic. Section 4 presents the problem formulation, followed by the method proposed in Section 5. The experimental results and their analysis are discussed in Section 6. Section 7 concludes the proposed work.

2. Related works

With the prevalence of on-chip networks, the problem of application mapping has become a critical issue. Numerous mapping strategies have been proposed to optimize performance by improving communication cost, energy consumption, chip area, etc. When mapping multiple applications, it is essential to ensure that mapping one application does not compromise the performance of others. Considerable research efforts have been undertaken to address the multi-application mapping problem, but they are primarily for fault-free application mapping applicable to a particular NoC topology. Very little focus has been given to fault-tolerant multi-application mapping for regular NoCs. A summary of the different approaches, along with their advantages and disadvantages, is presented in Table 1.

P. Veda Bhanu et al. proposed a two-step fault-tolerant approach to the multi-application mapping problem in [1]. In the first phase, they employ a particle swarm optimization-based approach to map fault-tolerant applications to a reconfigurable mesh-based architecture by adding spare cores. In the second step, they propose a reconfiguration heuristic that is used to reduce communication costs. Sepúlveda et al. presented a multi-objective adaptive immune technique that considers multiple latency conditions for multi-application mapping in [4]. They utilized the same platform for different applications at different times to reduce latency and power consumption. However, the main limitations of these systems were the time-consuming reconfiguration of NoC, the loading of new applications, and the exclusion of several applications from execution simultaneously.

In [10], Yang et al. proposed a multi-application mapping mechanism where each application is defined as an ideal mapping area on a fixed platform. However, this approach maps several applications sequentially, which limits the mapping options for later applications. In [11], Khalili et al. proposed a fault-tolerant multi-application mapping technique. This technique aims to locate the optimal locations for spare processing cores and map an application to free, non-faulty cores. The approach comprises three phases. First, they create a new core graph using a spare core from an existing application core graph; then, using a heuristic method, they place the application in the smallest rectangular area possible; and finally, during the search phase, they select a mapping area with the overall best performance.

In [12], Zhu et al. proposed a heuristic-based technique to balance packet latency and performance in the multi-application mapping scenario. In [13], Khasanov et al. presented an approach for mapping applications in heterogeneous multi-core systems, primarily focusing on scheduling threads or jobs within each application. They proposed software-based task migration for managing core failures during run-time. Lee et al. in [14] described an approach that examines all possible processor failure scenarios at compilation time and stores the resulting

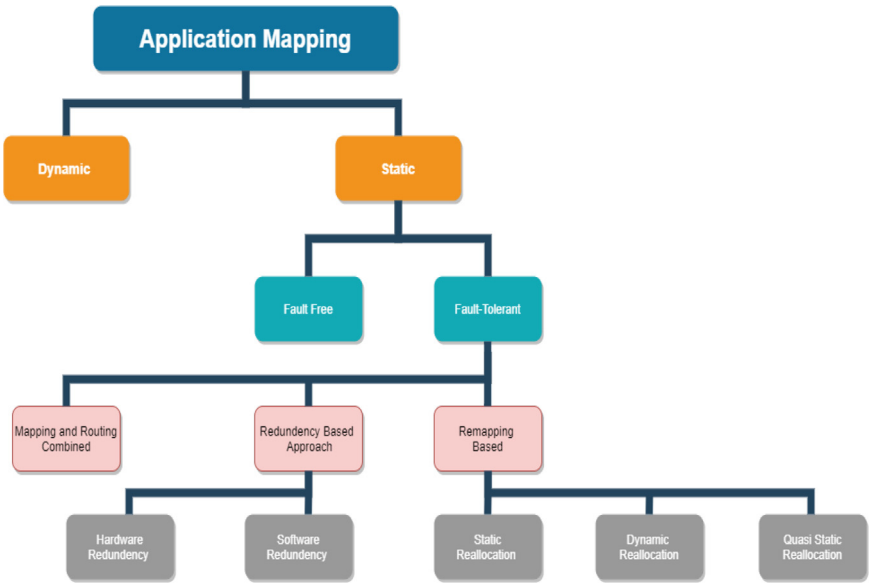

Fig. 2. Application mapping classification [8].

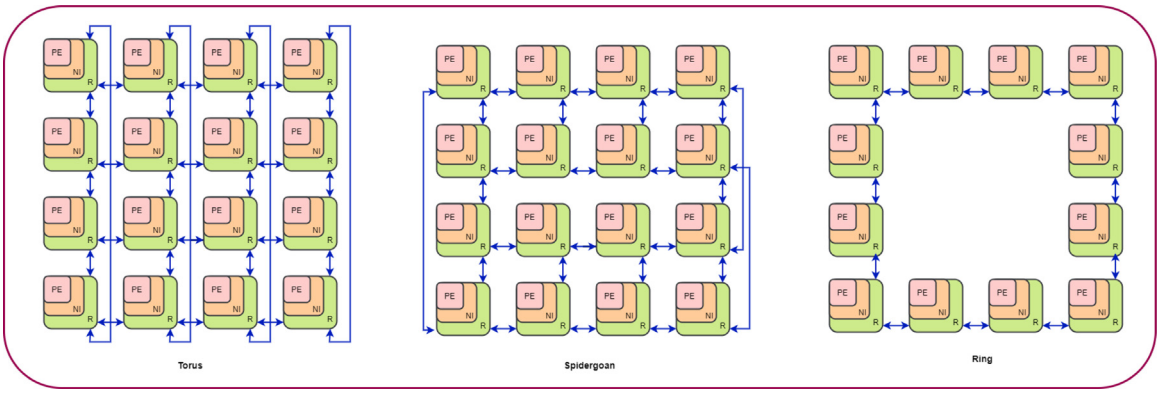

Fig. 3. NoC topologies [5].

remapping data to tolerate processor failures during runtime. However, this method limits the system's capacity to scale.

Chou et al. proposed a fault-aware resource management strategy for NoC-based multiprocessors in [15]. Instead of changing for each application, this technique keeps extra cores in the same place. Failure recovery through this approach results in considerable communication energy and performance overhead. Modarressi et al. in [16] introduced a reconfigurable NoC architecture that allows topology alteration via topology switches. They proposed an NoC architecture that tailors its topology based on the traffic of the input application. In [17], Ritesh et al. proposed fault diagnosis and reconfiguration for multi-application mapping. The reconfiguration would be done using software-based and hardware-based approaches and compared in terms of complexity.

Trivio et al. in [28] divided the NoC network into several partitions for specific applications' mapping and traffic flow. The dynamic allocation of resources enables the reconfiguration of the partitions based on the demands of the applications. Soumya J et al. described a reconfigurable NoC based on mesh topology in [18], which utilizes multiplexers to reconfigure the connections between the cores and the routers. First, the cores of the combined application are provisionally assigned to individual routers, and then unique core locations are determined for each application.

Zhu et al. presented a platform-aware design methodology for multi-application mapping on NoCs in [19]. Their proposed technique incorporates single-application heuristics to provide accurate mapping for multi-applications. Farias et al. suggested a metaheuristic-based technique to map programs to an NoC-based multiprocessor architecture in [20]. Soumya J. et al. suggested a reconfigurable architecture using configuration switches in [21]. They created reconfiguration algorithms and mapping based on integer linear programming and Particle Swarm Optimization (PSO) for application mapping. In [22], Ge et al. described a multiphase, multi-application mapping approach for NoC design.

All the research mentioned above-used heuristic, metaheuristic, nature-inspired, or genetic techniques to generate optimal solutions for multi-application mapping. However, multiple researchers utilized ML in the last few years to solve combinatorial optimization, achieving excellent accuracy at a lower computing cost than heuristics. Qingkun et al. in [23] proposed NMA algorithm using a pointer network and Reinforcement Learning (RL) to map the IP core, and in [24], they proposed MPN-GA and MPN-PSMAP algorithms, which are a combination of neural networks, pointer networks, heuristics, and RL. In [25], Jitesh et al. present a technique inspired by RL to map applications onto the NoC mesh. Samala et al. in [26] proposed an RL-MAP method based on actor–critic RL for optimized application mapping.

The ML-based approaches discussed here demonstrated the potential of ML to solve application mapping problems, but they are limited to fault-free multi-application mapping scenarios. This article proposes ML-based application mapping techniques to handle permanent core faults in multi-applications systems.

Table 1
Comparison of different approaches.

Reference	Author	Approach	Objective	Advantage	Disadvantage
[1]	P. VedaBhanu et al.	Heuristic and Reconfiguration	Fault-tolerant multi-application mapping with minimal communication cost	Demonstrated that reconfiguration reduces communication cost	Two-step process increases execution time
[4]	Sepúlvedaet al.	Metaheuristic	Multi-objective multi-application mapping with latency and power restrictions	Reduced latency and power consumption	Loading applications sequentially delays the execution process
[10]	Yanget al.	Heuristic	Multi-application mapping with minimal application area on a fixed platform	Each application is defined as an ideal mapping area on the platform.	Sequential mapping limits mapping options for later applications.
[11]	Khaliliet al.	Heuristic	Fault-tolerant multi-application mapping with optimal locations for spare cores	Reduced communication energy	High execution time
[12]	Zhuet al.	Hungarian based Heuristic technique	Multi-application mapping with packet latency balancing	Performance awareness and packet latency balancing	Stuck in a local optimum and may not find the optimal solution
[13]	Khasanovet al.	Metaheuristic	Multi-core application mapping for thread/job scheduling for heterogeneous application	Software-based task migration. Runtime core failure management	Not suitable for all types of applications High execution time
[14]	Leeet al.	Heuristic	Multi-application mapping with minimal throughput degradation and task migration overhead	Examines all possible processor failure scenarios at compilation time	Limited Scalability
[15]	Chouet al.	Heuristic	Multi-processor mapping with minimal communication energy	Fault-aware resource management Scalable	High-Performance overhead Location of spare cores is fixed
[16]	Modarressiet al.	Heuristic and Reconfiguration	Multi-application mapping with tailored topology based on input application traffic	Dynamic reconfiguration of inter-router connections. Topology alteration based on the input application	Additional switches are required for topology alteration.
[17]	Riteshet al.	Heuristic and Reconfiguration	Fault diagnosis and reconfiguration based on routing for multi-application mapping	Software and hardware-based reconfiguration. Reconfiguration reduce area overhead	High timing complexity
[18]	Triviñoet al.	Reconfiguration	Multi-application mapping using resources virtualization Reconfiguration	Dynamic allocation of resources Reconfigurable reduces execution time and latency	Fixed partitioned for each application and traffic flow.
[19]	Soumya Jet al.	Mathematical, Heuristic and Reconfiguration	Multi-application mapping with minimal communication cost	Exact mapping, Flexible reconfigurable architecture	High execution time. Multiplexer add to complexity and delay
[20]	Zhuet al.	Platform-aware heuristics	Multi-applications mapping to achieve design targets.	Fast and accurate mapping for multi-applications.	Framework serves only as a guideline.
[21]	Fariaset al.	Metaheuristic	Multi-task multi-application to reduce energy consumption	Less number of interactions during mapping reduced the time	Require more computational resources May take longer to find solution
[22]	Soumya J.et al.	Mathematical, Heuristic and Reconfiguration	Multi-application mapping with minimal communication cost	Reconfigurable architecture Scalable and efficient	Take more time to find the solution Complex to implement
[23]	Geet al.	Metaheuristic	Multiphase, multi-application with minimized application mapping area	Efficient and scalable	Long execution time for larger applications Complex to implement

(*continued on next page*)

Table 1 (*continued*).

