applied sciences

MDPI

Article

An Efficient Algorithm for Mapping Deep Learning Applications on the NoC Architecture

Zeeshan Ali Khan [1], Ubaid Abbasi [2] and Sung Won Kim [3,*]

[1] Department of Computer Science, National University of Computer and Emerging Sciences, Lahore 54770, Pakistan; zeeshanali.khan@nu.edu.pk
[2] Department of Computer Science, GPRC, Grand Priaire, AB T8V 4C4, Canada; uabbasi@gprc.ab.ca
[3] Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, Korea
* Correspondence: swon@yu.ac.kr

Abstract: Network-on-chip (NoC) is replacing the existing on-chip communication mechanism in the latest, very-large-scale integration (VLSI) systems because of their fault tolerant design. However, in addition to the design challenges, NoC systems require a mechanism for proper application mapping in order to produce maximum benefits in terms of application-level latency, platform energy consumption, and system throughput. Similarly, the neural-network (NN)-based artificial intelligence (AI) techniques for deep learning are gaining particular interest. These applications can be executed on a cloud-based system, but some of these applications have to be executed on private cloud to integrate the data privacy. Furthermore, the public cloud systems can also be made from these NoC platforms to have better application performance. Therefore, there is a need to optimally map these applications on existing NoC-based architectures. If the application is not properly mapped, then it can create a performance hazard that may lead to delay in calculations, increase in energy consumption, and decrease in the platform lifetime. Hence, the real-time applications requiring AI services can implement these algorithms in NoC-based architectures with better real-time performance. In this article, we propose a multilevel mapping of deep learning AI applications on the NoC architectures and show its results for the energy consumption, task distribution profile, latency, and throughput. The simulation is conducted using the OCTAVE, and the simulation results show that the performance of the proposed mapping technique is better than the direct mapping techniques.

Keywords: network-on-chip; artificial intelligence; neural networks; application mapping

check for updates

Citation: Khan, Z.A.; Abbasi, U.; Kim, S.W. An Efficient Algorithm for Mapping Deep Learning Applications on the NoC Architecture. *Appl. Sci.* **2022**, *12*, 3163. https://doi.org/10.3390/app12063163

Academic Editors: Anikó Costa and Remigiusz Wiśniewski

Received: 23 January 2022
Accepted: 18 March 2022
Published: 20 March 2022

1. Introduction

System-on-chip (SoC) has multiple communication links to meet the data transmission requirement of the platform [1]. The sub-micron design has a number of design challenges for these on-chip communication wires, hence the need for globally asynchronous, locally synchronous (GALS) systems arise. In this design strategy, the application tasks are divided into concurrent synchronous areas in a platform. This domain is only synchronous internally, and it would communicate with other locally synchronous areas in an asynchronous fashion [2]. The network-on-chip (NoC) provides an intra-chip communication paradigm for a particular topology design. Therefore, based on the GALS-based system, the communication can take place in a seamless manner. This also leads to a scalable and efficient design. Moreover, in many-core heterogeneous processors, the NoC is a scalable solution that can meet the application communication requirements [3].

A GALS-based SoC architecture needs to map various tasks of a target application onto different cores, which would help in obtaining better performance from this system. This is the foundation of the application mapping problem in the NoC architecture, and the solution can lead to a number of solutions. Therefore, it is pertinent to select an optimal solution for better NoC performance [4].

AI is the future of computing that can meet the data processing requirements for innovative applications. One such evolving artificial intelligence (AI) application is neural networks (NNs), which require a high level of parallelism to achieve the application processing deadline [5]. Different neurons of NNs can be mapped to different zones of the NoC architecture to exploit this parallelism.

Many researchers in the past have worked on various application techniques in multiple scenarios. In [6], the authors discuss the technique for mapping multiple applications on the NoC architecture. However, this does not consider artificial intelligence (AI) algorithm mapping on an NoC architecture. A multi-objective algorithm is proposed in [7] by considering the varying time constraints for the targeted applications. However, due to the reconfiguration overhead, the proposed approach did not produce promising results.

In [8], the authors propose a technique to locate optimal region for a particular application. Because of sequential nature, this did not result in significant outcomes. Another article [9] targeted a fault tolerant algorithm for application mapping by considering non-faulty cores for better core placements. Ref. [10] discusses an NoC application mapping to balance packet latency with other performance factors, while [11] discusses a mapping algorithm on heterogeneous multi-core processors having different features. In [12], a rectangle-analysis-based approach is proposed, which selects NoC regions for multi-application mapping using genetic algorithms. This design space exploration (DSE) mechanism tries to locate the best region for application execution based on latency and power consumption.

In this paper, we describe a method for mapping AI applications on an NoC platform in a dynamic run-time manner using a multilevel approach. We also give a comparison of the proposed multilevel technique with the direct mapping technique, which is the baseline for state of the art mapping methods. Moreoever, the abbreviations are listed in Table 1.

The article is structured as follows. In Section 2 we discuss related work about machine-learning-based algorithms, NoC platforms, and application mapping that would be used in this research. Thereafter, our proposed mechanism in explained in Section 3, and details about the simulation analysis are given in Section 4. Finally, we conclude the article in Section 5.

Table 1. List of abbreviations.

AI	Artificial intelligence
BB	Branch and bound
BEMAP	BB-based exact mapping
$B_{ta.tb}$	Bandwidth between two routers t_a and t_b
CC	Communication cost of the NoC
DNN	Deep neural network
DSE	Design space exploration
DVFS	Dynamic voltage and frequency scaling
E_{Link}	Link energy consumption
Lat_{avg}	Average latency
Lt_b	Latency of packet b
MET	Maximal empty triangle
n	Number of neurons
N	Number of processing cores
N_m	Manhattan distance from source to destination tile
NN	Neural network
NoC	Network-on-chip
N_x	Packets received by the core x
PSO	Particle swarm optimization
RL	Reinforcement learning
SNN	Spiking neural network
SoC	System-on-chip
SotAs	State-of-the-art
T_{sim}	Simulation time
VLSI	Very-large-scale integration

2. Related Work

There are a lot of machine learning applications that are used to process large datasets, and they require a lot of processing capabilities. In a modern computer system, NoC can provide an important communication infrastructure to implement these machine learning algorithms. In this section, a basic introduction about neural networks, NoC, and application mapping is given below.

2.1. Machine Learning, Deep Learning, and Neural Networks

Machine learning is an AI-based technique to analyze and generate a mathematical model from the input dataset [13]. Broadly speaking, the machine learning is used to process different datasets and it is normally classified into four categories, which include supervised, unsupervised, semi-supervised, and reinforcement learning. Supervised learning [14] is provided with the labeled data as an input to the training algorithm in order to develop a mathematical model, whereas an unsupervised learning algorithm [15] is not provisioned with the labeled data and it has to make a decision about the boundaries. If we have partial information about the labeled data, then, in that case, semi-supervised learning techniques [16] are useful. Finally, reinforcement learning (RL) [17,18] deploys an iterative mechanism to make a decision using an agent, which interacts with the external world.

Out of these techniques, an important approach is supervised learning and it deploys a neural network [19,20] to mimic the functionality of a brain by using multilayers to perform linear and nonlinear functions on the input dataset. This results in a better mathematical model, as it uses a layered architecture having input layer, inner layer(s), and output layer. An example of a neural network task graph is given in Figure 1. It can use a feedback path from output to the input layer for effectively generating the results; however, it requires a lot of computational power and it needs to be effectively deployed on a processing infrastructure to obtain the results in real time.

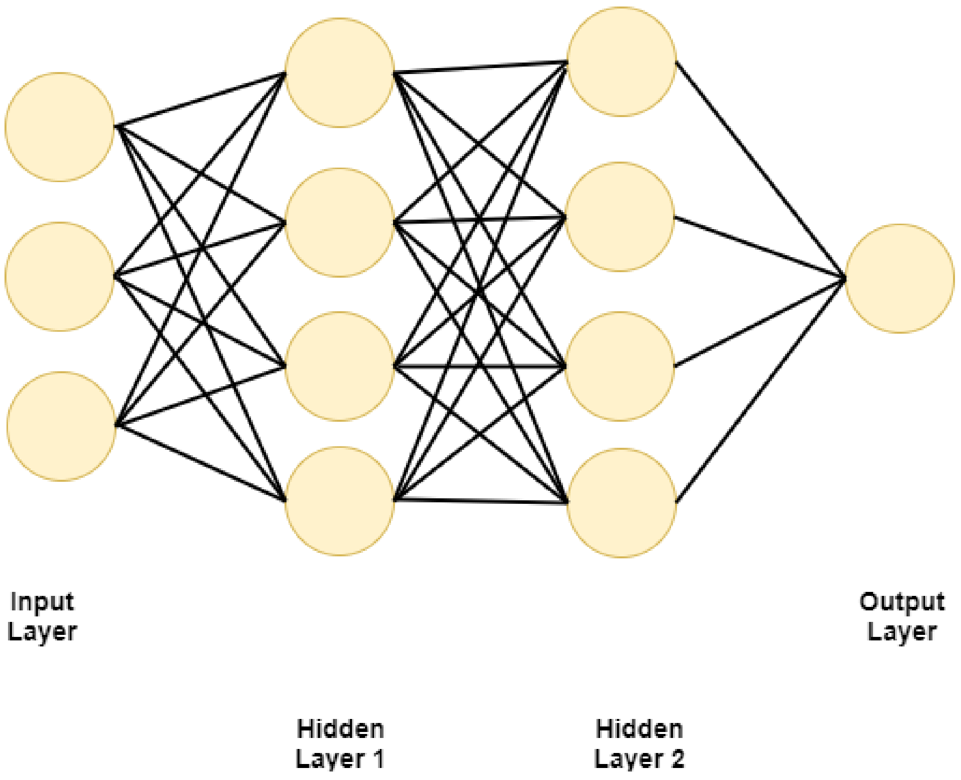

Figure 1. AI task graph as an input.

