electronics

MDPI

Article

An FPGA-Accelerated CNN with Parallelized Sum Pooling for Onboard Realtime Routing in Dynamic Low-Orbit Satellite Networks

Hyeonwoo Kim [1], Juhyeon Park [2], Heoncheol Lee [1,3,*], Dongshik Won [4] and Myonghun Han [5]

[1] School of Electronic Engineering, Kumoh National Institute of Technology, Gumi 39177, Republic of Korea; hwkim@kumoh.ac.kr
[2] PGM R&D Lab, LIGNEX1, Seongnam 13488, Republic of Korea
[3] Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gumi 39177, Republic of Korea
[4] TelePIX Corporation, Techno 4-ro, Daejeon 34013, Republic of Korea
[5] Agency for Defense Development, Daejeon 34186, Republic of Korea
* Correspondence: hclee@kumoh.ac.kr; Tel.: +82-54-478-7476

Abstract: This paper addresses the problem of real-time onboard routing for dynamic low earth orbit (LEO) satellite networks. It is difficult to apply general routing algorithms to dynamic LEO networks due to the frequent changes in satellite topology caused by the disconnection between moving satellites. Deep reinforcement learning (DRL) models trained by various dynamic networks can be considered. However, since the inference process with the DRL model requires too long a computation time due to multiple convolutional layer operations, it is not practical to apply to a real-time on-board computer (OBC) with limited computing resources. To solve the problem, this paper proposes a practical co-design method with heterogeneous processors to parallelize and accelerate a part of the multiple convolutional layer operations on a field-programmable gate array (FPGA). The proposed method was tested with a real heterogeneous processor-based OBC and showed that the proposed method was about 3.10 times faster than the conventional method while achieving the same routing results.

Keywords: convolutional neural network (CNN); deep reinforcement learning (DRL); FPGA; low earth orbit (LEO) satellites; algorithm parallelization

Citation: Kim, H.; Park, J.; Lee, H.; Won, D.; Han, M. An FPGA-Accelerated CNN with Parallelized Sum Pooling for Onboard Realtime Routing in Dynamic Low-Orbit Satellite Networks. *Electronics* **2024**, *13*, 2280. https://doi.org/10.3390/electronics13122280

Academic Editors: Fabio Garzetti and Bruno Da Silva

Received: 10 May 2024
Revised: 4 June 2024
Accepted: 6 June 2024
Published: 11 June 2024

1. Introduction

Low earth orbit satellite networks (LEO-SN), which consist of tens to thousands of satellites located at relatively low altitudes (300 km to 1500 km), are used for a variety of applications, including communications, internet, and observation [1]. Compared to geostationary orbit satellites, LEO-SNs have the advantage of lower transmission losses and lower latency due to their low orbital rotation [2–4]. The most powerful aspect of LEO satellite networks, and a major topic of discussion among researchers, is whether they can achieve global coverage. In order to achieve global coverage in a low earth orbit satellite network, many satellites must be deployed, and accurate and fast routing techniques between LEO satellites are required. Therefore, in recent years, research on routing techniques for LEO satellite networks has been actively conducted.

Routing in a LEO satellite network is different from routing on the ground due to a number of considerations. First of all, low-orbit satellites travel at high speeds, so routing must be able to reflect the satellite topology [5–9]. Table 1 summarizes the previous research on LEO satellite routing algorithms. LEO satellite routing algorithms have been actively studied; firstly, some research [10,11] applied reinforcement learning techniques to routing algorithms. In addition, some research [12–14] applied reinforcement learning techniques and included a dynamic environment. Next, one paper [15] included the previous topics

but also included satellite link disconnection. The previous works described focus on LEO satellite routing algorithms, but they did not consider parallelization. Our proposed method is different in that it incorporates the previous topics and uses parallelization to accelerate the algorithm.

Table 1. Routing algorithm research related to reinforcement learning.

Related Works	RL	Dynamic Environment	Link Disconnection	Parallelization
[10,11]	O	X	X	X
[12–14]	O	O	X	X
[15]	O	O	O	X
Proposed Method	O	O	O	O

As discussed previously, LEO satellite networks require fast and accurate routing techniques that account for frequent changes in satellite topology, making it difficult to replicate conventional ground-based methods. In a real LEO satellite network, satellite positions change in real time, and there are link disconnects. In addition, in the space environment, each satellite needs to actively and adaptively schedule and map routes without relying on ground stations. Routing algorithms based on deep reinforcement learning can be a solution to these problems because they are highly adaptive to the real-time changing environment. However, most deep reinforcement learning-based algorithms are computationally intensive; so, it is necessary to reduce the computation time before it can be used for real-time LEO satellite network routing.

An additional problem is the constraints of the LEO satellite network environment. In the space environment, the computing resources used on the ground cannot be used, as such, due to the temporary alteration of bits by cosmic radiation (SEU), permanent damage to semiconductors (SEL), extreme environmental changes, and thermal differences due to the presence or absence of sunlight. Considering these problems, this paper assumes that the computing resources on a satellite are an on-board environment consisting of a central processing unit (CPU) and a field-programmable gate array (FPGA) instead of conventional ground resources. FPGAs are being studied in a variety of fields, because their reconfigurable architecture allows for development customized to user needs, offers the hardware advantage of parallel processing, and is resistant to heat and power consumption [16–19]. However, FPGAs also have a disadvantage in that the performance may be inferior to a CPU if the naive routing technique used in conventional computing resources is applied; so, careful adjustment is required [20].