Reference	Author	Approach	Objective	Advantage	Disadvantage
[24]	Qingkunet al.	Heuristic and ML	Application mapping with minimal communication cost	Fast and Lower computing cost than heuristics	Need significant data to train the ML model
[25]	Qingkunet al.	Metaheuristic, ML and RL	Application mapping with minimal communication cost	Accurate and Lower computing cost than heuristics	More complex than traditional heuristics and metaheuristics
[26]	Jiteshet al.	Heuristic and RL	Application mapping with minimal communication cost	Lower computing cost than heuristics	Execution time increases with application size
[27]	Samalaet al.	Heuristic and RL	Application mapping with minimal communication cost	More efficient than traditional heuristics and metaheuristics	Need significant data to train the RL model

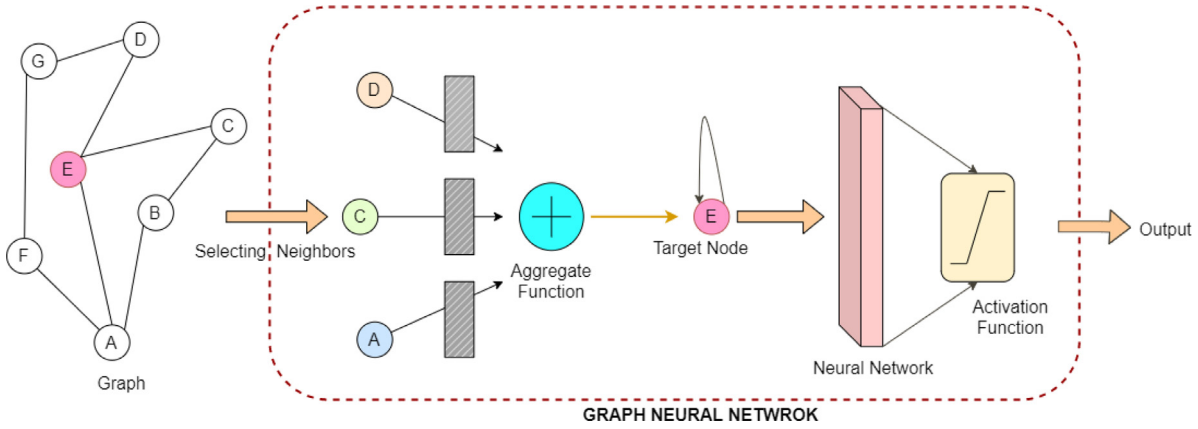

Fig. 4. GNN pipeline [27].

3. Preliminaries

This section presents the basic definitions and concepts of two essential topics involved in the proposed work: the Graph Neural Network (GNN) and RL.

3.1. Graph neural network

A GNN is a specific type of neural network that can be trained to graph-type input data for representation learning. GNNs perform direct graph operations, using information diffusion and neural networks to execute their transition and output functions. The nodes in GNNs share information with their neighbors, modifying their states and sending messages until they reach stability. The transition function determines the node's new state based on various factors or features, such as edge information, states of neighboring nodes, and node characteristics. GNNs are adaptable to input graphs and can aggregate data across vertices to reflect system connections. This allows for predicting and extrapolating properties for nodes, links, and their connections in new graphs. A typical GNN pipeline consists of the following three functions shown in Fig. 4:

- A message-passing function that enables the exchange of information between nodes across edges.
- An aggregation function that combines received messages into a single, fixed-length representation.
- An activation function that generates node-level representations based on the previous layer representation and aggregated information.

GNNs contain layers that disseminate node information, and their count determines the significance of remote node interactions in an application. Sampling and pooling are the functions of GNN that make it easier to evaluate dynamic graphs. Sampling cuts down on the area around each node to balance total resources and run time. Whereas pooling reduces the number of nodes that need to be brought together. GNNs can be used for various prediction applications, including node classification, link prediction, community detection, network similarity, and graph classification.

3.2. Reinforcement learning

RL is a branch of machine learning involving an agent's ability to receive rewards or penalties based on performance. An RL agent can perceive and comprehend its surroundings, execute actions, and learn from its experiences to identify optimal behavior. The agent learns to make informed decisions by interacting with the environment and analyzing its responses. RL employs the Markov Decision Process (MDP) framework to make decisions, which can be categorized as model-free and model-based learning. In model-based learning, the best action is determined by an environment-based prediction model. On the other hand, model-free methods learn control strategies through trial-and-error algorithms that explore the environment and determine the best policy. Fig. 5 shows a simple RL agent interacting with its environment E over several discrete time steps. At each time step t, the agent attains a state S_r from the state space S and chooses an action based on its policy P, which maps the state S_r to the action A. The agent moves to the next state S_i, and a scalar reward $R_t : SxAxS \leftarrow R$ is awarded. This process continues until the agent reaches the required state. In value-controlled model-free learning, the right policy can be found by calculating the value function for each state. Q-learning [29] is an example of a value-based method that approximates Q(s, a) through function approximations. At each state S, the agent chooses the optimum action to maximize its reward.

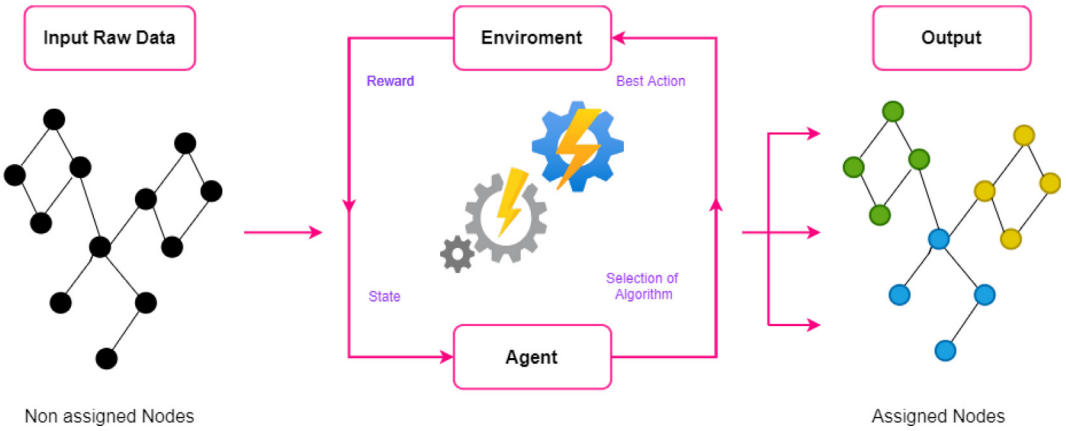

Fig. 5. Reinforcement learning model [30].

4. Problem formulation

Application Core Graphs (ACGs) illustrate the connection between different cores along with their communication bandwidth, and an NoC Topology Graph (NTG) shows the connection between routers in an NoC topology. The proposed work assumes task-to-core mapping is completed, application cores are homogeneous, the most communicative core has the highest risk of failure, and XY routing determines the route and hop count for data transfer between the source and destination routers. These assumptions are common to most application mapping approaches described in the literature. In the event of a core failure in an application, spare cores provide reliability without affecting the system's performance. Mapping a single application aims to find the best location for each application core based on specific performance parameters. However, in multi-application environments, the difficulty is exacerbated by the need to choose the appropriate placement for each application. This work proposes a GNN-based fault-tolerant application mapping method onto regular topology to minimize communication costs. To formulate the fault-tolerant mapping problem, we require the following definitions:

4.1. Definition 1: Combined application core graph

The weighted directed graph ACG represents an application communication graph ACG(C, E) where:

- # C is a set of nodes representing the cores of an application, $c_i \in C : i = (1, 2, 3, \dots, N_c)$;
- # E is a set of weighted directed edges, representing the communication bandwidth between two adjacent cores c_i and c_j, $e_{ij} \epsilon E : i, j = (1, 2, 3, \dots, N_e)$;
- # The weight B_{ij} of the edge e_{ij} between the cores c_i and c_j represents the shared data represented as the number of bits transferred per second between them.

The combined application core graph is a set $S = (ACG_1, ACG_2, ACG_3, \dots ....ACG_n)$ of applications, where n is the number of given applications in a multi-application environment. An edge e is included in the combined application core graph only if $e \in E$ is an edge for at least one application graph ACG_i. As demonstrated in Eq. (1), the total weight of an edge e equals the sum of the weights of all such edges in the entire set of applications [21].

$$Weight\,(e) = \sum_{i=1}^{n} (Weight\,of\,e\,in\,E_i) \tag{1}$$

Creating a combined application core graph identifies cores with high communication bandwidth and places them as close as possible during application mapping.

4.2. Definition 2: NoC topology graph

The NoC platform considered in this article for application mapping is a regular 2D architecture comprising routers R and physical connections L. In regular topologies, every router is connected to every other router. An NoC platform is represented as a directed graph called NTG(R, L), where:

- # R is a set of nodes, representing routers with processing elements in the NoC architecture, $r_i \in R : i = (1, 2, 3, \dots, N_R)$;
- # L is a set of links, representing the routing path between two adjacent routers r_i and r_j, $l_{ij} \in L : i, j = (1, 2, 3, \dots, N_L)$;
- # D_{ij} represent the communication distance from node r_i to node r_j and defined by Eq. (2), where (x_i, y_i) and (x_j, y_j) represent the coordinates of the router r_i and r_j, respectively.

$$D_{ij} = |\, x_i - x_j \,| + |\, y_i - y_j \,| \tag{2}$$

Routing Mechanism: The routing mechanism determines the shortest route between the source and destination routers in the NoC architecture. In regular NoCs, the XY routing methodology is a well-known data transport technique. In XY routing, a packet must always be routed horizontally along the X-axis until it reaches the same column as the destination. Then move upward or downward along the Y-axis. The communication distance D_{ij} in the XY routing is calculated using Eq. (2) [25].

4.3. Definition 3: Problem definition

The multi-application mapping problem involves finding an NoC mapping zone for each ACG where all cores can be arranged for optimal performance. Hence, all applications are placed in ideal mapping zones, saving space. The mapping function described by Eq. (3) assigns each core of ACG to an appropriate NTG router. The combined application core graph must have fewer cores than NTG's routers for correct mapping, and each core must be assigned to a single router.

a. The mapping problem can be formulated as follows:

$$\forall c_i \in C, map(c_i) = (x, y) \tag{3}$$

where (x, y) indicates a free and non-failed core that is located in the row xth and the column yth of the NoC topology.