The neural network can be used to implement a number of applications using deep learning methods [21,22]. Deep learning comprises linear-regression-based classifiers and activation function, which is based on traditional linear regression technique. Deep learning has a neural network consisting of multiple neurons, whereas the linear regression has only one node. Furthermore, the deep learning approaches have multiple layers which process the input and forward it to the output [23]. Each layer can have multiple neural units for data processing. There is an input layer, an output layer, and multiple hidden layers for data processing. For classical machine learning classifiers, the hypothesis function has to be written by the user. On the other hand, the deep learning network generates this function itself.

There is another type of learning technique, which is called the shallow learning. The main difference between shallow and deep learning is that shallow learning can have a maximum of two layers [24].

Another type of multilayer neural network is convolution neural network, which has two dimensional planes, and each plane has neurons independently processing the data having sparse connections. This structure depends on the shared weight for data processing [25].

In the next section, we discuss network-on-chip (NoC) architecture, which can deploy these AI applications.

2.2. Network-on-Chip (NoC)

The network-on-chip describes a communication-centric processor architecture design. In an NoC architecture [26], there are multiple processing elements that are connected to the neighboring processing elements. The data generated by the processing elements are converted into fixed-length flow-control digits (flits), which are routed to the neighboring router using a routing algorithm. The flits have tail, head, and body bits which are routed towards the destination using the intermediate routers. The NoC architecture has many wires and routers, which are used for interconnection of various processing elements. The NoC router has input/output ports towards the north, east, south, west, and local processing elements. These ports help in interconnection with other ports of a nearby router using physical wires. Similar to an OSI layer router, this NoC router will forward the incoming packets towards the appropriate destination port using the routing mechanism defined for the platform. Therefore, there is an input buffer associated with each incoming port to store excessive traffic, and a crossbar switch to move a packet from an input port towards an output port.

The functionalities of an NoC can be categorized into multiple layers, including application, transport, network, data link, and physical layers. The application layer normally divides the target application into tasks that can be executed onto multiple processing cores. This step helps in energy optimization, latency reduction, and throughput improvement. This leads to the placement problem for the processing cores, and mapping problem for the application. The placement problem deals with the physical placement of heterogeneous processing cores on various geographical locations of the platform, whereas the mapping problem places various tasks of an application onto different cores for optimal performance [27].

The transport layer of the NoC deals with the issues of congestion control, flow control, and buffer overflow by devising end-to-end, as well as local, techniques. This layer helps in decreasing the congested areas within the platform, as these areas can form energy hot-spots. Uneven energy consumption can lead to reduction in processor lifetime and increase in application latency; therefore, it is an important design consideration. For this layer, NoC topology plays an important role. The placement of cores help in achieving the design goals of the NoC architecture. Based on the type of application, the placement of cores can be set in a way to reduce the congestion in the network [28].

The purpose of the network layer is to define the routing schemes of the packets. It is heavily dependent on the type of architecture, and it would help in achieving the main objectives of the NoC architecture. If the packet routing strategy is not carefully selected, then the energy consumption, latency, and throughput of the application would drastically reduce. There are various routing algorithms which can be used for the NoC design, which include XY, IX/Y, and XYX routing algorithms [29].

The data link layer is used to increase the reliability of the individual link. The error may arise in the packet transmission due to cross-talk, electromagnetic interference, and signal radiation. This layer would introduce mechanisms to reduce this error [30].

Finally, the physical layer is related to the actual transmission of the data on physical wires, and it is related to the microelectronic design of the platform. This deals with the capacity and design of the individual link, using the state-of-the-art technology [31].

In this article, the XY routing algorithm [32] is used for evaluation purposes. This algorithm forwards the packet in the x-direction until the packet's x-coordinates and those of the destination are same. Then, the packet is forwarded in the y-direction using the same algorithm, which will eventually arrive at the destination. In a router, there are five input and output ports. Out of these five ports, four are in the east, west, north, and south directions. The fifth port is in the direction of the local processing element. This router is responsible for forwarding the packet using the XY routing algorithm mentioned above. An example of such an architecture (16 × 16 NoC) is given in Figure 2.

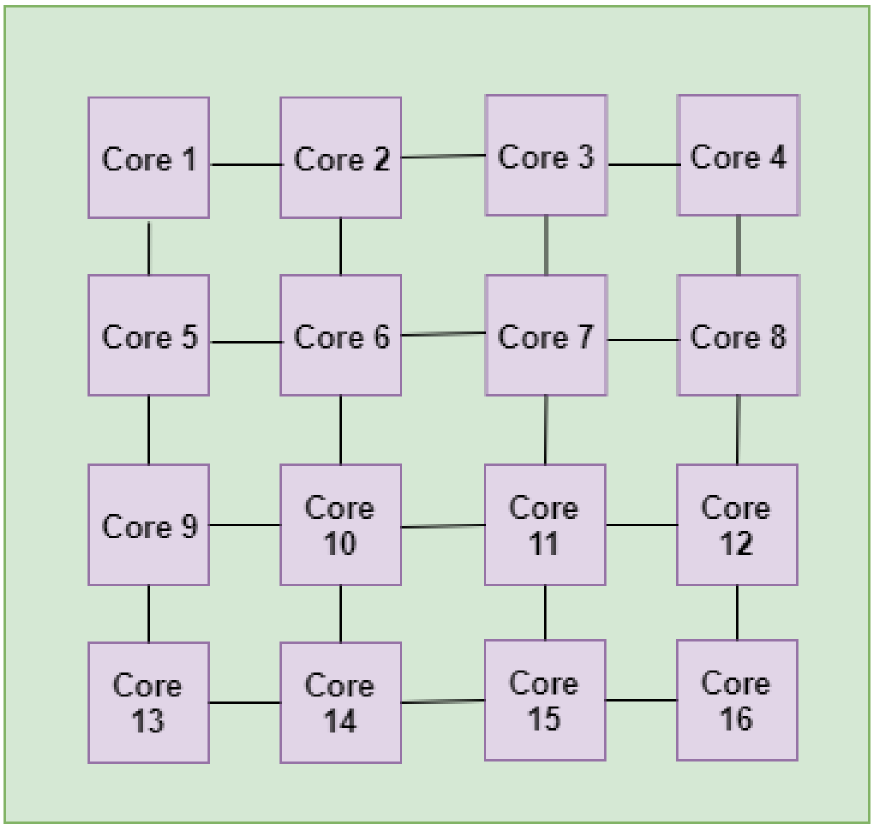

Figure 2. A 16 × 16 NoC architecture.

2.3. Application Mapping on the Network-on-Chip (NoC)

In this section, we discuss the mapping algorithms that exist for the NoC architectures. Mapping is a necessary part of the NoC design process, as various tasks of the application are assigned to different cores of the processor for execution. There can be various objectives that are selected based on the type of application. They include application latency, energy consumption, real-time deadlines, and throughput. Various optimization techniques have been used to implement these algorithms [33].

Ref. [34] explains a real-time application mapping on the NoC using the branch-and-bound (BB)-based exact mapping (BEMAP) algorithm. For a particular NoC bandwidth, the approach reduces the energy consumption and latency; however, it increases the throughput. Thus, the solution is related to the multi-objective optimization for NoC application mapping. The algorithm claims to have up to 19.93% reduction in energy consumption and 61.10% depletion in network latency for mesh and torus typologies. However, the article does not give an implementation for the AI-based applications.

Ref. [12] explains a way to map multiple applications on the NoC, which starts by performing an analysis on various NoC regions. Using a genetic algorithm, application tasks are mapped onto the potential regions and their performance is analyzed. In the second step, a B*tree-based simulated annealing algorithm is used to deduce updated mapping of the tasks. Using experimental results, the authors claim to have 23.45% reduction in power consumption and 24.42% reduction in the latency for the selected applications. However, the algorithm does not consider an AI as an input.

Ref. [11] proposes a precompute step for partially mapping the application, and final mapping at the run time for hard real-time applications. Thus, the applications have the possibility to adapt their mapping based on the processor resources. The authors claim to have 13% reduction in the energy consumption compared to the state-of-the-art approaches. However, the proposed approach can be extended for the AI-based input applications.

Ref. [10] describes multi-application mapping on the NoC under the latency constraints. The problem is solved using a heuristic-based algorithm, which is able to reduce the maximum average packet latency by 10.42%, with better overall performance. However, the AI-based applications are not mapped onto this platform using the proposed approach, which is an open area of research.

Ref. [9] is a fault-tolerant mapping approach for NoC platform. It has two steps; the first step maps the application core graphs to those processing cores that are fault-free, and in the second step, the algorithm utilizes the free cores identified in the first step. This helps in increasing the fault tolerance; however, it may increase the run-time overhead. Moreover, this technique may be utilized for an AI-based application core graph.

Ref. [8] is another multi-application mapping technique, which deploys two steps. In the first step, the technique first finds a region for a particular application. The second step maps the tasks of a particular application onto the cores available in that region. The maximal empty rectangle (MER) technique is used to find an optimal region for each application, and a tree-model-based algorithm is utilized for the second step of the task mapping. The authors claim to have an 18% reduction in latency and energy consumption. However, we think that this idea can be enhanced for recently emerging AI applications.