In this paper, we propose a parallelized reinforcement learning-based routing algorithm for accurate and fast routing in a real-time LEO satellite network environment. The parallelism is achieved through a co-design of a CPU and an FPGA. The contributions of this paper are as follows.

- A practical co-design method with heterogeneous processors to parallelize and accelerate the Dueling-DQN-based routing algorithm was proposed to solve the problem of real-time onboard routing for dynamic LEO satellite networks.
- The sum pooling process in the Dueling-DQN was significantly accelerated on an FPGA by the proposed method, which can be applied to its other applications.
- The proposed method was tested with a real heterogeneous processor-based OBC and showed significantly reduced computation time while achieving the same routing results as the conventional method.

The rest of the paper is organized as follows. Section 2 provides the problem description. In Section 3, the proposed method is described in detail. Section 4 shows the experimental results of the proposed method. Finally, Section 5 gives the conclusions.

2. Problem Description

2.1. Routing Problem in the LEO Network

The routing problem in LEO satellite networks refers to the series of problem of determining the path for each satellite to transmit data to the other satellites in a network consisting of multiple satellites. In this paper, Markov decision processes (MDP) are used to represent this problem mathematically, in particular, grid Markov decision processes [21]. The advantage of using a Grid MDP is that the underlying theory of reinforcement learning can be easily expressed mathematically. In addition, since the state space is organized in the form of a grid, the computation of state transitions and rewards is simpler than in a general MDP. In this paper, a deep reinforcement learning-based routing algorithm that can operate in the Grid MDP environment is used.

The Grid MDP is based on an assumption called the first-order Markov assumption, which states that the probability of a state at time n is only affected by the state at the previous time, $n - 1$. In a Grid MDP, the problem can be modeled in four terms: state, action, reward, and state transition probability. Here, state represents the location of each satellite and the data in the packet stored on that satellite, and action refers to the direction in which the satellite sends data. The reward is the score obtained for sending data in a certain direction, and the state transition probability represents the change in position due to packet transmission.

Using a reinforcement learning algorithm and applying it to a grid MDP eliminates the need to define a global state. In a conventional grid MDP environment, the state, actions, rewards, and state transition probabilities must be modeled in advance to be applied successfully; however, with reinforcement learning, they can be calculated and derived directly without any prior modeling [22,23]. In addition, information about states and rewards is collected in real time through learning, and based on this, the reinforcement learning algorithm helps determine the optimal routing path. Due to the characteristics of reinforcement learning, the initial performance is inferior, because the training data do not exist; however, the performance improves as the training is repeated. In summary, an MDP allows the learning processor to use dynamic programming algorithms such as value iteration or policy iteration to find the optimal value function or policy, and reinforcement learning allows the agent to gain direct experience through state observation and reward to learn the value function or policy. In conclusion, applying a reinforcement learning-based algorithm to low-orbit satellite network routing will require additional time due to the learning time required, but it will eliminate the modeling process of predefining the entire state. A Grid MDP can be used to find a robust policy, and reinforcement learning can be used to improve the results through learning, although the initial results may be unstable; so, it is suitable as a routing algorithm in this environment due to its high adaptability to the dynamically changing satellite network environment.

2.2. Dueling DQN-Based Reinforcement Learning Model

The existing DQN model is an algorithm that combines deep learning and reinforcement learning by constructing a Q-network through a CNN. The conventional DQN has a structure that takes the state as input and outputs a Q-function [24,25]. In this paper, we use Dueling DQN as a reinforcement learning technique, and Dueling DQN is a model that applies a dueling architecture that is more suitable for reinforcement learning instead of the existing neural network structure in the existing DQN model. Dueling DQN has the advantage of effective learning through state-value and noise-resistant learning [26]. The main feature of the Dueling DQN is that it explicitly separates the state-value and advantage functions and uses them to estimate the target. This allows the neural network to estimate the difference between state-value and advantage, which is the direct cause of its performance improvement over the DQN. The structure of a Dueling DQN is characterized by using the same inputs as a conventional DQN, but it adds a separate fully connected layer at the output that considers state-value and advantage. Dueling DQN is a structure that separates state-value and advantage and then combines them to output Q-values.

In dynamic LEO satellite networks and heterogeneous device environments, the computational complexity should be fully considered when using Dueling DQN-based routing algorithms. Since Dueling-DQN involves a complex computation process by a neural network, it is necessary to adjust the amount of computation or accelerate the computation by adjusting it appropriately.

2.3. Problem of On-Board Real-Time Inference

Reinforcement learning-based routing algorithms are effective for dynamic LEO. However, computing resource issues in the on-board environment and the heavy computation time of dueling DQNs are critical to the real-time performance of the algorithm. Reinforcement learning-based routing algorithms must provide optimal routing paths in response to the dynamic situation of the satellite network, which is updated periodically. The goal of this paper is to provide optimal routing paths while responding quickly to various factors such as bandwidth, latency, error rate, and satellite orbits.