4.3.1. Constraints

While forming a path, a node in the proposed architecture can be a source, a destination, or an intermediary. A source node has an out-degree of 1, whereas a target node has an in-degree of 1. Any route node, except for the source and destination, has zero degrees. In the

proposed work, all routers and the application core must satisfy the following conditions:

a. Each core should be assigned to exactly one router.

$$\forall c_i \neq c_j \in C, map(c_i) \neq map(c_j) \tag{4}$$

b. Each router can have at most one core mapped on it.

$$\forall r_s \in R, \sum_{c_i \in C} m_{c_i}^{r_s} = 1 \tag{5}$$

4.3.2. Objective function

The objective of the proposed mapping technique in the multi-application scenario is to reduce the communication cost for each application. Within each application, the communication cost is defined as the product of the maximum bits communicated per second (bandwidth B) between the source core c_i and the destination core c_j on the NoC topology and communication distance D_{ij}. The communication distance is the number of hops between the source and destination routers in the NoC topology. The communication cost is calculated using Eq. (6), which suggests that we must limit the distance between the cores of each application to reduce communication costs. In the proposed work, the communication cost of the combined applications graph is the sum of the communication costs of all the applications in the NoC. The bandwidth (edge weight) B_{ij} and communication cost are zero when c_i and c_j are not linked. Any application can reduce latency and power consumption by reducing the communication costs of its mapping onto NoC [31].

$$\begin{aligned} Comm.Cost &= \sum_{i=1}^{n} (No.\,of\ Hops * Bandwidth) \\ &= (\sum_{i=1}^{n}\sum_{i=1}^{n} D_{ij} * L_{ij}) * (\sum_{k=1}^{n}\sum_{m=1}^{n} E_{km} * B_{km}) \end{aligned} \tag{6}$$

5. Proposed methodology

This section discusses the methodology of the proposed FANC: Fault-Tolerant Multi-Application onto Regular NoC, and the pseudocode for the same is presented in Algorithm 1. FANC is based on the GNN-RL model to efficiently find the optimal mapping for application onto regular NoC topologies.

5.1. Design flow

The design flow of the proposed method is presented in Fig. 6, and can be broken down into three stages, which are as follows:

- Data Preparation: Construction of the Combined Core Graph (CCG)
- Mapping Search: Mapping of cores in the CCG to the NoC using the GNN-RL Model.
- Optimization: Optimization for each application

The FANC algorithm is a process that aims to optimize the placement of the core on an NoC topology. To begin with, it combines the Application Communication Graphs (ACGs) into a single graph, representing the communication pattern of the application. Once the ACGs are combined, the algorithm injects a fault into the model, assuming that the most communicating core is faulty, and adds a spare core to the combined application graph. After this, FANC computes the bandwidth and distance matrices representing the communication and physical distances between each core and router.

FANC assumes an ideal condition to calculate the ideal cost in which all ACG cores are mapped onto NoC routers with one hop distance. The algorithm then generates a random placement of cores on the routers, which is used as initial input for the GNN-RL model.FANC employs a GNN-RL framework, where the agent learns from the environment's feedback to improve its decision-making process. The algorithm trains a model to predict the optimal mapping and observes the environment to receive feedback on the current mapping. The algorithm repeats the above process for a specified number of iterations. Finally, the GNN-RL model returns the optimal mapping as a solution. In the end, the router-core pair in optimal mapping swap their position to further check for possible improvement in the solution.

5.1.1. Data preparation

All input ACGs are combined to create a single multi-application communication graph at this stage. The most communicated core is injected as a faulty core, and a spare core is added. A random mapping is also created to feed as the initial input to the mapping search stage.

5.1.2. Mapping search

In the proposed method, only the ACG and NTG data are fed to the GNN-RL model without any intended output data. The self-organization or adaptive learning of the model determines the qualities of the input data needed to achieve the desired result without any explicit programming. The model trains on initial mapping data and builds a probability matrix for router core pairings using hidden-layer weights and functions. The probability matrix predicts the optimal mapping, and the communication cost is calculated based on the optimal mapping. After each iteration, the model adjusts its weights according to the calculated loss, which is the difference between the present and ideal communication costs. The detailed architecture of the proposed GNN-RL model for application mapping is presented in the following sections.

5.1.3. Optimization

The loss function in the GNN-RL model frequently encounters saddle points and is occasionally stuck in the solution space. We used heuristics to ensure that the solution we arrived at was not the local minima. The optimal mapping obtained after the mapping search stage is further enhanced by employing the core-swapping heuristic.

5.2. GNN-RL model description

In this article, a GNN-based RL approach presents a novel way to map multiple applications onto the NoC. Fig. 7 presents an overview of the GNN-RL architecture. The mapping problem is formulated as a multi-agent search in a given environment. The NTG is the environment, and the ACG nodes act as agents. The proposed model has several network layers, and each layer gets information from all the layers before it and sends its feature maps to all the layers after it to keep feedforward going. The approach employs the explore and exploit policy to search the entire design space and select states based on the probability matrix generated by the model for each router-core combination. Once all agents find their placement, the communication cost is calculated, and a reward is determined based on the cost. An observation matrix is generated at every iteration consisting of communication cost, reward, agent's location, and details of their neighbors to maintain the contextual information of the cores. The KL divergence loss function calculates the loss, and the Adam optimizer adjusts the weight of all the layers in the model with the calculated loss. The number of search cycles depends on the size of the ACG and NoC topology graph (NTG), where a larger search space requires more iterations.

5.2.1. Architecture

The mapping graph is dynamic in a multi-agent search and constantly changes as agents move around. As the ecosystem evolves, it becomes necessary for the model to adapt and capture the dynamics of the mapping graph. To address this challenge, the mapping problem is formulated as a partly observable MDP in this article. It involves agents in the application network collecting data on their location in the NoC topology, determining their next movement, and receiving rewards

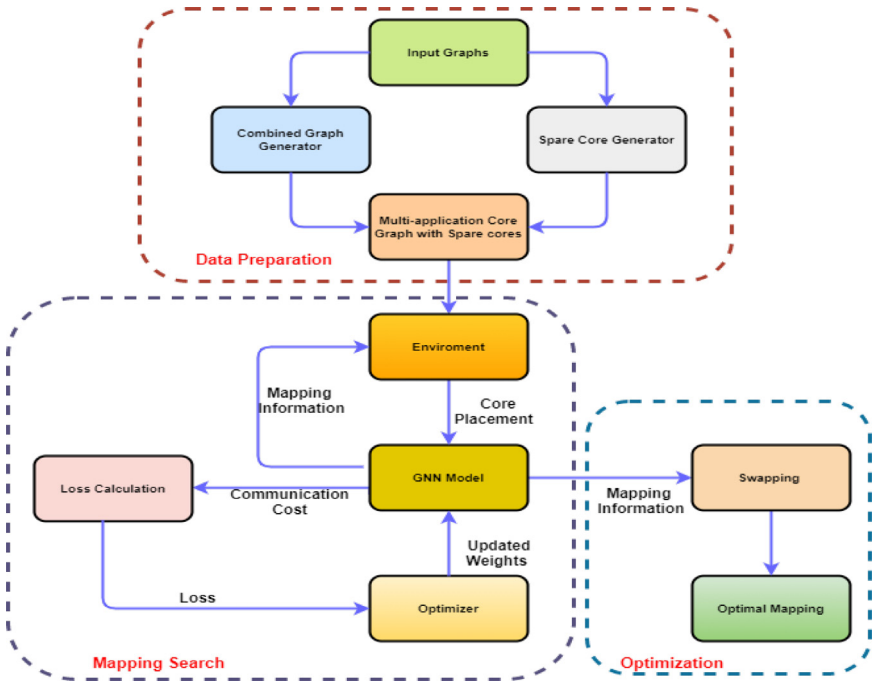

Fig. 6. FANC flowchart.

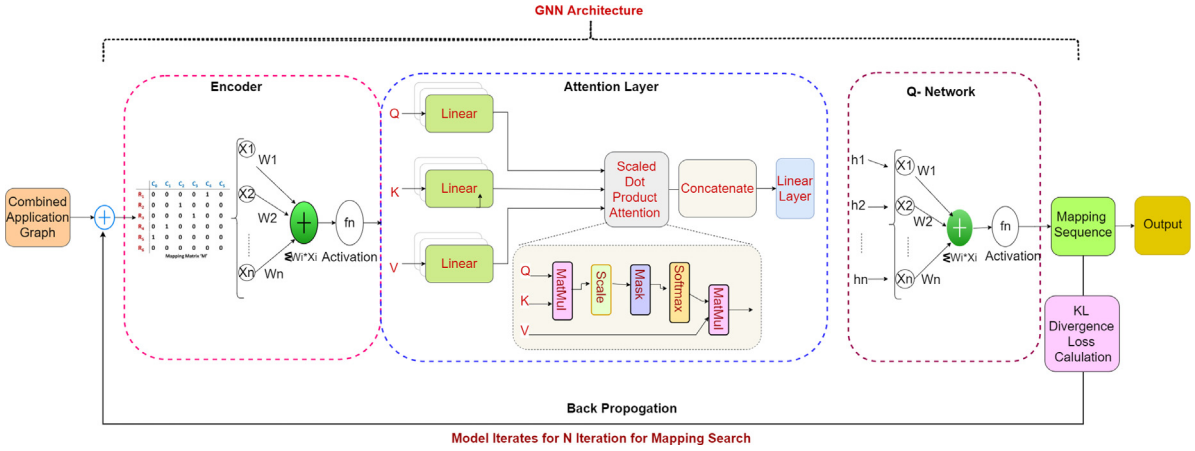

Fig. 7. FANC architecture.

after each iteration. Fig. 7 illustrates the three main components of the proposed model: the encoder, the attention layer, and the Q network. The encoder maps agents' observations to a low-dimensional latent space, which is then passed through the attention layer to obtain an attention matrix. The Q network then utilizes the attention map to generate the optimal action for each agent. By incorporating the proposed approach, the model can effectively adapt to the dynamics of the mapping graph in a multi-agent environment, leading to improved communication cost.

5.2.2. Encoder

The encoder, denoted as $ENC(Mapping;\Psi)$, is designed to efficiently transform the attributes of an input graph into a low-dimensional representation. It receives random mapping information of ACG onto NTG as input to transform graph attributes into a low-dimensional feature matrix. The effectiveness of this design depends on the ability of the resulting embedding to capture the essential data of the network that encapsulates the change in input. We introduce a parameterization function Ψ that maps the input to generate the node embedding matrix E to achieve this goal. This matrix captures different attributes of the input graph. The resulting embeddings of nodes are independent of their proximity to the original graph, but are based on their local representations. This means that nodes with similar local representations will have equivalent embeddings regardless of their location in the network.

5.2.3. Attention layer

The proposed work adapts the attention mechanism to operate on graph-structured mapping data by utilizing graph convolutional layers to encode the locations of the application core on NoC and their neighbor's features. The attention mechanism is applied to the output of the graph convolutional layer, which produces a matrix of core representations. Each row of the matrix corresponds to a single core's representation, while the columns represent different features considered for the processing, for example, location, neighbors, etc.

At the time t, a feature matrix M_t^F of size Cxf_l, where C is the number of cores, and f_l is the length of the feature vector of all local feature vectors is created. Then, Adjacency Matrix A is generated by

aggregating one-hop information from all neighbors. The local feature matrix for each core is generated by multiplying these two matrices. By using masked self-attention layers, the attention mechanism can operate independently on each row, allowing the agent to focus on different features of each node. The layers are stacked to enable cores to evaluate their neighbors' properties and modify weights to differentiate cores in a neighborhood without requiring prior knowledge of network topology.