Ref. [7] is a multi-application mapping technique using an evolutionary approach, and the target is to optimize latency and energy consumption. The technique is compared with the branch-and-bound and genetic algorithms to test its usability. However, this technique can be enhanced for the neural-network-based AI applications.

Ref. [6] presents an application mapping approach for each individual use-case by exploiting the dynamic voltage and frequency scaling (DVFS). This results in power saving; however, other performance factors needs to be explored as well. This work can be extended for the AI applications using the DVFS technique for obtaining power optimization.

Ref. [35] describes a comparison of NoC application mapping for optimizing communication, energy consumption, and delay for VOPD and MPEG4 applications. The main focus is about exploring the application mapping for new design technologies, but it does not focus on the AI-based applications.

An interesting approach is discussed in [36], where the authors discuss a mapping algorithm for decreasing latency and energy consumption in the hardware architecture. The application considered is spiking neuron networks (SNN), which have to be mapped on many-core neuromorphic hardware. However, this technique needs to be improved and it is an example of direct application mapping. For the sake of comparison, we have considered this type of direct technique in this article.

Another recent article, ZigZag [37] focuses on exploring the design space for the deep neural networks (DNNs) by focusing on the memory hierarchy design space. They claim to have a 64% better performance than state-of-the-art solutions, but this work can be further improved by focusing on the communication aspect.

Ref. [38] proposes a two step technique that is based on the genetic algorithm for mapping applications onto the NoC architecture. The proposed approach has a better result than the traditional genetic algorithm in terms of delay and power consumption; however, it does not consider AI-based applications.

A comparison of the articles mentioned in the literature review is given in Table 2. Out of all these techniques, we found only a few articles that talk about mapping deep neural networks on the processing cores that are connected by the NoC. Therefore, our main focus in this article is to present a mapping strategy for the deep neural networks and explore this research domain.

Table 2. Comparison of the recent work.

Article	Mapping Technique	Performance Improvement	AI Application Mapping
[6]	DVFS-based application mapping	Large power savings	✗
[8]	Multi-application mapping	18% reduction in latency and energy consumption	✗
[9]	Fault-tolerant mapping	9.5% communication energy reductions and 7.94% performance improvement	✗
[10]	Heuristic-based algorithm	Reduction in the maximum average packet latency by 10.42%	✗
[11]	Run-time mapping for hard real-time applications	13% reduction in the energy consumption	✗
[12]	B*tree-based simulated annealing algorithm/genetic algorithm	23.45% reduction in power consumption and 24.42% reduction in the latency	✗
[34]	Branch-bound (BB)-based exact mapping (BEMAP) algorithm	19.93% reduction in energy consumption and 61.10% depletion in network latency	✗
[35]	Comparison of most of the reported application mapping techniques for NoC	Conclusion is provided for NoC application mapping-based on algorithm run-time	✗
[36]	Particle swarm optimization (PSO) algorithm and TABU search	Reduction in average latency by 63% and average energy consumption by 69%	✓
[37]	Combining uneven and search mapping strategies	Up to 64% more energy-efficient in comparison with SotAs	✓
[38]	Multilevel genetic algorithm based technique	Reduction in power consumption and delay in comparison with traditional genetic algorithm	✗

3. Multilevel Task Mapping for NN Applications

In this section, the main objective is about exploring the AI-based application mapping on various NoC platforms by considering latency and energy consumption. The technique used in this article is depicted in Algorithm 1. There are multiple phases of this algorithm. In the first phase (region analysis), we select particular region(s) where the various tasks of an application can be deployed based on core availability. In the next phase, the neurons of that task are mapped onto the processor cores of the region selected in phase one, and hence the complete application is mapped onto the processor architecture. To the best of our knowledge, the task mapping of AI applications on NoC architecture is rarely studied and we shall consider a mesh-based NoC for application mapping to evaluate this solution. However, the prospective research will focus on exploring NoC architectures other than mesh-based structures.

The algorithm first partitions the application into tasks, which are respective layers of the neural network, and selects a particular region for its mapping. It also looks into the processor cores which are available at the moment. In the second step, the neurons are mapped onto the particular cores to enhance the performance in terms of latency and energy consumption. In the following sub-sections, we discuss the two steps which are required for realizing these objectives.

Algorithm 1: AI application mapping on mesh-based NoC.

```
1 Input: AI application task graph, a mesh-based NoC architecture
2 Output: Application mapping on targeted architecture
3 Analyze which cores are free at the moment
4 Locate the potential region based on core availability
5 Level 1 Mapping: Layers of neural network are mapped on particular regions
6 Level 2 Mapping: Each neuron of a layer are mapped onto a particular
   processing core
```

3.1. Level 1 Mapping: Region Mapping

First of all, we divide the mesh-based NoC into multiple regions, and this will help in identifying the available regions. The regions are selected based on the number of neural network layers to be mapped on the processor architecture. The number of regions are obtained as follows:

$$n_r = \frac{n_c}{n_{NN}} \tag{1}$$

where n_r is the number of regions, n_c is the number of cores, and n_{NN} is the number of neural network layers.

We select closely located cores for one region based on the region count obtained by the abovementioned formula. If any or multiple cores are available after this process, then they are merged into a nearby region. In the next step, we identify the regions which are available at the moment by considering a predefined occupancy threshold. This is to select regions that are not occupied at the moment. This makes sure that only those regions should be selected that are not highly busy; rather, we focus on regions having lesser occupancy. This would have a positive impact on energy consumption profile of the processor. Algorithm 2 explains the details of this technique. The proposed algorithm is inspired from [39], which tries to optimize the global objectives based on a game theoretic approach. This process is explained in Figure 3. This figure shows that we are selecting a particular region for mapping various closely related tasks of the considered NN application. The tasks are the neurons of a particular neural network layer, and these tasks have to be mapped on a particular region. This technique will make sure that the tasks of a particular layer are processed first, followed by the next one. The technique will try to ensure that the communication latency is minimized at this level of mapping.

3.2. Level 2 Mapping: Neurons Mapping on the Cores

In the second step, we map the neurons to a particular processor core based on the assigned NoC region in level-1 mapping. Our goal in this step is to select a particular core to minimize the energy consumption and communication latency of that region. This process is explained in Algorithm 3. The idea of the proposed algorithm is obtained from [39], which tries to optimize the global objectives based on a game theoretic approach. This will have a positive impact as the hot-spots in the NoC are minimized, leading to a better solution, which is explained in Figure 4. In this figure, we show that the neurons of a task are assigned to a particular processing element to reduce the communication energy consumption, which is a major source of energy depletion in the NoC. Furthermore, we also have to evenly distribute the neurons on all the available cores to have an even energy consumption profile. If the neurons are assigned to a single processor, then it would reduce the communication energy but it would create the energy hot-spots. On the other hand, if the neurons are evenly distributed, then it would increase the communication energy but it would minimize the hot-spots in the energy consumption. This will eventually increase the lifetime of the processor.

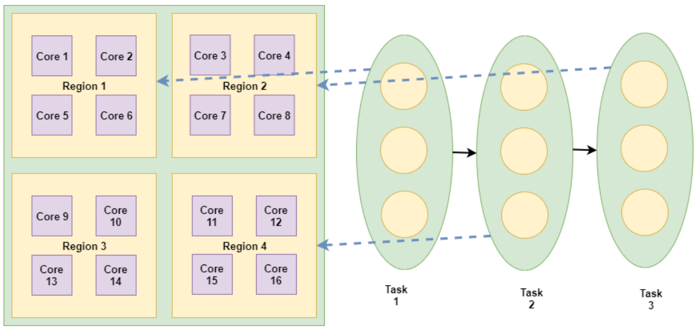

Figure 3. AI sub-task mapping on various NoC regions.

Algorithm 2: Neural-network-level mapping on the NoC region.

```
1: Input: Tasks of a Neural Network level and NoC regions
2: Output: Mapping of input on a particular NoC region
3: Analyze occupancy level of a region
4: Create a list of NoC regions in ascending order of their occupancy
5: List_of_Allowed_Regions = Least occupied NoC region
6: Current_Neural_Network_Layer = First Layer of the input Neural Network
7: Current_Neural_Network_Mapping = Nil
8: while All the regions not processed do
9:     Map the Current_Neural_Network_Layer on least occupied region available in
       List_of_Allowed_Regions
10:    Remove the selected region from the List_of_Allowed_Regions
11:    Current_Neural_Network_Layer = Next Layer of the Neural Network
12:    List_of_Allowed_Regions = All NoC regions that lie in the neighborhood of the
       selected NoC region for the last Neural Network Layer
13: end while
```

3.3. Discussion about the Proposed Technique

The proposed technique aims to reduce the communication energy consumption and latency in the NoC architecture, as they are a major source of battery depletion. The proposed solution is based on game-theory (inspired from [39])-based mechanism and it tries to choose the best core for the neuron execution that would lead to energy consumption reduction. The proposed technique is compared with direct mapping techniques which are used in most of the related research work described in the introduction. These techniques are not aimed at deploying the neural network applications, which have an inherent parallelism. Furthermore, in these techniques, the multilevel approach is not used for a single application, as each application does not have an inherent parallelism. Therefore, our multilevel technique should perform better than direct mapping techniques for NN applications. This is also illustrated in the simulation analysis that is presented in the next section.

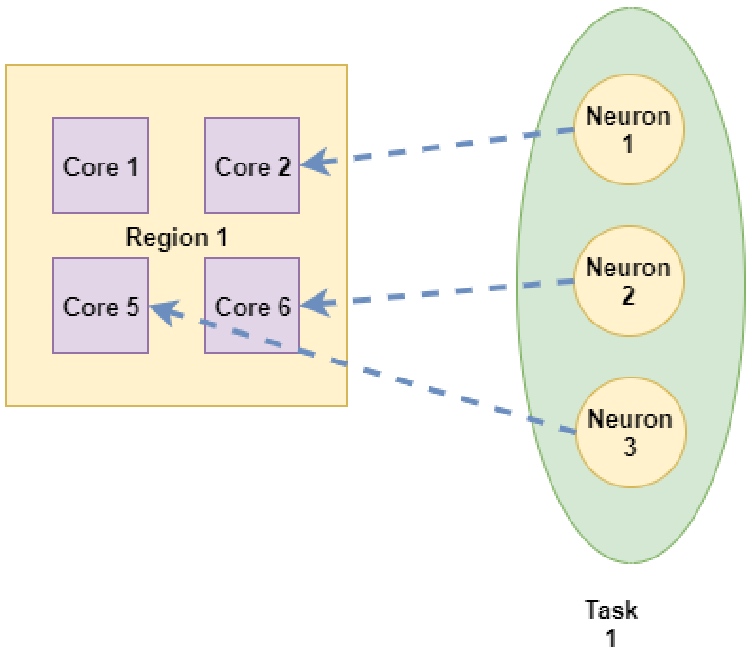

Figure 4. AI task mapping on processor cores.

Algorithm 3: Neuron mapping on the NoC core.

```
1:  Input: Neurons of a Neural Network layer, and selected NoC region
2:  Output: Mapping of neurons on a particular NoC core
3:  Analyze the current occupancy of the NoC cores
4:  Create a list of processor cores in ascending order of their occupancy
5:  List_of_Allowed_Cores = Least occupied NoC core
6:  Current_Neuron = First Neuron of the Neural Network layer
7:  Current_Neuron_Mapping = Nil
8:  while All the neurons not processed do
9:      Map the Current_Neuron on least occupied core available in List_of_
        Allowed_Cores and update the occupancy level of each core
10:     Current_Neuron = Next Neuron of the Neural Network layer
11:     List_of_Allowed_Cores = Selected NoC core and all NoC cores that lie in the
        neighborhood of the selected core for the last Neuron
12: end while
```

4. Evaluation

In this section, the results obtained from simulation analysis are discussed. The main task is to develop an in-house simulation that can measure performance evaluation of the algorithms. The considered NoC has a varying size, and the AI application is shown in Figure 1. In this section, first we explain the analytical model followed by the simulation results.

4.1. Analytical Model

Inspired from [34], in the following section, the energy and latency model is described. As per the bit energy model, the total communication energy consumption *CC* is given by the following formula:

$$CC = \sum_{a,b}^{n} [B_{ta.tb} \times (N_m \times E_{Switch} + N_{m-1} \times E_{Link})] \tag{2}$$

In this equation, the $B_{ta.tb}$ parameter indicates the bandwidth between two routers t_a and t_b, E_{Switch} is the switch energy consumption, and E_{Link} is the energy consumption by the link of the NoC element. n gives us the count of neurons or nodes in the considered AI application which need to be mapped, and N_m represents the Manhattan distance between source tile (x_a, y_a) and the destination tile (x_b, y_b).

Its formula is as follows:

$$N_m = |x_a - y_a| + |x_b - y_b| \tag{3}$$

In order to compute the communication cost per bit, we divide CC by total number of bits communicated during the simulation.

To obtain an efficient solution, CC has to be optimized using the techniques proposed in this article. The average value of latency can be calculated from the following equation:

$$Lat_{avg} = \frac{1}{N} \sum_{x=1}^{N} \frac{1}{N_x} \sum_{y=1}^{N} Lt_b \tag{4}$$

Here, latency of packet b is given by Lt_b, N_x gives the packets received by the core x, and number of processor cores in the platform are given by N. Furthermore, the average throughput of the network is given by

$$Throughput_{avg} = \frac{1}{N \times T_{sim}} \sum_{x=1}^{N} N_x \tag{5}$$

where T_{sim} is the simulation time. Finally, the average hop distance is defined as follows:

$$HopDistance_{avg} = \frac{n_{flits}}{n_{links}} \tag{6}$$

where n_{flits} indicates the number of flits flowing in the NoC architecture, and n_{links} is the total number of NoC links.

The simulation parameters for this analysis are given in Table 3. The simulation is conducted using OCTAVE [40], which is a freely available scientific programming language. Using the equations mentioned above, we have generated the results for a 2D mesh with varying sizes, from 4 till 16 cores. The considered application is a neural network graph with four layers, comprising input, output, and two hidden layers. For the sake of comparison, we have used the direct mapping that is the state-of-the-art mapping technique for most of the articles mentioned in the related work.

Table 3. List of parameters.

Parameter	**Value**
NoC type	2D Mesh
NoC sizes	$2 \times 2, 3 \times 3, 4 \times 4$
Embedded applications	Artificial intelligence (neural network)
Packet length	128 bits (1 flit)
Mapping algorithm	Multilevel and direct mapping
Simulation time	1000 s
Clock frequency	2000 MHz

4.2. Simulation Results

In this section, the simulation results are described using the techniques mentioned in the previous section. We simulated the proposed approaches for three different types of NoC architectures having simulation parameters depicted in Table 3. In the following sections, we discuss the result for three types of NoC configurations.

4.2.1. Visual Analysis of Application Mapping

The result of direct mapping on a four-cores architecture is shown in Figure 5, while our proposed technique has two steps results, which are shown in Figures 6 and 7. Here, you can observe that our proposed approach has mapped the AI neurons based on their task activity and this helps in optimizing the overall energy consumption of the network as well. On the other hand, the direct mapping technique is relatively more homogeneous, and it does not allocate neurons according to the task load of a particular level. Hence, we observe a higher energy consumption profile for such a network. Nevertheless, visual analysis reveals that the performance of both the techniques would result in similar output. However, as we move towards a higher NoC configuration, our proposed technique would homogeneously map all the tasks of our application onto the processor.

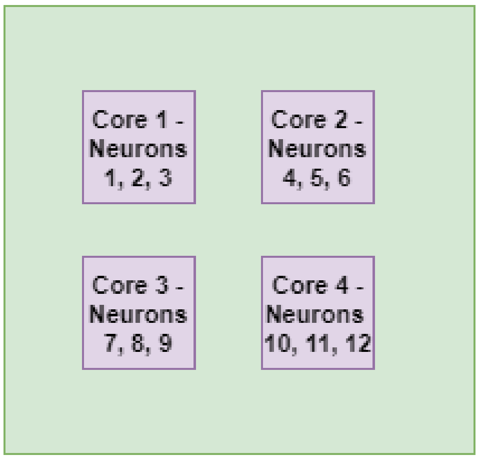

Figure 5. Direct mapping for four-cores architecture.

For a processor comprising nine cores, we observe that after level-1 mapping of application on the cores, the multiple levels are homogeneously mapped on the cores (see Figure 8). However, after the level-2 mapping (see Figure 9), the neurons are mapped in a much better way and the communication energy consumption will be reduced in this case in comparison with the direct mapping, as shown in Figure 10. Visual analysis reveals that both the techniques may have similar results. However, our technique tries to optimize the communication energy of such an NoC architecture by closely mapping the tasks of a certain stage. This would be explained in the energy consumption analysis for both these techniques.

Finally, direct AI application mapping on a 16-cores architecture depicts that some cores are vacant and they did not receive any neurons for processing (see Figure 11). On the other hand, after the two levels of application mapping, our proposed approach is able to distribute the neurons on all the processor cores, which would help to reduce the energy hot-spots (see Figures 12 and 13). Now, for this case, the visual analysis tells us that most of the processor cores are utilized. This may increase the communication energy consumption, but it would give us a better energy consumption profile by evenly distributing the energy. We would discuss this point in the context of overall system-level energy consumption.

4.2.2. Energy Consumption, Communication Latency, and Throughput Analysis

In order to compare the two techniques, we used the analytical model described previously. For the forthcoming results, we mention the size of the NoC in the x-axis. The number indicates a particular type of NoC, as per the following list:

- 1 indicates four-cores architecture.
- 2 indicates nine-cores architecture.
- 3 indicates 16-cores architecture.

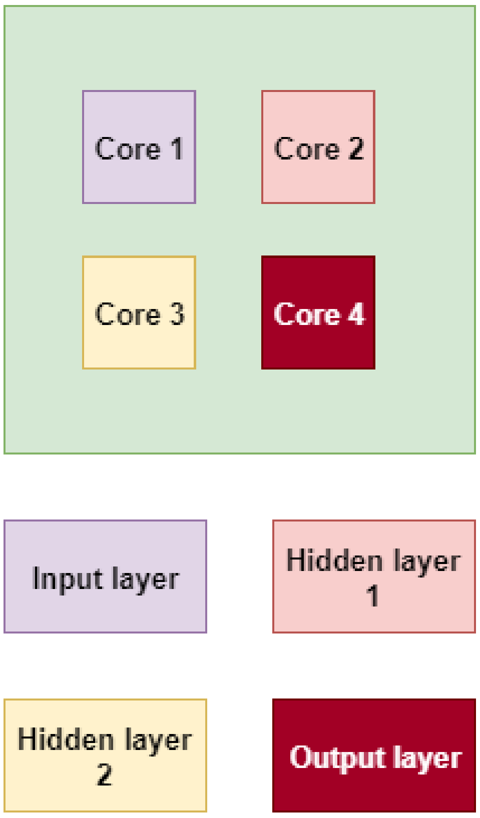

Figure 6. Level-1 mapping for four-cores architecture using multilevel mapping technique.