Unlike typical neural networks that use compressed context vectors to store sequential information, the proposed attention layer helps design a neural network that can retain long information sequences. By modifying output weights and linking input and context vectors, the attention layer emphasizes inter-core connection information rather than individual cores, thereby increasing neural network reliability and restricting it to the required data.

The attention mechanism is based on the multi-head dot-product attention mechanism used in natural language processing tasks. In this mechanism, linear transformations transform each core's representation into a query, key, and value representation. The attention scores between a node and its neighbors are computed by taking the dot product between the query and the key representations of each neighbor. These scores weigh the values of neighboring nodes, which are then summed and normalized to obtain the output representation of the core.

The attention mechanism is computed for each head of a multi-head attention layer, allowing the agent to simultaneously attend to multiple aspects of the input. Concatenating all attention heads of the cores and feeding them into a single-layer linear neural network with Logsoftmax non-linearity produces the output representation of each node.

The attention mechanism can be mathematically defined as follows: We have a mapping sequence of input vectors, $M_i = [x_1, x_2, \ldots, x_n]$, where each x_i is a vector of dimension d and represents the router-core pair. We want to compute a new mapping sequence of vectors, $M_o = [y_1, y_2, \ldots, y_n]$, where each y_i is also a vector of dimension d and represents the router-core pair. In self-attention, we compute the weights $w_{i,j}$ between each pair of input vectors x_i and x_j using the dot product of their corresponding feature vectors. We first apply three linear transformations to each input vector x_i to obtain a query vector q_i, a key vector k_i, and a value vector v_i:

$q_i = W_q x_i$; $k_i = W_k x_i$; $v_i = W_v x_i$;

Here, W_q, W_k, and W_v are learned weight matrices of dimensions dxd. We then compute the attention scores between x_i and x_j as the dot product of their corresponding query and key vectors; it is equivalent to finding the relative information of a particular core and set of its neighbor N_r and can be calculated as:

$$a_{ij}^m = \frac{e^{(\tau * W_Q^m x_i * (W_k^m x_j)^T)}}{\sum_{n \in Nr} e_{(\tau * W_Q^m h_i * (W_k^m h_n)^T)}} \tag{7}$$

where $'\tau'$ is a scaling factor and T, indicates the transpose function. The value representations of all the input features are weighted by the relation and summed together for each attention head. Then, the outputs of M attention heads for the agent are concatenated and then fed into the $'\sigma'$ function, i.e., one-layer MLP with Logsoftmax non-linearities, to produce the output of the convolutional layer as follows:

$$H = \sigma(concatenate[\sum_{n \in Nr} a_{ij}^m W_V^m h_i, \forall m \in H]) \tag{8}$$

Finally, this output generates a probability matrix for core-router mapping.

5.2.4. Q-network

Each agent can assemble and reuse observation representations and features from various receptive fields in the proposed architecture. To do this, all features of the preceding layer are concatenated and fed into the Q network using [32]. Each agent uniquely contributes to a strategy that considers cooperation in various areas. The Q network is a critical component of the proposed approach. It is used to predict the next action of an agent by incorporating observations and attributes from previous layers. It operates iteratively, where the agent reduces communication costs by acquiring the knowledge of router-to-core mappings. To incentivize efficient communication, the agent offers rewards determined by the current action, state, and communication costs. The Q-value of each router-core pair is retained in a Q-matrix, which is updated throughout the training process until all epochs have been completed. Once a router-core pair has been mapped, it cannot be reassigned to any other router or core. Q-matrix is updated at each epoch, and the associated reward value (R) is computed.

Algorithm 1: FANC

Input : Application Core Graphs (ACGs) and NoC Topology
Output: Mapping of Routers and Cores

Initialize Map=0, Obs = 0, Cost = 0, loss = 0, I = No. of Iterations, No. of Cores = C
Combined ACG ← Merge(ACGs)
Spare core ← Find(faulty Core)
FT-ACG ← Add(Combine ACG, Spare Core)
Compute Bandwidth matrix
Compute Distance Matrix
Target Cost ← Best Cost (FT-ACG)
Placement ← Random(Core,Routers)
for $i \leftarrow 1$ **to** I **do**
  $Obs \leftarrow$ Model (Placement)
  $Action \leftarrow$ prob2loc(Obs)
  Observation, Reward, Comm. Cost ← Perform(Action)
  Loss = KLDiv(Target Cost-Comm. Cost)
  $Model \leftarrow$ Optimize (loss)
end
$Map \leftarrow$ Swap (Map)
$Comm.Cost \leftarrow$ Cost(Map)
return *Map, Cost*

5.3. FANC working example

Fig. 8 and Table 2 demonstrate multi-application mapping using FANC. ACG1 and ACG2 are the two applications with three cores each, whereas the NTG is a 2×3 mesh network. The combined application graph has five cores. The mapping s accomplished by rearranging the application cores $C_0, C_1, C_2, \ldots, C_4$ and C_S in the combined application core graph onto the NoC topology's routers R_1 to R_6 to reduce communication costs. The execution steps of FANC given in Algorithm 1 are explained here:

Stage 1: Data Preparation

a. Initially, all the required variables of the program are initialized as follows: Mapping Matrix 'Map'= 0, Observation Matrix 'Obs' = 0, Cost = 0, loss = 0, I (No. of Iterations)=100, No. of Cores = 5.
b. The input application graphs ACG1 and ACG2 are combined using the Merge function, generating a Combined Application Graph (CAG).
c. Most communicating core C_0 is selected as faulty core, and Faulty core is injected in the CAG.
d. A spare core C_S is added in the CAG to make application mapping fault-tolerant.
e. A Bandwidth Matrix is created with the weight of the links of CAG. The communication bandwidth of the links between C_0 and C_1 is 40 bps, between C_0 and C_2 is 20 bps, between C_2 and C_3 is 30 bps, and between C_2 and C_4 is 10 bps. C_S has the same bandwidth as $C0$ that is between C_S and C_1 is 40 bps and between C_S and C_2 is 20 bps because it is a spare core for C_0.
f. A Distance Matrix is created from a given NTG using the XY routing mechanism.

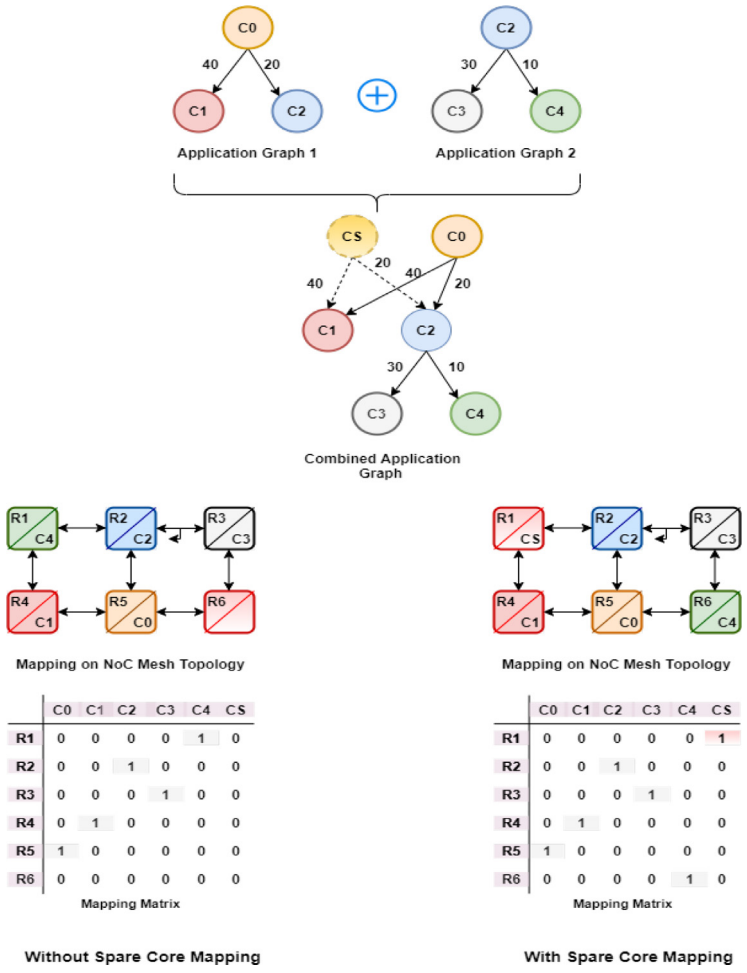

Fig. 8. Application mapping example.

g. Ideal communication cost of the CAG is calculated assuming all the cores are placed at one hop distance onto NoC as an ideal condition. In this example, it is 100 Mbps.
h. Initially, Mapping Matrix 'Map' is a randomly generated mapping for CAG onto NoC.

Stage 2: Mapping Search

i. Mapping Matrix 'Map' is fed to the first stage of GNN-RL architecture i.e., the Encoder section.
j. Encoder encodes the information of the mapping matrix to transform mapping attributes into a low-dimensional representation as a node embedding matrix E.
k. The node embedding matrix E is fed to the convolution layer of the Attention section of GNN-RL architecture. At the time t, a feature matrix M_t^F and Adjacency Matrix A are generated by aggregating one-hop information from all neighbors. The local feature matrix for each core is generated by multiplying these two matrices.
l. The attention scores between a node and its neighbors are computed using the dot product between their query and key representations. The attention mechanism is computed for each head and concatenated.
m. The concatenated result fed into a single-layer linear neural network with Logsoftmax nonlinearity. It produces the output representation of each node as a probability matrix.
n. Probability matrix is fed to the Q network. It is used to predict the next action of an agent. It operates iteratively for all the cores. The Q-value of each router-core pair is retained in a Q-matrix, which is updated throughout the training process until all epochs have been completed.
o. Mapping Matrix 'Map' is updated based on the new router location predicted by the Q-Network for each core. As shown in Fig. 8, C_4 is assigned to R_1, C_2 maps to R_2, C_3 maps to R_3, C_1 maps to R_4 and C_0 maps to R_5 without fault-tolerant mapping. When introducing the spare core C_S for fault tolerance, the ML model suggested the application mapping, as C_S is assigned to R_1, C_2 maps to R_2, C_3 maps to R_3, C_1 maps to R_4, C_0 maps to R_5 and C_4 maps to R_6.
p. Communication Cost is calculated, and a Reward is generated.
q. Loss of the model is calculated, and the weight of the layers in the Network is adjusted as per the loss.
r. Step 'i' to Step 'q' repeats for the given number of iterations (in this example, it is 100).

Stage 3: Optimization

s. After completion of all iterations, an optimal mapping is achieved.
t. To check for any further optimization in mapping, router-core pairs swap their locations and update the mapping matrix.

Table 2 shows the calculation of the communication cost, which was calculated using Eq. (6). The first column presents the sequence of mapping of cores to routers. The second column indicates the communication link details according to the input application graph. The third column shows how much bandwidth each of the communication links in the second column needs. The fourth column presents the distance between the router in the NoC topology on which communicating cores

Table 2
Application mapping example.