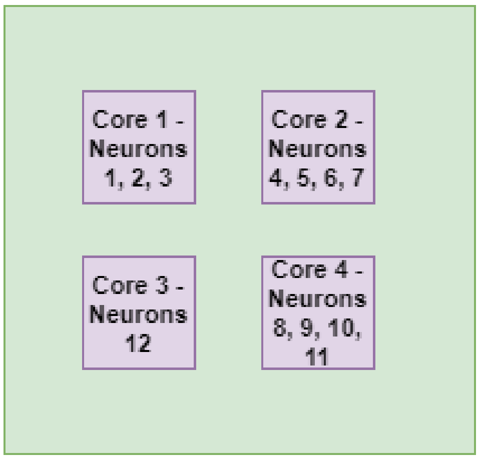

Figure 7. Level-2 mapping for four-cores architecture using multilevel mapping technique.

Figure 14 describes the comparison of communication cost between different configurations. For configuration 1, the direct mapping results in lesser communication cost; however, for other configurations the communication cost is decreased for our proposed multilevel mapping. This is because the proposed multilevel mapping approach tries to decrease the communication overhead and it results in less communication cost if the number of cores are greater. Therefore, it helps in better exploiting the parallelism that is available in the processing architecture. On the other hand, Figure 15 shows per-bit communication cost and it is almost the same for any technique, as it depends on the underlying hardware architecture and bandwidth, which is the same for all the cases.

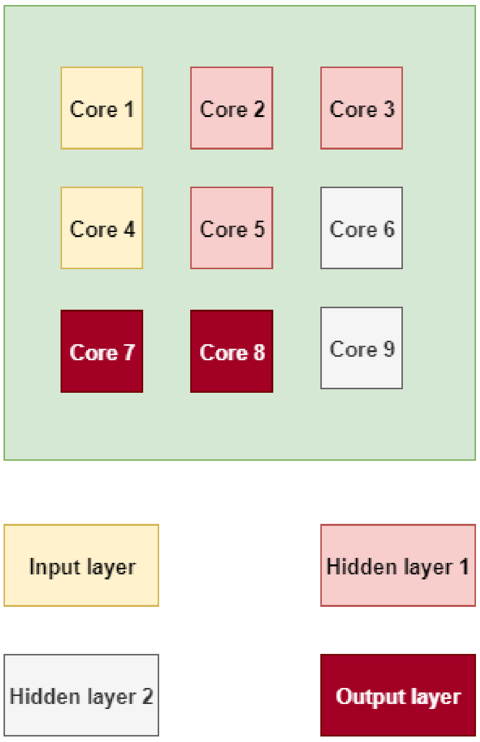

Figure 8. Level-1 mapping for nine-cores architecture using multilevel mapping technique.

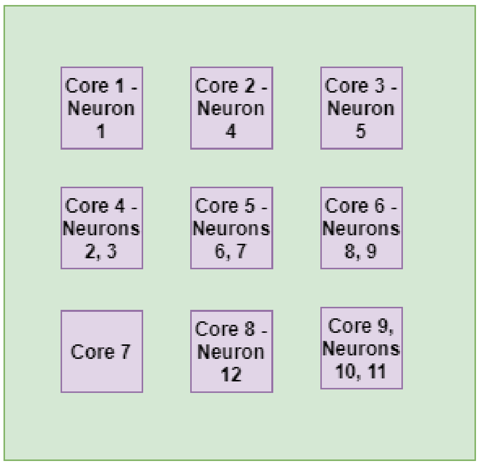

Figure 9. Level-2 mapping for nine-cores architecture using multilevel mapping technique.

Hence, configuration 1 corresponds to 2*2-sized NoC, configuration 2 is a 3*3-sized NoC, and configuration 3 is a 4*4-sized NoC.

As per the visual analysis described previously, our technique gives a better energy consumption profile by evenly distributing the neurons when the number of cores are increasing. We observe the same result for the communication energy of the NoC architecture. As the NoC size is increasing, the proposed technique is giving better results. Therefore, the proposed technique is good for platforms having large number of cores in terms of communication energy consumption and even task placement.

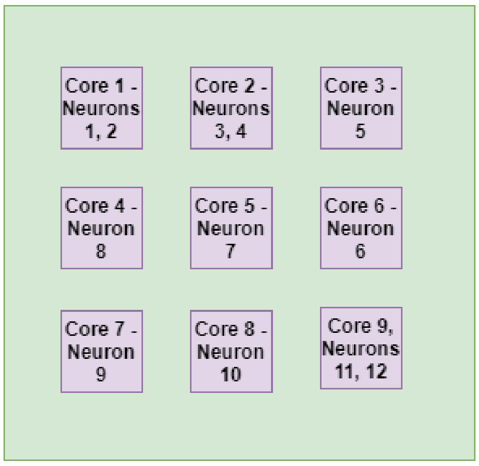

Figure 10. Direct mapping for nine-cores architecture.

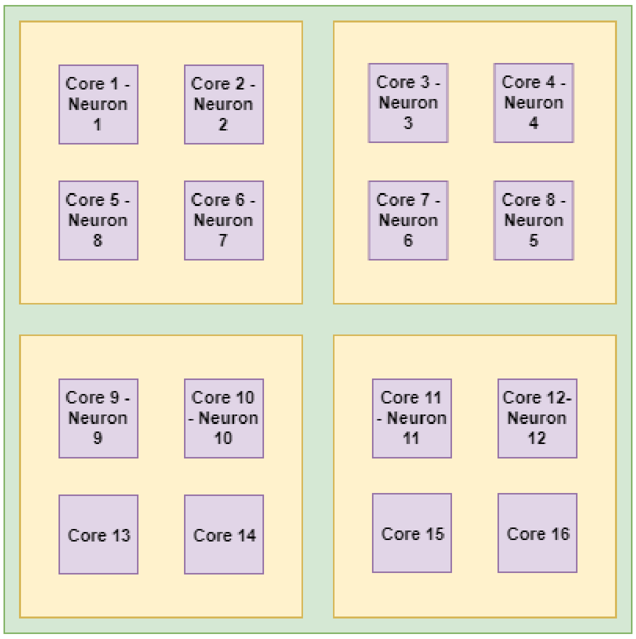

Figure 11. Direct mapping for 16-cores architecture.

Then, Figure 16 describes the average communication latency for both the schemes. This shows that the communication latency starts to decrease for larger number of processing cores using the proposed multilevel mapping, as this technique tries to minimize the communication overhead by exploiting the parallelism. As it first selects the similar level of AI application to a particular region, and then it assigns the respective tasks to a processing core, this helps in minimizing the communication latency, and the overall performance of the system is also improved.

Figure 17 discusses the average throughput of the proposed techniques and it shows that for higher number of processing cores, the system throughput is better for multilevel mapping. This also implies that the performance of multilevel technique is better for higher number of cores as it tries to exploit the parallelism in the system. If there are fewer cores, then there are not a lot of processing elements available and, as a result, the multilevel technique does not perform very well. However, the performance in comparison with the direct technique is not very promising. In our opinion, it is because of the fact that the multilevel technique tries to evenly distribute the neurons on the entire platform and it increases the communication costs. This would help us achieve both the objectives at the same time.

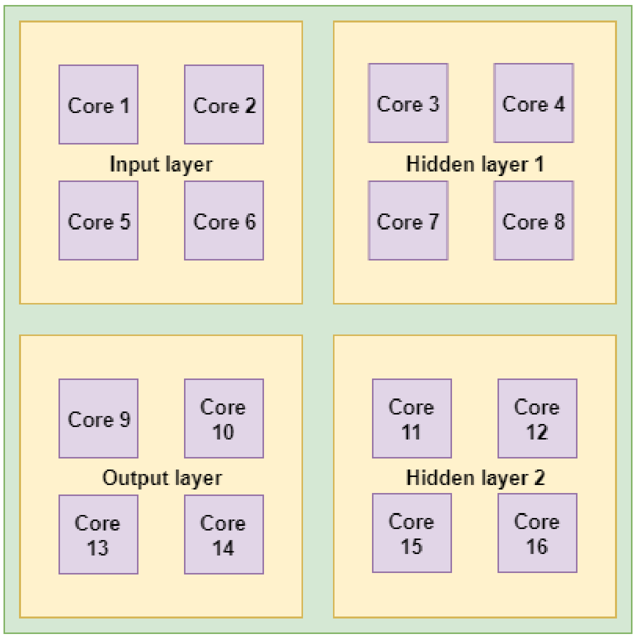

Figure 12. Level-1 mapping for 16-cores architecture using multilevel mapping technique.

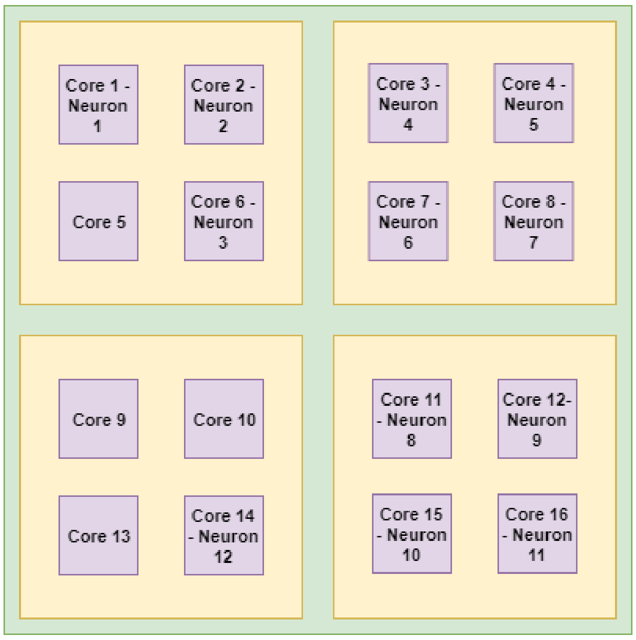

Figure 13. Level-2 mapping for 16-cores architecture using multilevel mapping technique.

Figure 18 gives an idea about the number of hops traversed by neurons in the NoC during the simulation. If this value is decreasing, then it has a positive impact on the communication energy consumption. The results indicate that the communication in the NoC is increasing as a result of increasing the number of processing cores. Furthermore, for a smaller number of cores in the NoC, the multilevel mapping is not producing better results. However, if the NoC size is increased, then its performance improves.

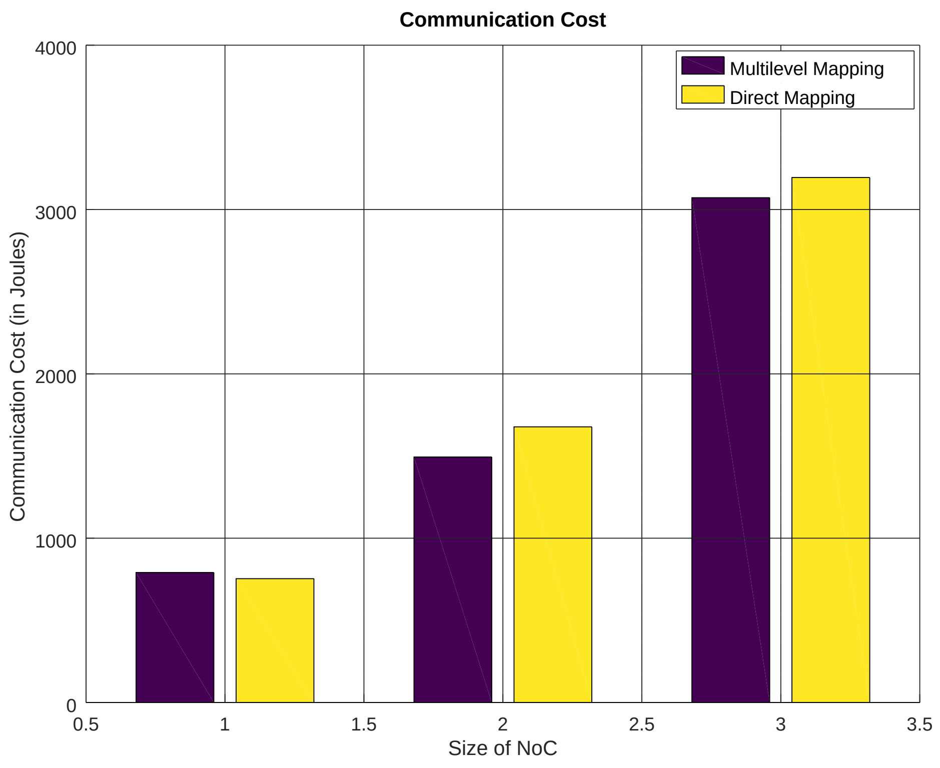

Figure 14. Communication cost.

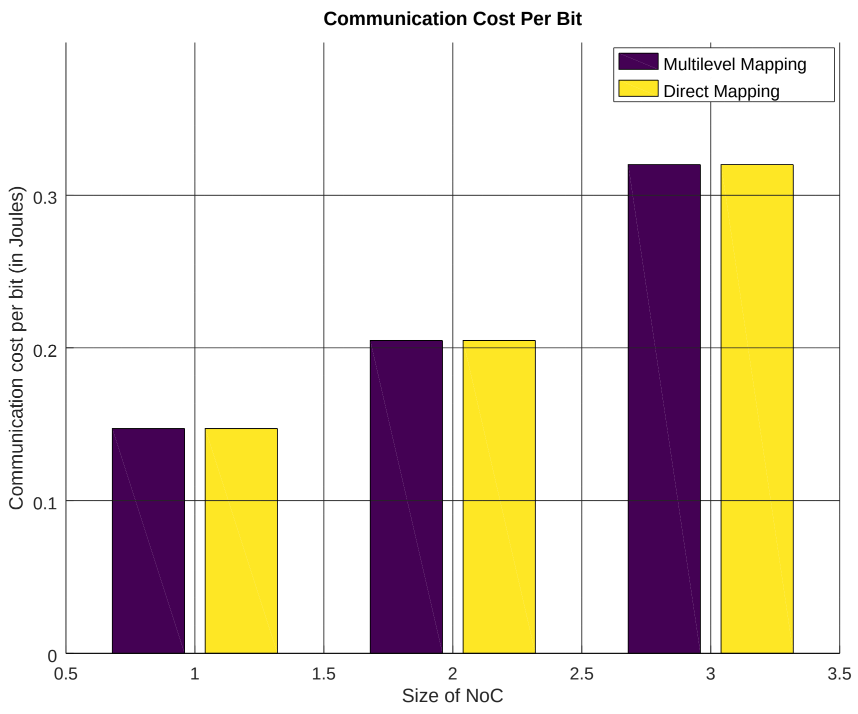

Figure 15. Communication cost per bit.

Finally, Figure 19 shows the average hop distance that is traversed by data that are flowing from one neuron to the next one. If this value is decreasing, then it indicates that we have less communication per communication link, and the communication energy consumption is decreasing. One observation is that for a larger number of NoC cores, the average value of hop distance is decreasing. This is because the number of communication links in the architecture are increasing. However, for fewer processing cores, the direct mapping is giving good results in terms of the average hop distance, but for a larger NoC architecture, this value is decreasing. Hence, the multilevel solution is better from this prospective.

We can conclude that the proposed multilevel technique is better than direct mapping techniques for neural-networks-based AI applications if the number of cores is a large number. We may stick to the direct technique for a platform with smaller number of processing cores.

4.3. Discussion about the Results and Prospective Work

In this section, we discuss the results of the experiment. As described earlier, we compared the proposed multilevel approach with the direct approach. The direct approach is relatively straightforward but, on the other hand, the complexity of the multilevel approach is around twice that of the direct one. This is because the multilevel approach has two stages, which are similar to two direct approaches, but the increase in complexity gives us better selection of the cores to reduce the overall energy consumption of the platform. However, it enhances the complexity of the algorithm. In addition to that, this technique is better in terms of online performance. If the neural network is to be executed alongside other applications, then it would be better to use an online technique that takes into account the number of cores that are currently available. Furthermore, the multilevel approach has a better average latency in comparison with the direct mapping as it first analyzes the availability of cores, and then it would select any particular region to map a layer of neural network. On the other hand, the direct approach does not take into consideration the number of available cores, and this is reflected in the results for the average latency. Finally, the multilevel approach tries to reduce the communication energy consumption and it results in less communication overhead. This leads to a lower average throughput value for the proposed technique.

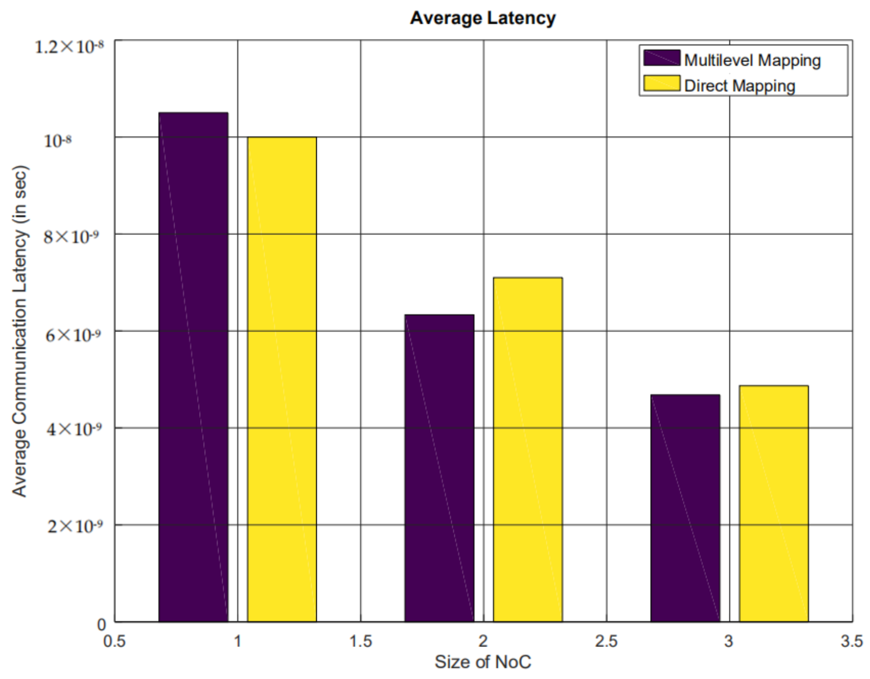

Figure 16. Average latency.

The results are improving when the size of the NoC architecture is increasing. In a number of recent similar processor architectures, it is observed that the number of processing cores is very large. Hence, we can conclude that the performance of the proposed technique would be better in a real-world scenario.