Mapping without spare core					Mapping with spare core				
Mapping sequence	Application graph links	Bandwidth 'B'(bps)	Distance 'D' (Hops)	Link cost (B*D)	Mapping sequence	Application graph links	Bandwidth 'B'(bps)	Distance 'D' (Hops)	Link cost (B*D)
R1-C4	C0-C1	40	1	40	R1-CS	CS-C1	40	1	40
R2-C2	C0-C2	20	1	20	R2-C2	CS-C2	20	1	20
R3-C3	C2-C3	30	1	30	R3-C3	C2-C3	30	1	30
R4-C1	C2-C4	10	1	10	R4-C1	C2-C4	10	2	20
R5-C0	Total Communication Cost			100	R5-C0 R6-C4	Total Communication Cost			110

are mapped, and the fifth column presents the communication cost of the link, which is the product of bandwidth and distance.

For example, $R_5 - C_0$ indicates that C_0 is mapped to the R_5 router, and $C_0 - C_1$ indicates a communication link in the application graph with a bandwidth of 40 bps. C_0 is placed at R_5 and C_1 is placed at R_4. Their distance in hops is one on the NoC topology. Hence the link cost is 40 bps (40×1). Similarly, the link cost of all the other links is calculated. The communication cost in the fault-free case is 100 bps, and in the fault-tolerant case with a spare core, the communication cost is 110 bps. Given the locations of the spare cores, the proposed mapping has the lowest communication cost, 110 bps. The communication cost obtained using the proposed approach is high due to the limitation of the NoC topology, as no other router is available for mapping. However, the proposed method is flexible in selecting routers to map spare cores and learns multi-application spare core placement without explicit programming.

6. Experimental results and discussion

We found very few works in the literature that address fault-tolerant multi-application mapping. The work closest to FANC is mentioned in [1]. Hence, FANC's results are compared with those reported in [1]. The approach in [1] employs the reconfiguration method to reduce the cost of communication. PSO is used before reconfiguration to develop an optimum fault-tolerant mapping. FANC can improve the results by replacing PSO as an optimization strategy in [1] and can be paired with a reconfigurable approach to efficiently address the problem of fault-tolerant multi-applications mapping. In the approach [1], the network sizes for the application mapping were not specified; therefore, it has been assumed that the application mapping was performed on the smallest network size available to accommodate the total number of cores.

The results section is organized as follows, Section 6.1 describes the experiment setup, and the results of testing fault-free multi-application scenarios in regular topologies with FANC are presented in Section 6.2. Section 6.3 analyzes the cost of communication with single-core failure in one application. Section 6.4 discusses single-core failure communication costs for all applications. Section 6.5 analyzes communication costs if all applications share multiple-core failures. Section 6.6 shows the results of the dynamic simulation.

6.1. Experiment setup

FANC is implemented and simulated using Python, which runs on a computer equipped with an Intel Xeon (R) CPU@ 2.20 GHz 2.0 GHz 16 GB of RAM. MPEG, MWD, 263ENC, MP3ENC, and 263DEC are embedded benchmark applications [33] used for evaluation, while synthetic applications G1–G4 were created using the TGFF tool [34]. We experimented with different applications, network sizes, and fault scenarios to better understand the static and dynamic behavior of fault-tolerant application mapping solutions. The experimental flow for the FANC analysis is shown in Fig. 9.

In this research, static and dynamic simulations are conducted to evaluate the effectiveness of the proposed method. The static simulation is performed using a Python script, which utilizes the mapping matrix generated by FANC to calculate communication cost. The quality of the mapping solution and the efficiency of FANC are analyzed based on the communication cost (in Mbps) and computation time. To study the dynamic behavior of the proposed approach, a SystemC-based cycle-accurate NoC simulator [35] and the Orion 2.0 tool [36] are employed. The dynamic simulation involves the evaluation of network latency (in clock cycles), throughput (in flits/cycle/core), and total power consumption (in μW). The mapping matrix generated by FANC is converted into an appropriate input format for the cycle-accurate NoC simulator to calculate network latency and throughput. The mapping matrix and the NoC configuration parameters are inputs to the Orion 2.0 tool to calculate power consumption.

FANC assesses several multi-applications with multiple topologies and core failure scenarios. The input application sequence is created at random using synthetic and embedded benchmarks. For example, MPEG4 and MWD applications were integrated to form new MPEG4+MWD application graphs. In the result, tables, "MPEG4 (MPEG4+MWD)" denotes the communication cost for MPEG4 in the case of a combined MPEG4+MWD application. All other table entries use a similar syntax. To conduct the experiments, the following NoC configuration was considered:

- Traffic: Application-Specific
- Channel width: 32 bits
- Packed size: 64 flits
- Buffer Length: 8 flits
- Flit size: 32 bits
- Router Type: Wormhole

Hardware Implementation: For implementing a fault-tolerant mechanism in hardware, we considered the reconfiguration methodology based on the architecture proposed in [1]. In this architecture, different cores are connected to routers via multiplexers. The interconnections between the routers and the routing algorithm used remain the same as those of [1]. Fig. 10 shows the fault-tolerant reconfigurable architecture for 4×4 mesh topology.

Depending on the number of spare cores to be placed, the size of the reconfigurable architecture varies. Referring to the standard mesh topology, each router contains a maximum of four global ports to connect to four neighbor routers and one local port to connect to one core via a multiplexer. Multiplexers allow changes in the connections between the cores and routers. These connections are changed to minimize the communication cost.

The proposed work can be validated by implementing an NoC design using an FPGA and a five-port virtual channel-based wormhole router architecture. The design begins by synthesizing a single NoC router with a specific addressing scheme and routing algorithm. Once the NoC topologies are synthesized on the FPGA, the static and dynamic power on the chip can be calculated using the power analyzer tool. The power result provides resource utilization and power profiles that can be used to validate fault-tolerant application mapping solutions.

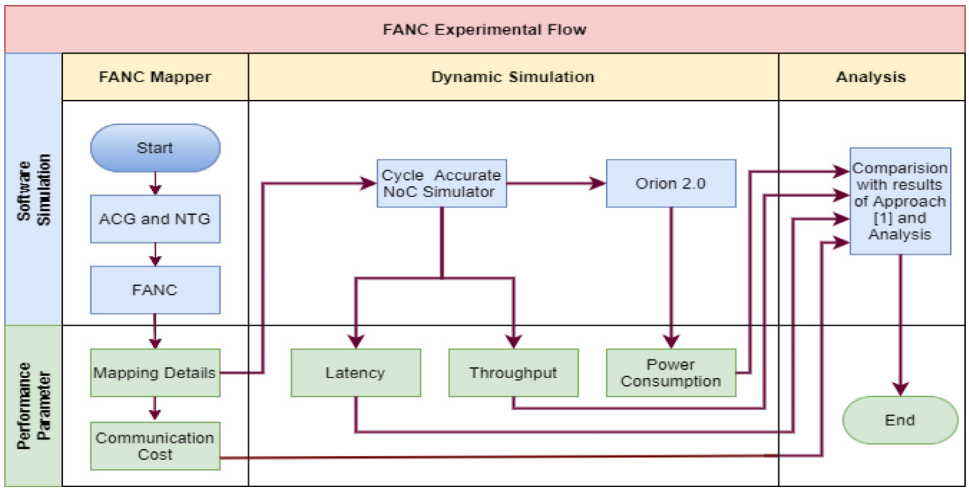

Fig. 9. Experimental flow for FANC analysis.

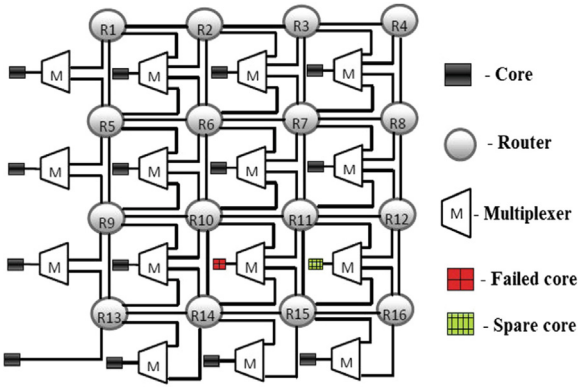

Fig. 10. 4 × 4 mesh based Fault-Tolerant reconfigurable NoC architecture [1].

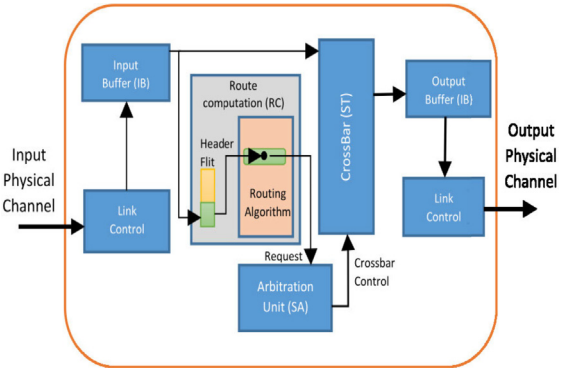

Fig. 11. NoC router data flow architecture on FPGA [37].

After mapping the application to the NoC topology, real fault-tolerance-specific traces are given to the cores via a virtual input–output IP core. Traffic traces for the application can be taken from the NoC simulator [27]. Fig. 11 shows the NoC router data flow architecture [37] that can be used for the proposed work and implemented on an FPGA using Verilog. The detailed specifications of the NoC router can be seen in Table 3. Fault tolerance prototyping on the FPGA of the proposed method is a work in progress, and the same can be presented in our future work.

Table 3
NoC router parameter for synthesizing on FPGA.

NoC router parameter	Value
Architecture	Wormhole
Topology	Application-Specific
No. of Virtual channels	4/3/2/1
Buffer type	FIFO Buffer
Packet length	4 flits
Flit width	32 Bits
FIFO depth	6/8
Routing algorithm	X-Y Deterministic
Flow control	Credit Based
Arbiter type	Round Robin
Traffic pattern	Application Specific

6.2. Multi-application mapping onto regular topology without core failure

Choosing the best NoC topology to meet the application bandwidth, latency, cost, and space requirements takes time and effort. NoC topologies affect resource management, scalability, routing, and flow management. A generalized fault-tolerant multi-application mapping technique for regular NoC topologies has yet to be found in the literature. This has motivated us to generalize the FANC for multi-application mapping onto regular NoC topologies. Table 4 shows the total communication cost for benchmarks and synthetic applications in multi-application mapping onto regular topologies without fault. Since we have not found any similar work for comparison, we have considered the result obtained for fault-free multi-application mapping with mesh topology as a reference for comparing the communication cost of other topologies. Among regular topologies, mesh topology has gained more consideration from designers due to its simplicity. Mesh topology has advantages, such as a single PE failure will not break the network. Each PE has dedicated point-to-point connectivity, reducing congestion and offering privacy, security, and easy fault identification over other topologies.

FANC finds the optimal mapping in different regular NoC topologies without topology-specific training, whereas traditional approaches need training. FANC is evaluated by mapping embedded benchmark and synthetic applications onto the mesh, torus, ring, and spidergon topologies. FANC scalability for different-size networks is assessed by mapping input applications on the smallest possible NoC topologies and calculating communication cost.

In Table 4, the first column indicates the combined application graph; for example, MPEG4 and MWD applications were combined to create new MPEG4+MWD application graphs. The second column shows the NoC topology on which the combined application is mapped.

Table 4
Comparison of communication cost for different regular topologies.

Combined application graph	Topology type	NoC routers	Total cores	Application	FANC		
					Cost (Mb/s)	Normalized cost to mesh	Time (s)
MPEG	MESH	25	21	MPEG	3533.00	1.00	38.00
+MWD		25	21	MWD	1472.00	1.00	39.06
MPEG	TORUS	25	21	MPEG	3714.00	1.05	40.39
+MWD		25	21	MWD	1280.00	0.87	40.64
MPEG	RING	25	21	MPEG	5516.00	1.56	39.63
+MWD		25	21	MWD	2848.00	1.93	34.93
MPEG	SPIDERGON	25	21	MPEG	4214.00	1.19	33.69
+MWD		25	21	MWD	1536.00	1.04	32.89
MP3ENC	MESH	30	29	MP3ENC	17.91	1.00	40.58
+263DEC		30	29	263DEC	19.93	1.00	39.66
+263ENC		30	29	263ENC	255.30	1.00	38.87
MP3ENC	TORUS	30	29	MP3ENC	19.39	1.08	45.04
+263DEC		30	29	263DEC	21.30	1.07	43.98
+263ENC		30	29	263ENC	255.30	1.00	43.73
MP3ENC	RING	30	29	MP3ENC	19.92	1.11	30.88
+263DEC		30	29	263DEC	24.16	1.21	30.40
+263ENC		30	29	263ENC	404.80	1.59	30.55
MP3ENC	SPIDERGON	30	29	MP3ENC	18.08	1.01	31.58
+263DEC		30	29	263DEC	20.22	1.01	31.11
+263ENC		30	29	263ENC	255.07	1.00	31.28
G1	MESH	36	36	G1	27384.07	1.00	85.20
+G2		36	36	G2	11753.41	1.00	87.16
+G3		36	36	G3	8219.65	1.00	85.63
G1	TORUS	36	36	G1	35691.01	1.30	100.43
+G2		36	36	G2	10978.23	0.93	99.13
+G3		36	36	G3	6859.26	0.83	99.27
G1	RING	36	36	G1	64214.22	2.34	55.80
+G2		36	36	G2	18477.37	1.57	54.48
+G3		36	36	G3	14587.44	1.77	55.88
G1	SPIDERGON	36	36	G1	27429.03	1.00	57.65
+G2		36	36	G2	12302.30	1.05	58.13
+G3		36	36	G3	7567.48	0.92	57.51
G1	MESH	42	39	G1	29423.31	1.00	117.20
+G4		42	39	G4	73018.14	1.00	114.50
G1	TORUS	42	39	G1	28701.43	0.98	135.20
+G4		42	39	G4	69813.94	0.96	137.10
G1	RING	42	39	G1	50451.82	1.71	68.46
+G4		42	39	G4	160027.33	2.19	67.70
G1	SPIDERGON	42	39	G1	25347.29	0.86	68.12
+G4		42	39	G4	59782.12	0.82	68.24

The third column mentions the number of routers available for mapping, and the fourth column mentions the total number of cores in the combined application graph. The fifth column indicates the individual applications from the combined application graph, and their communication cost is reported in column six. Column seven presents the normalized communication cost to the mesh topology communication cost; finally, column eight gives the computation time.

To demonstrate the generalized learning capability of FANC, initially, results are generated for different multi-application core graphs with a mesh topology. Then, without retraining the model or adjusting any configuration, other topologies are fed as input to FANC to generate the results.

FANC effectively suggests the optimal topology for any given input application, among a range of regular topologies, in a few seconds. For example, FANC predicts that the torus topology is preferable for the combined application graph of MPEG+MWD because of its lower total cost than other topologies. Although the cost for MPEG with torus topology is 5% higher than the cost with mesh topology, but MWD has a 13% lower cost. Similarly, for MP3ENC+263DEC+263ENC, mesh topology can be preferred for mapping, as it incurs lower costs than other topologies. Compared to the cost of MP3ENC, 263DEC, and 263ENC with mesh topology, the overhead for MP3ENC is up to 11%, for 263DEC, it is up to 59%, and for 263ENC, it is up to 7% with other topologies. Furthermore, FANC shows that the spidergon topology for random graphs (G1+G2+G3 and G1+G4) is better as it incurs less overall communication cost than other topologies. In the case of G1+G2+G3, the cost of G1 remains unchanged, G2 is 5% better, and G3 is 7% better than the cost with the mesh topology. For G1+G4, the cost of G1 is 14% and the cost of G2 is 18% less than the cost of the mesh topology.

The results in Table 4 demonstrate the potential of FANC in efficiently computing the optimal mapping and communication cost for all topologies. The average computation time is 60.24 s, the maximum computation time is 137.10 s, and the minimum computation time is 30.40 s. Considering the size and complexity of the graphs evaluated, FANC can be considered as a promising approach.

6.3. Single core failure

In this section, we conducted experiments with single-core failure in one application for a given multi-application input. It was assumed that one core fails at a time and is not part of other applications. In the fault-free scenario for multiple applications, such as MPEG+MWD, the total number of cores was 21. However, if a single core fails, a spare core is added to the combined core graph, increasing the number of cores to 22. It was assumed that one core failed at a time in constituting

Table 5
Communication cost with single core failure.

Combined application graph	Application	Faulty cores	Total cores	NoC routers	FANC		Approach [1]		
					Cost (Mb/s)	Time (s)	Cost (Mb/s)	Normalized cost to FAN	Average improvement
MPEG	MPEG	idct	22	25	4274.00	36.41	6611.50	1.55	1.79
+MWD	MWD				1568.00	36.74	3296.00	2.10	
MPEG	MPEG		22	25	3739.00	36.20	4138.00	1.11	
+MWD	MWD	hvs			1696.00	35.92	4064.00	2.40	
MP3ENC	MP3ENC	ft			23.34	31.22	67.55	2.89	2.98
+263DEC	263DEC		30	30	230.40	43.73	438.58	1.90	
+263ENC	263ENC				20.84	48.60	80.26	3.85	
MP3ENC	MP3ENC				17.57	44.83	60.25	3.43	
+263DEC	263DEC	mem2	30	30	230.40	31.24	666.29	2.89	
+263ENC	263ENC				24.53	48.60	76.38	3.11	
MP3ENC	MP3ENC				17.57	44.83	67.55	3.84	
+263DEC	263DEC		30	30	407.36	43.73	438.58	1.08	
+263ENC	263ENC	dct			20.84	31.56	80.26	3.85	
G1	G1	T7			32889.42	56.56	60306.59	1.83	3.30
+G2	G2		37	42	12292.58	57.73	33761.78	2.75	
+G3	G3				6859.56	57.14	20568.71	3.00	
G1	G1				28239.56	59.09	67570.70	2.39	
+G2	G2	T6	37	42	11663.56	56.36	28190.24	2.42	
+G3	G3				6859.56	57.14	13300.53	1.94	
G1	G1				28239.56	59.09	59547.90	2.11	
+G2	G2		37	42	12292.58	57.73	126696.39	10.31	
+G3	G3	T22			8928.28	55.78	26735.67	2.99	
G1	G1	T16	40	42	31491.01	47.30	66740.60	2.12	6.27
+G4	G4				10969.31	60.19	171599.92	15.64	
G1	G1		40	42	30188.33	60.82	56126.12	1.86	
+G4	G4	T17			33630.38	60.52	183239.00	5.45	
Average improvement									3.59

applications of a combined application graph. MPEG had a faulty core, while MWD was fault-free, and vice versa. This assumption continued for all test cases.

As shown in Table 5, the average improvement of FANC results is 3.59 times higher than the results of approach [1]. This improvement can be attributed to the non-explicit programming nature of FANC, which places spare cores in the best possible location. The average improvement achieved in communication cost was 1.79 times for MPEG+MWD and 2.98 times for MP3ENC+263DEC+263ENC compared to the cost reported in the approach [1]. For random applications G1+G2+G3, it was assumed that one core failed in G1 while G2 and G3 were fault-free. In the second case, it was assumed that a core in G2 had failed, but G1 and G3 remained fault-free. In the third instance, one core in the G3 application failed, while G2 and G1 remained fault-free. Similarly, for G1+G4, the assumption remained the same.

The average improvement achieved was 3.30 times for G1+G2+G3 and 6.27 times for G1+G4 compared to the cost reported for the approach [1]. FANC is also very efficient in the time taken for computation. On average, it takes less than one minute for the computation. The ability of FANC to consume less computational time allowed it to increase search cycles to improve the result further. As seen in Table 5, there is a reduction in communication costs and execution time for all types of multiapplication scenarios with single-core failure scenarios using FANC.

6.4. Common single core failure

In this section, the results of experiments carried out to calculate communication costs for a set of input applications where all input applications share the failed core are presented in Table 6. For MPEG+MWD, we assumed that the mem1 core is the failed core and that a spare core with the same functionality has been incorporated into the application to address the fault. The results for MPEG+MWD obtained using FANC have shown an improvement of 1.23 times compared to the results of the approach [1]. Similarly, in the case of MP3ENC+263DEC+26ENC, bitres1 core is a failed core, and the FANC results are better than the results of the approach [1] by 2.79 times.

In the case of the combined application graphs of random application, TO is considered the failed core for G1+G2+G3 and G1+G4 and FANC achieves 2.52 times and 8.08 times better results than the approach [1], respectively. The improved communication cost in different scenarios showed the effectiveness of FANC in fault-tolerant cases. Again, FANC achieved the results in less than a minute on average in this test case. This demonstrates FANC's adaptability to various core failures in a multi-application context.

6.5. Common multi core failure

This section presents the experimental results of multi-application mapping with multiple common core failures. The performance evaluation of the proposed FANC is presented in Table 7, which provides the communication costs and execution times for different test scenarios. The number of cores increased by two due to adding two spare cores during mapping to compensate for two core failures.

Compared to the approach [1], FANC significantly improved communication costs by up to an average of 2.72 times in different test scenarios. Specifically, for MPEG and MWD applications with mem1 and mem2 as shared failed cores in the MPEG+MWD case, FANC improved the communication cost by 1.91 times for MPEG and 2.68 times for MWD. Furthermore, for MP3ENC + 263DEC + 263ENC with bitres1 and bitres2 as shared failed cores, the communication cost improved by 2.36 times for MP3ENC, 1.50 times for 263DEC, and 4.69 times for 263ENC compared to the results of the approach [1]. Similarly, for random applications G1+G2+G3 and G1+G4, T0, and T1 were considered as shared failed cores. FANC improved communication costs by 2.12 times for G1, 1.99 times for G2, and 2.03 times for G3 compared to the results of the approach [1] in the case of G1+G2+G3. In the case of G1+G4, communication cost is improved 2.25 times for G1, and 5.11 times for G4 compared to the results of the approach [1].

Table 6
Communication cost with common single core failure.