Furthermore, the proposed technique is tested using an analytical model that is inspired by [34]. Moreover, in order to perform the initial evaluation, we only used the XY routing. However, we propose that in the prospective works, the approach can be simulated using a network-on-chip simulation such as [41]. Using this simulator, we can realize a better level of abstraction by including multiple routing protocols, different flow control algorithms, and changing buffer depths. For the current task, we wanted to obtain preliminary results for a higher level of abstraction; that is why an OCTAVE-based simulation is used for obtaining the results. Furthermore, a four-layer neural network is considered for the evaluation purpose in this article. In the future work, we may enhance the complexity by adding more layers in the neural network. This will increase the confidence of a reader in the findings of the research.

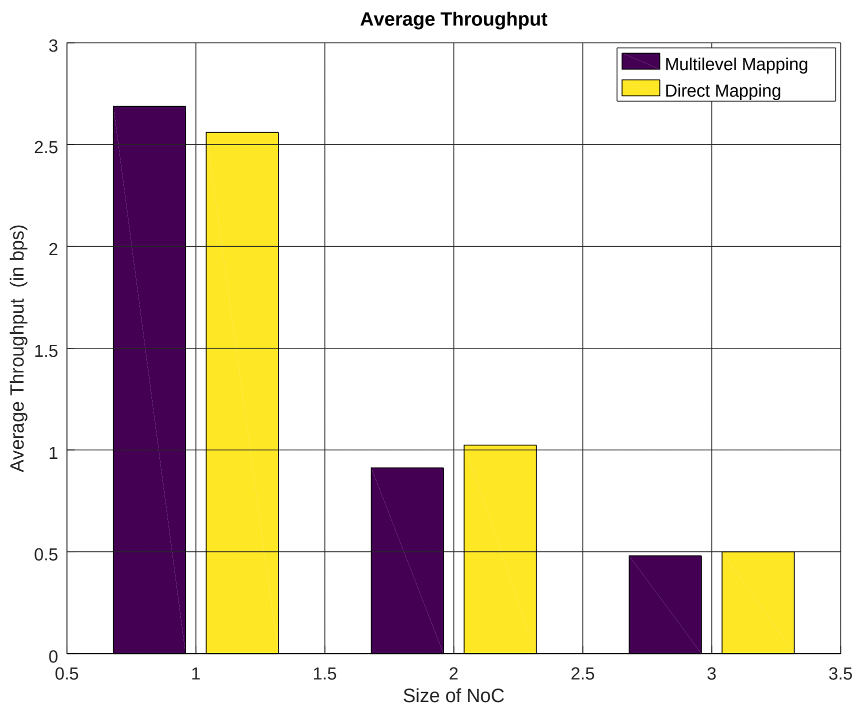

Figure 17. Average throughput.

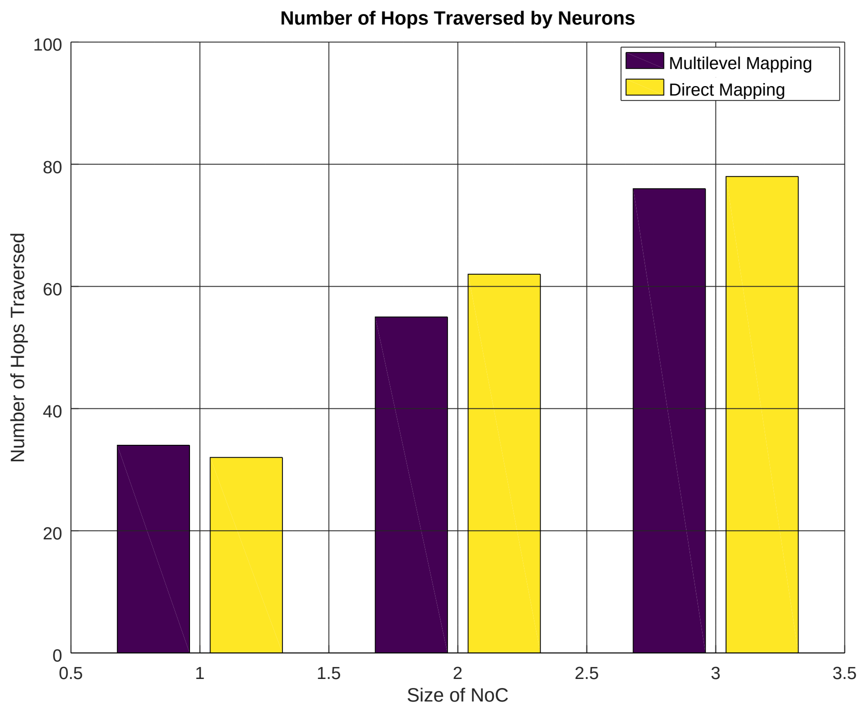

Figure 18. Number of hops traversed by neurons.

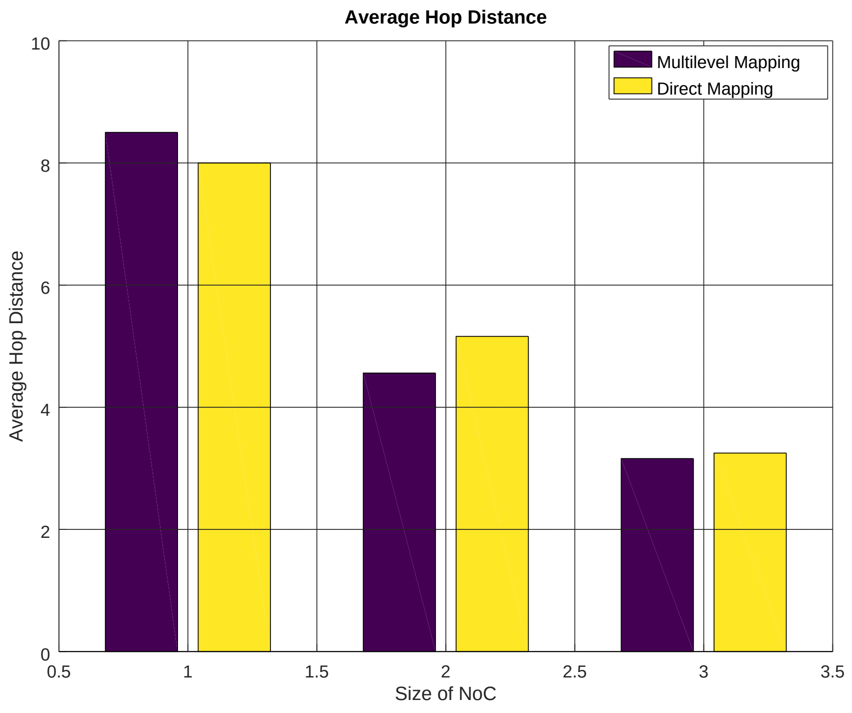

Figure 19. Average hop distance.

5. Conclusions

There are a lot of emerging AI applications that can process huge amounts of training data for numerous applications. In the current scientific era, neural networks are emerging as a deep learning application that can solve a number of problems in various domains. These can be solved using a cloud computing service, but sometimes we need to perform these calculations on a single platform due to the data privacy requirements. Here, we need to have an NoC platform that can efficiently help us in solving the problem. However, its performance depends on the underlying architecture. Hence, it is required to optimize the performance of the system by optimally mapping the application tasks to the NoC processing elements. Therefore, for the case of NoC architecture, we have to optimally map the AI application on various processing cores such that the communication performance and energy consumption profile of the platform can be improved.

In this article, we proposed a multilevel application mapping technique for AI applications. The proposed technique is compared with the state-of-the-art and simulated using an OCTAVE program. The simulation results show that it helps in optimizing the system parameter values such as throughput, latency, and energy consumption. The proposed technique can better optimize the NoC-based system parameters. For prospective work, we propose to consider NoC architectures other than mesh-based technology for neural network applications. Furthermore, this work can be evaluated on a real platform in the future works.