Combined application graph	Application	Faulty cores	Total cores	NoC routers	FANC		Approach [1]		
					Cost (Mb/s)	Time (s)	Cost (Mb/s)	Normalized cost to FAN	Average improvement
MPEG	MPEG	mem1	22	25	7788.00	38.60	13935.00	0.44	1.23
+MWD	MWD	mem1			1520.30	39.46	3072.00	2.02	
MP3ENC	MP3ENC	bitres1			21.58	50.51	47.59	2.21	
+263DEC	263DEC	bitres1	30	30	233.78	49.76	450.86	1.93	2.79
+263ENC	263ENC	bitres1			22.67	48.8	96.19	4.24	
G1	G1	TO			36304.56	59.6	83852.81	2.31	
+G2	G2	TO	37	42	13114.36	58.48	34646.07	2.64	2.52
+G3	G3	TO			9662.36	56.49	25283.45	2.62	
G1	G1	TO	40	42	37838.61	60.02	68954.49	1.82	8.08
+G4	G4	TO			124771.41	60.83	178961.01	1.43	
Average improvement									3.66

Table 7
Communication cost with common multi core failure.

Combined application graph	Application	Faulty cores	Total cores	NoC routers	FANC		Approach [1]		
					Cost (Mb/s)	Time (s)	Cost (Mb/s)	Normalized cost to FAN	Average improvement
MPEG	MPEG	mem1,	23	25	6829.50	38.20	13069.00	1.91	2.30
+MWD	MWD	mem2			1824.25	36.24	4896.00	2.68	
MP3ENC	MP3ENC	bitres1,			21.69	50.51	51.11	2.36	
+263DEC	263DEC	bitres2	31	36	297.68	49.76	447.65	1.50	2.85
+263ENC	263ENC				24.71	48.80	115.88	4.69	
G1	G1				37814.84	57.44	80145.80	2.12	
+G2	G2	T0,T1	38	42	14462.77	58.45	28810.42	1.99	2.05
+G3	G3				13706.01	58.69	27806.83	2.03	
G1	G1	T0,T1	41	42	35097.70	68.25	78942.64	2.25	3.68
+G4	G4				40671.87	83.65	207951.10	5.11	
Average improvement									2.72

The effectiveness of FANC in a multi-core failure scenario is also demonstrated in terms of computational time. FANC found the optimal mapping and communication cost for any given application in less than two minutes, indicating its efficiency in consuming less computational time. This feature also enables FANC to increase the search cycles for large application graphs without exponentially increasing the search time. Furthermore, FANC allows efficient positioning of spare cores, demonstrating scalability while improving communication costs.

6.6. Dynamic simulation results

This section presents dynamic simulations that analyze the network behavior after the application mapping. Simulations were carried out for multimedia and synthetic applications, with different failure scenarios. The cycle-accurate simulator was utilized to calculate the average network latency and throughput by providing application-specific traffic patterns to the simulator. This provided a comprehensive analysis of network behavior between the FANC and the approach [1]. The simulation parameters set for the experiment are as follows:

- Clock period: 5 clock cycles
- Packet length: 64 flits
- Flit length: 32 bits
- Simulation time: 2,00,000 router clock cycles
- Saturation time: 10,000 router clock cycles
- Routing algorithm: X-Y Routing
- Mapping: FANC and Approach [1]

The dynamic simulation results for different mesh network applications are presented in Tables 8 and 9. The first column of these tables represents the combined application graph, the second column presents the faulty core, and the third column represents the number of cores in an application. The network latency is presented in columns four through six, throughput is presented in columns seven through nine, and power consumption is presented in columns ten through twelve.

The average network latency of the proposed approach is calculated using the formula shown in Eq. (9), where the total latency of the network is calculated for all edges of the application, and the total packets received in time represent the data packets received by the application cores within the simulation time frame. The results presented in Tables 8 and 9 demonstrate that the proposed approach significantly improves network latency compared to the existing approach [1].

The improvement in network latency can be attributed to the optimal application mapping of the proposed approach, which reduces the average number of network hops between cores. The number of hops is directly proportional to the latency of an application. The proposed approach maps the cores as closely as possible to minimize the hop count, leading to lower average network latency for the applications mapped onto the mesh network. Specifically, the proposed approach achieves an average network latency improvement of 9% over the existing approach [1].

$$Average\ Network\ Latency = \frac{Total\ Latency\ of\ the\ Network}{Total\ Packets\ received\ within\ time\ *\ Clock\ period} \tag{9}$$

In addition, the proposed approach also improved the network throughput. Throughput is defined as the amount of data transmitted per unit of time. The improvement in throughput is attributed to the efficient use of available resources and optimal data routing. The maximum packet receiving time depends on the average hop count, which is the distance between an application's source and destination core pairs. The throughput (in flits/cycles/IP) is calculated using the formula

Table 8
Dynamic simulation result for common core failure to different applications.

Combined application graph	Faulty cores	Total cores	Average network latency (in Cycle)			Throughput (in flits/cycle/core)			Total power (in μW)		
			FANC	Approach [1]	Normalized w.r.t FANC	FANC	Approach [1]	Normalized w.r.t FANC	FANC	Approach [1]	Normalized w.r.t FANC
MPEG +MWD	mem1	22	90.57	99.36	1.10	0.000126	0.000123	0.98	66.15	71.23	1.08
MPEG +MWD	mem1, mem2	23	87.71	95.78	1.09	0.000121	0.000118	0.98	66.15	71.23	1.08
MP3ENC +263DEC +263ENC	Bitres1	30	90.50	96.25	1.06	0.000694	0.000694	1.00	79.38	85.11	1.07
MP3ENC +263DEC +263ENC	Bitres1, Bitres2	31	90.10	97.28	1.08	0.000656	0.000661	1.01	79.38	85.12	1.07
G1 +G2 +G3	T0	37	92.60	100.35	1.08	0.000756	0.000759	1.00	79.40	82.11	1.03
G1 +G2 +G3	T0, T1	38	91.15	98.35	1.08	0.000591	0.000590	1.00	79.41	82.21	1.04
G1 +G4	T0	40	120.88	135.43	1.12	0.000794	0.000809	1.02	111.17	120.37	1.08
G1 +G4	T0, T1	41	110.69	124.51	1.12	0.000694	0.000639	0.92	111.16	120.38	1.08
Average improvement					1.09	Average improvement		0.99	Average improvement		1.07

shown in Eq. (10).

$$Throughput = \frac{Total\ Packets\ received\ *\ Packet\ Length\ *\ Clock\ period}{Total\ Cores\ *\ Max.Packet\ received\ time\ -\ saturation\ time} \quad (10)$$

Where the packet length, the clock period, the total number of cores per application, and the saturation time remain the same for our approach and the approach [1], therefore, Eq. (10) can be represented further as follows.

$$Throughput \propto \frac{Total\ Packets\ received\ within\ time\ window}{Maximum\ Packet\ receive\ time} \quad (11)$$

FANC was found to require fewer hops between the cores of an application compared to the approach [1]. This reduces the maximum packet receive time and increases the throughput for applications. The experimental results demonstrated an average improvement of 1% in throughput. This improvement in throughput is a significant achievement in improving network performance and can benefit various applications that require efficient data transmission.

In addition to improving network latency and throughput, the proposed approach effectively reduces power consumption. The power consumption is calculated using the ORION 2.0 power tool, which includes both the switching power and the internal power of the network. The switching power consumption is caused by the switching activity of the routers that transmit the data bits from the source core to the destination core in an application. However, the internal power consumption in the network is due to the free clock of the routers.

We observed a significant reduction in power consumption using the FANC. This is primarily attributed to optimal data routing and efficient utilization of available resources, which minimizes the routers' switching activity and reduces the network's internal power consumption. The simulation of power consumption in the NoC is as follows:

$$P_T = P_i + P_s \quad (12)$$

$$P_d = a * C_{eff} * V_{dd} * F \quad (13)$$

$$P_s = N_t * I_{leak} * V_{dd} * K_d \quad (14)$$

The relation between bandwidth and power dissipation P_T can be given by Eq. (15):

$$P_T \propto Bandwidth\ (B_{ij}) \quad (15)$$

The power consumption of the router is a critical metric that must be considered when evaluating network performance. In this research, the power consumption of the router was calculated considering the given application mapping, traffic patterns, and the size of the network. It is worth noting that the network size and the application traffic pattern remain constant for both FANC and the approach [1] during a given simulation period.

The results presented in Tables 8 and 9 demonstrate that the proposed approach significantly reduces the average power consumption of the router compared to the approach [1]. This is because the switching activity factor per router involved in the communication path obtained for our approach is lower than that for the approach [1]. On average, the proposed approach has shown an improvement of 7% in terms of the power consumption of the router compared to the approach [1]. This reduction in power consumption is crucial in energy-constrained systems and highlights the efficiency of the proposed approach in a dynamic environment.

7. Conclusion and future work

This research presents a novel method for fault-tolerant multi-application mapping on a regular NoC using the ML technique. The proposed method optimally maps the primary and spare cores to the NoC topology, which is evaluated through static and dynamic simulation environments for communication cost, network latency, throughput, and power consumption. The performance of the proposed method is compared with an existing reconfiguration technique reported in [1] using several embedded benchmark applications.

The proposed method can identify optimal solutions for unseen graphs without prior training, making it a viable solution for similar real-world problems. Different test scenarios are examined to evaluate the effectiveness of the proposed method, and the results are compared to the findings reported in [1]. The experimental results demonstrate that the proposed method, FANC, achieves an average improvement of 266% in communication cost when experiments are carried out by scaling the network size.

In addition, the proposed method shows promising results for dynamic simulations, with an average improvement of 9% in average network latency, 1% in throughput, and 7% in power consumption.

Table 9
Dynamic simulation result for single core failure to different applications.

Combined application graph	Faulty cores	Total cores	Average network latency (in Cycle)			Throughput (in flits/cycle/core)			Total Power (in μW)		
			FANC	Approach [1]	Normalized w.r.t FANC	FANC	Approach [1]	Normalized w.r.t FANC	FANC	Approach [1]	Normalized w.r.t FANC
MPEG +MWD	idct (MPEG)	22	85.33	91.32	1.07	0.000104	0.000105	1.01	66.15	71.23	1.08
MPEG +MWD	hvs (MWD)	22	83.60	91.38	1.09	0.000090	0.000089	0.99	66.15	71.23	1.08
MP3ENC +263DEC +263ENC	ft (MP3ENC)	30	91.54	100.21	1.09	0.000651	0.000650	1.00	79.38	85.11	1.07
MP3ENC +263DEC +263ENC	mem2 (263DEC)	30	93.85	100.58	1.07	0.000585	0.000580	0.99	79.38	85.12	1.07
MP3ENC +263DEC +263ENC	dct (263ENC)	30	92.65	95.36	1.03	0.000610	0.000615	1.01	79.38	85.11	1.07
G1 +G2 +G3	T7(G1)	37	96.54	105.36	1.09	0.000517	0.000535	1.03	79.40	85.11	1.07
G1 +G2 +G3	T6(G2)	37	96.13	98.25	1.02	0.000694	0.000701	1.01	79.41	85.11	1.07
G1 +G2 +G3	T22(G3)	37	95.00	101.75	1.07	0.000687	0.000700	1.02	79.40	85.10	1.07
G1 +G4	T16(G1)	40	96.00	100.28	1.04	0.000630	0.000650	1.03	111.16	119.50	1.08
G1 +G4	T17(G4)	40	133.24	134.56	1.01	0.000719	0.000730	1.02	111.17	119.50	1.07
Average improvement					1.06	Average improvement		1.01	Average improvement		1.07

Furthermore, the proposed method demonstrates effective results for application mapping in different regular NoC topologies. The experimental results indicate that the proposed method can address application mapping challenges without significant complexity or processing costs. The proposed method can also be combined with the reconfigurable approach to achieve better performance. In addition, the applicability of the proposed approach can be explored in a 3D NoC environment.

CRediT authorship contribution statement

Jitesh Choudhary: Conceptualization, Methodology, Software, Writing – original draft. **Chitrapu Sai Sudarsan:** Software, Data curation. **Soumya J.:** Conceptualization, Validation, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

The authors thank the Center for the Development of Advanced Computing, India, for their support and encouragement in doing this work.

References

[1] Veda Bhanu, Pranav Kulkarni, Sai Avadhanam, Joshi Soumya, Linga Reddy Cenkeramaddi, Multi-application based fault-tolerant network-on-chip design for mesh topology using reconfigurable architecture, ISBN: 978-981-32-9766-1, 2019, pp. 442–454, http://dx.doi.org/10.1007/978-981-32-9767-8_37.

[2] Johanna Sepúlveda, Marius Strum, Wang Jiang Chau, Guy Gogniat, A multi-objective approach for multi-application NoC mapping, in: 2011 IEEE Second Latin American Symposium on Circuits and Systems, LASCAS, 2011, pp. 1–4, http://dx.doi.org/10.1109/LASCAS.2011.5750275.

[3] Fatemeh Khalili, Hamid R. Zarandi, A fault-tolerant low-energy multi-application mapping onto NoC-based multiprocessors, in: 2012 IEEE 15th International Conference on Computational Science and Engineering, 2012, pp. 421–428, http://dx.doi.org/10.1109/ICCSE.2012.65.

[4] Martha Johanna Sepúlveda, Wang Jiang Chau, Guy Gogniat, Marius Strum, A multi-objective adaptive immune algorithm for multi-application NoC mapping, Analog Integr. Circuits Signal Process. (ISSN: 1573-1979) 73 (3) (2012) 851–860, http://dx.doi.org/10.1007/s10470-012-9869-9.

[5] Haytham Elmiligi, Ahmed A. Morgan, M. Watheq El-Kharashi, Fayez Gebali, Power optimization for application-specific networks-on-chips: A topology-based approach, Microprocess. Microsyst. (ISSN: 0141-9331) 33 (5) (2009) 343–355, http://dx.doi.org/10.1016/j.micpro.2009.03.002.

[6] Waqar Amin, Fawad Hussain, Sheraz Anjum, Sarzamin Khan, Naveed Khan Baloch, Zulqar Nain, Sung Kim, Performance evaluation of application mapping approaches for network-on-chip designs, IEEE Access PP (2020) 1, http://dx.doi.org/10.1109/ACCESS.2020.2982675.

[7] P. Veda Bhanu, Soumya J., Fault-tolerant application mapping on mesh-of-tree based network-on-chip, J. Syst. Archit. (ISSN: 1383-7621) 116 (2021) 102026, http://dx.doi.org/10.1016/j.sysarc.2021.102026.

[8] Nassima Kadri, Mouloud Koudil, A survey on fault-tolerant application mapping techniques for network-on-chip, J. Syst. Archit. (ISSN: 1383-7621) 92 (2019) 39–52, http://dx.doi.org/10.1016/j.sysarc.2018.10.001.

[9] Fatemeh Khalili, Hamid R. Zarandi, A reliability-aware multi-application mapping technique in networks-on-chip, in: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2013, pp. 478–485, http://dx.doi.org/10.1109/PDP.2013.77.

[10] Bo Yang, Liang Guang, Thomas Canhao Xu, Alexander Wei Yin, Tero Säntti, Juha Plosila, Multi-application multi-step mapping method for many-core network-on-chips, in: NORCHIP 2010, 2010, pp. 1–6, http://dx.doi.org/10.1109/NORCHIP.2010.5669454.

[11] Fatemeh Khalili, Hamid R. Zarandi, A fault-aware low-energy spare core allocation in networks-on-chip, in: NORCHIP 2012, 2012, pp. 1–4, http://dx.doi.org/10.1109/NORCHP.2012.6403143.

[12] Di Zhu, Lizhong Chen, Siyu Yue, Timothy M. Pinkston, Massoud Pedram, Balancing on-chip network latency in multi-application mapping for chip-multiprocessors, in: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, 2014, pp. 872–881, http://dx.doi.org/10.1109/IPDPS.2014.94.

[13] Robert Khasanov, Jeronimo Castrillon, Energy-efficient runtime resource management for adaptable multi-application mapping, in: 2020 Design, Automation & Test in Europe Conference & Exhibition, DATE, 2020, pp. 909–914, http://dx.doi.org/10.23919/DATE48585.2020.9116381.

[14] Chanhee Lee, Hokeun Kim, Hae-woo Park, Sungchan Kim, Hyunok Oh, Soonhoi Ha, A task remapping technique for reliable multi-core embedded systems, in: 2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS, 2010, pp. 307–316.

[15] Chen-Ling Chou, Radu Marculescu, FARM: Fault-aware resource management in NoC-based multiprocessor platforms, in: 2011 Design, Automation & Test in Europe, 2011, pp. 1–6, http://dx.doi.org/10.1109/DATE.2011.5763113.

[16] Mehdi Modarressi, Arash Tavakkol, Hamid Sarbazi-Azad, Application-aware topology reconfiguration for on-chip networks, IEEE Trans. VLSI Syst. 19 (2011) 2010–2022, http://dx.doi.org/10.1109/TVLSI.2010.2066586.

[17] Ritesh Parikh, Valeria Bertacco, Resource conscious diagnosis and reconfiguration for NoC permanent faults, IEEE Trans. Comput. 65 (7) (2016) 2241–2256, http://dx.doi.org/10.1109/TC.2015.2479586.

[18] Soumya J., Ashish Sharma, Santanu Chattopadhyay, Multi-application network-on-chip design using global mapping and local reconfiguration, ACM Trans. Reconfigurable Technol. Syst. (ISSN: 1936-7406) 7 (2) (2014) http://dx.doi.org/10.1145/2556944.

[19] Guolei Zhu, Yingmin Wang, A multi-application mapping case study for noc-based mpsocs, in: 2013 IEEE International Conference on Signal Processing, Communications and Computing, ICSPCC 2013, 2013, http://dx.doi.org/10.1109/ICSPCC.2013.6664092.

[20] Max Farias, Edna Barros, André Araújo, An approach for multi-task and multi-application mapping onto noc-based mpsoc, in: 2014 IEEE 57th International Midwest Symposium on Circuits and Systems, MWSCAS, 2014, pp. 205–208, http://dx.doi.org/10.1109/MWSCAS.2014.6908388.

[21] J. Soumya, K. Niranjan Babu, Santanu Chattopadhyay, Multi-application mapping onto a switch-based reconfigurable network-on-chip architecture, J. Circuits Syst. Comput. 26 (11) (2017) 1750174, http://dx.doi.org/10.1142/S0218126617501742.

[22] Fen Ge, Chenchen Cui, Fang Zhou, Ning Wu, A multi-phase based multi-application mapping approach for many-core networks-on-chip, Micromachines (ISSN: 2072-666X) 12 (6) (2021) http://dx.doi.org/10.3390/mi12060613.

[23] Qingkun Chen, Wenjin Huang, Yuanshan Zhang, Yihua Huang, An IP core mapping algorithm based on neural networks, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. PP (2020) 1–14, http://dx.doi.org/10.1109/TVLSI.2020.3033658.

[24] Qingkun Chen, Wenjin Huang, Yuze Peng, Yihua Huang, A reinforcement learning-based framework for solving the IP mapping problem, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 29 (9) (2021) 1638–1651, http://dx.doi.org/10.1109/TVLSI.2021.3097712.

[25] Jitesh Choudhary, Soumya J, Linga Reddy Cenkeramaddi, RAMAN: Reinforcement learning inspired algorithm for mapping applications onto mesh network-on-chip, in: 2021 ACM/IEEE International Workshop on System Level Interconnect Prediction, SLIP, 2021, pp. 52–58, http://dx.doi.org/10.1109/SLIP52707.2021.00019.

[26] Samala Jagadheesh, P. Veda Bhanu, Soumya J., Noc application mapping optimization using reinforcement learning, ACM Trans. Des. Autom. Electron. Syst. (ISSN: 1084-4309) 27 (6) (2022) http://dx.doi.org/10.1145/3510381.

[27] Yufan Zeng, Jiashan Tang, RLC-GNN: An improved deep architecture for spatial-based graph neural network with application to fraud detection, Appl. Sci. (ISSN: 2076-3417) 11 (12) (2021) http://dx.doi.org/10.3390/app11125656.

[28] Francisco Triviño, Francisco Alfaro, J.L. Sanchez, José Flich, Noc reconfiguration for CMP virtualization, in: Proceedings - 2011 IEEE International Symposium on Network Computing and Applications, NCA 2011, 2011, pp. 219–222, http://dx.doi.org/10.1109/NCA.2011.37.

[29] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, Jong Wook Kim, Q-learning algorithms: A comprehensive classification and applications, IEEE Access 7 (2019) 133653–133667, http://dx.doi.org/10.1109/ACCESS.2019.2941229.

[30] TechVidvan Team, Reinforcement Learning Algorithms and Applications, TechVidvan, 2021.

[31] Bo Yang, Thomas Canhao Xu, Tero Säntti, Juha Plosila, Tree-model based mapping for energy-efficient and low-latency network-on-chip, in: 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, 2010, pp. 189–192, http://dx.doi.org/10.1109/DDECS.2010.5491789.

[32] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, Kilian Q. Weinberger, Densely connected convolutional networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017, pp. 2261–2269, http://dx.doi.org/10.1109/CVPR.2017.243.

[33] Pradip Kumar Sahu, Santanu Chattopadhyay, A survey on application mapping strategies for network-on-chip design, J. Syst. Archit. (ISSN: 1383-7621) 59 (1) (2013) 60–76, http://dx.doi.org/10.1016/j.sysarc.2012.10.004.

[34] Robert P. Dick, David L. Rhodes, Wayne Wolf, TGFF: Task graphs for free, in: Proceedings of the 6th International Workshop on Hardware/Software Codesign, CODES/CASHE '98, IEEE Computer Society, USA, ISBN: 0818684429, 1998, pp. 97–101.

[35] J. Soumya, System-C-based-NoC-Simulator, URL https://github.com/Choudhary-Jitesh/NoC-Simulator.git.

[36] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, Kambiz Samadi, ORION 2.0: A power-area simulator for interconnection networks, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 20 (1) (2012) 191–196, http://dx.doi.org/10.1109/TVLSI.2010.2091686.

[37] P. Veda Bhanu, Rahul Govindan, Plava Kattamuri, J. Soumya, Linga Reddy Cenkeramaddi, Flexible spare core placement in torus topology based NoCs and its validation on an FPGA, IEEE Access 9 (2021) 45935–45954, http://dx.doi.org/10.1109/ACCESS.2021.3066537.