Author Contributions: Conceptualization, Z.A.K. and U.A.; methodology, Z.A.K., U.A. and S.W.K.; investigation, Z.A.K. and U.A.; resources, Z.A.K., U.A. and S.W.K.; writing—original draft preparation, Z.A.K. and U.A.; writing—review and editing, Z.A.K., U.A. and S.W.K.; supervision, S.W.K.; funding acquisition, S.W.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF-2021R1A6A1A03039493) and in part by the NRF grant funded by the Korea government (MSIT) (NRF-2022R1A2C1004401).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Marcon, C.; Calazans, N.; Moraes, F.; Susin, A.; Reis, I.; Hessel, F. Exploring NoC mapping strategies: An energy and timing aware technique. In Proceedings of the Design, Automation and Test in Europe, Munich, Germany, 7–11 March 2005; pp. 502–507.
2. Teehan, P.; Greenstreet, M.; Lemieux, G. A survey and taxonomy of GALS design styles. *IEEE Des. Test Comput.* **2007**, *24*, 418–428. [CrossRef]
3. Charles, S.; Mishra, P. A Survey of Network-on-Chip Security Attacks and Countermeasures. *ACM Comput. Surv.* **2021**, *54*, 1–36. [CrossRef]
4. Mohiz, M.J.; Baloch, N.K.; Hussain, F.; Saleem, S.; Zikria, Y.B.; Yu, H. Application Mapping Using Cuckoo Search Optimization With Lévy Flight for NoC-Based System. *IEEE Access* **2021**, *9*, 141778–141789. [CrossRef]
5. Takai, Y.; Sannai, A.; Cordonnier, M. On the number of linear functions composing deep neural network: Towards a refined definition of neural networks complexity. In Proceedings of the International Conference on Artificial Intelligence and Statistics, Online, 13–15 April 2021; pp. 3799–3807.
6. Murali, S.; Coenen, M.; Radulescu, A.; Goossens, K.; De Micheli, G. Mapping and Configuration Methods for Multi-Use-Case Networks on Chips. In Proceedings of the 2006 Asia and South Pacific Design Automation Conference, Yokohama, Japan, 24–27 January 2006; pp. 146–151.
7. Sepúlveda, J.; Strum, M.; Chau, W.J.; Gogniat, G. A multi-objective approach for multi-application NoC mapping. In Proceedings of the 2011 IEEE Second Latin American Symposium on Circuits and Systems (LASCAS), Bogota, Colombia, 23–25 February 2011; pp. 1–4
8. Yang, B.; Guang, L.; Xu, T.C.; Yin, A.W.; Säntti, T.; Plosila, J. Multi-application multi-step mapping method for many-core Network-on-Chips. In Proceedings of the NORCHIP 2010, Tampere, Finland, 15–16 November 2010; pp. 1–6.
9. Khalili, F.; Zarandi, H.R. A Fault-Tolerant Low-Energy Multi-Application Mapping onto NoC-based Multiprocessors. In Proceedings of the 2012 IEEE 15th International Conference on Computational Science and Engineering, Paphos, Cyprus, 5–7 December 2012; pp. 421–428.
10. Zhu, D.; Chen, L.; Yue, S.; Pinkston, T.M.; Pedram, M. Balancing On-Chip Network Latency in Multi-application Mapping for Chip-Multiprocessors. In Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, 19–23 May 2014; pp. 872–881.
11. Khasanov, R.; Castrillon, J. Energy-efficient Runtime Resource Management for Adaptable Multi-application Mapping. In Proceedings of the 2020 Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 909–914.
12. Ge, F.; Cui, C.; Zhou, F.; Wu, N. A Multi-Phase Based Multi-Application Mapping Approach for Many-Core Networks-on-Chip. *Micromachines* **2021**, *12*, 631. [CrossRef] [PubMed]

13. Domingos, P. Machine Learning for Data Management: Problems and Solutions. In Proceedings of the 2018 International Conference on Management of Data, Guangzhou, China, 18–21 February 2022; Association for Computing Machinery: New York, NY, USA, 2018.
14. Li, Y.F.; Guo, L.Z.; Zhou, Z.H. Towards Safe Weakly Supervised Learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *43*, 334–346. [CrossRef]
15. Shang, C.; Chang, C.Y.; Chen, G.; Zhao, S.; Lin, J. Implicit Irregularity Detection Using Unsupervised Learning on Daily Behaviors. *IEEE J. Biomed. Health Inform.* **2020**, *24*, 131–143. [CrossRef]
16. Wang, X.; Kihara, D.; Luo, J.; Qi, G.J. EnAET: A Self-Trained Framework for Semi-Supervised and Supervised Learning With Ensemble Transformations. *IEEE Trans. Image Process.* **2021**, *30*, 1639–1647. [CrossRef]
17. Zhang, Z.; Wang, D.; Gao, J. Learning Automata-Based Multiagent Reinforcement Learning for Optimization of Cooperative Tasks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 1–14. [CrossRef]
18. Liu, X.; Zhang, D.; Zhang, T.; Cui, Y.; Chen, L.; Liu, S. Novel best path selection approach based on hybrid improved A* algorithm and reinforcement learning. *Appl. Intell.* **2021**, *51*, 9015–9029. [CrossRef]
19. Kaur, R.; Singh, A.; Singla, J. Integration of NIC algorithms and ANN: A review of different approaches. In Proceedings of the 2021 2nd International Conference on Computation, Automation and Knowledge Management (ICCAKM), Dubai, United Arab Emirates, 19–21 January 2021; pp. 185–190.
20. Zhang, T.; Zhang, D.; Yan, H.R.; Qiu, J.N.; Gao, J.X. A new method of data missing estimation with FNN-based tensor heterogeneous ensemble learning for internet of vehicle. *Neurocomputing* **2021**, *420*, 98–110. [CrossRef]
21. Dong, S.; Wang, P.; Abbas, K. A survey on deep learning and its applications. *Comput. Sci. Rev.* **2021**, *40*, 100379. [CrossRef]
22. Su, R.; Huang, Y.; Zhang, D.; Xiao, G.; Wei, L. SRDFM: Siamese Response Deep Factorization Machine to improve anti-cancer drug recommendation. *Brief. Bioinform.* **2022** , *23*, bbab534. [CrossRef] [PubMed]
23. Lauzon, F.Q. An introduction to deep learning. In Proceedings of the 2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA), Montreal, QC, Canada, 2–5 July 2012; pp. 1438–1439.
24. Playe, B.; Stoven, V. Evaluation of deep and shallow learning methods in chemogenomics for the prediction of drugs specificity. *J. Cheminform.* **2020**, *12*, 1–18. [CrossRef] [PubMed]
25. Bhuvaneshwari, M.; Kanaga, E.G.M.; Anitha, J.; Raimond, K.; George, S.T. Chapter 7—A comprehensive review on deep learning techniques for a BCI-based communication system. In *Demystifying Big Data, Machine Learning, and Deep Learning for Healthcare Analytics*; Kautish, S., Peng, S.L., Eds.; Academic Press: Cambridge, MA, USA, 2021; pp. 131–157.
26. Tsai, W.C.; Lan, Y.C.; Hu, Y.H.; Chen, S.J. Networks on chips: Structure and design methodologies. *J. Electr. Comput. Eng.* **2012**, *2012*, 2. [CrossRef]
27. Wang, J.; Zhang, M.; Qiu, M. A Diffusional Schedule for Traffic Reducing on Network-on-Chip. In Proceedings of the 2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), Shanghai, China, 22–24 June 2018; pp. 206–210.
28. Siast, J.; Łuczak, A.; Domański, M. RingNet: A Memory-Oriented Network-On-Chip Designed for FPGA. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 1284–1297. [CrossRef]
29. Ahmad, K.; Sethi, M.A.J. Review of network on chip routing algorithms. *EAI Endorsed Trans. Context-Aware Syst. Appl.* **2020**, *7*, 167793. [CrossRef]
30. Abadal, S.; Mestres, A.; Torrellas, J.; Alarcon, E.; Cabellos-Aparicio, A. Medium Access Control in Wireless Network-on-Chip: A Context Analysis. *IEEE Commun. Mag.* **2018**, *56*, 172–178. [CrossRef]
31. Shamim, M.S.; Narde, R.S.; Gonzalez-Hernandez, J.L.; Ganguly, A.; Venkatarman, J.; Kandlikar, S.G. Evaluation of wireless network-on-chip architectures with microchannel-based cooling in 3D multicore chips. *Sustain. Comput. Inform. Syst.* **2019**, *21*, 165–178. [CrossRef]
32. Zhang, W.; Hou, L.; Wang, J.; Geng, S.; Wu, W. Comparison research between xy and odd-even routing algorithm of a 2-dimension 3×3 mesh topology network-on-chip. In Proceedings of the 2009 WRI Global Congress on Intelligent Systems, Xiamen, China, 19–21 May 2009; Volume 3, pp. 329–333.
33. Kadri, N.; Koudil, M. A survey on fault-tolerant application mapping techniques for network-on-chip. *J. Syst. Archit.* **2019**, *92*, 39–52. [CrossRef]
34. Khan, S.; Anjum, S.; Gulzari, U.A.; Afzal, M.K.; Umer, T.; Ishmanov, F. An Efficient Algorithm for Mapping Real Time Embedded Applications on NoC Architecture. *IEEE Access* **2018**, *6*, 16324–16335. [CrossRef]
35. Amin, W.; Hussain, F.; Anjum, S.; Khan, S.; Baloch, N.K.; Nain, Z.; Kim, S.W. Performance evaluation of application mapping approaches for network-on-chip designs. *IEEE Access* **2020**, *8*, 63607–63631. [CrossRef]
36. Zhang, L.; Li, S.; Qu, L.; Kang, Z.; Wang, S.; Chen, J.; Wang, L. MAMAP: Congestion Relieved Memetic Algorithm based Mapping Method for Mapping Large-Scale SNNs onto NoC-based Neuromorphic Hardware. In Proceedings of the 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Cuvu, Fiji, 14–16 December 2020; pp. 640–647.
37. Mei, L.; Houshmand, P.; Jain, V.; Giraldo, S.; Verhelst, M. ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators. *IEEE Trans. Comput.* **2021**, *70*, 1160–1174. [CrossRef]

38. Fang, J.; Zong, H.; Zhao, H.; Cai, H. Intelligent mapping method for power consumption and delay optimization based on heterogeneous NoC platform. *Electronics* **2019**, *8*, 912. [CrossRef]
39. Puschini, D.; Clermidy, F.; Benoit, P.; Sassatelli, G.; Torres, L. Game-Theoretic Approach for Temperature-Aware Frequency Assignment with Task Synchronization on MP-SoC. In Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 3–5 December 2008; pp. 235–240.
40. GNU Octave. Available online: https://www.gnu.org/software/octave/index (accessed on 23 January 2021).
41. Jones, M. NoCsim: A Versatile Network on Chip Simulator. Ph.D. Thesis, University of British Columbia, Vancouver, BC, Canada, 2005.