However, the CNN-based training and inference process of reinforcement learning-based routing algorithms consumes an excessive amount of time, reducing real-time performance, and ultimately fails to find the optimal routing path. To solve this real-time problem, this paper proposes a heterogeneous device co-design-based inference parallelization method. In this paper, we perform heterogeneous device co-design using PYNQ-Z2, which has both a processing system (PS) that acts as a CPU and a programmable logic (PL) that has the advantages of a hardware FPGA. The dueling DQN-based LEO routing algorithm is basically performed on the PS, and the time-consuming inference process is accelerated in parallel on the PL. The proposed method can reduce the execution time of reinforcement learning and achieve real-time performance.

3. Proposed Method

3.1. Overall Structure of the Dueling-DQN-Based Routing Algorithm

In this paper, the overall flow of the Dueling-DQN-based routing algorithm for LEO satellite networks is as follows. First, the grid environment generation, behavior, and reward are defined. Actions are defined as up, down, left, right, and stop from the current position of the agent. For rewards, -1 point is given for each time step, regardless of behavior, and 1 point is given for reaching the destination. If the agent collides with an obstacle (satellite link disconnection) while moving along the path, it will not be able to perform the action in the next time step, making the penalty larger.

With this predefinition, we proceed to the Dueling-DQN part. The method of Dueling-DQN used in this paper is as follows. First, 2D convolution and ReLU are performed four times through the main layer. This process takes up the most time in the algorithm and is the target of the parallelization described later. After this, the advantage layer and state-value layer perform linear operations. It determines the action that corresponds to the highest result, then performs the action and calculates the position and reward value after the action. The obtained position and reward values are then delivered to the Dueling-DQN as input states and iterated to find the optimal routing path. Figure 1 shows this process schematically.

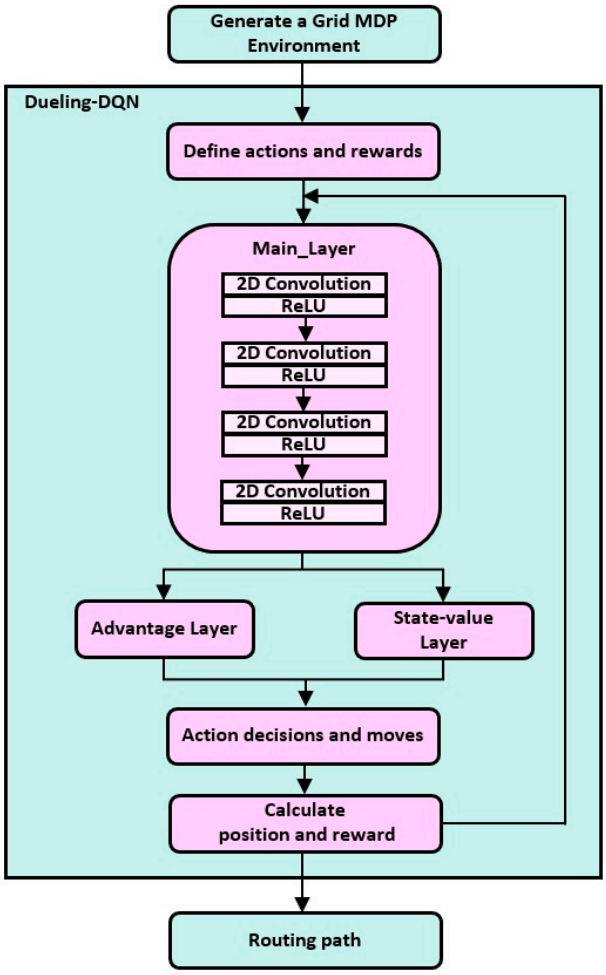

Figure 1. Overall flowchart of the Dueling-DQN-based routing algorithm.

3.2. Dependency Analysis

As explained earlier, the part targeted for acceleration in this algorithm is the convolution operation in the main layer. The parts that proceed before the convolution operation are executed in PS, and the inputs required for the convolution operation are moved to PL for parallel operation and then moved back to PS. Algorithm 1 shows the flow of the operation with pseudo-code for the convolution operation. The Calculate in_idx and Calculate weight_idx of lines 8 and 9 are responsible for calculating the index for the current position in the convolutional input data matrix and the convolutional weight matrix. The key point to note here is line 10 in Algorithm 1. As shown, line 10 is where the accumulation sum operation is located. Since the operation of a CNN uses the pooling method of Sum pooling, there is no way to avoid the accumulation sum operation. However, the problem with this is that it introduces dependencies between loops. This structure, where the result of the previous loop affects the operation of the next loop, is not parallelizable because it is not possible to move to the next loop before the value of the previous loop is calculated. Parallelization in the parent loop is also limited because the operation that introduces the dependency is located in the innermost loop of the six-loop. The following sections describe how to overcome these loop dependencies.

Algorithm 1	**The Conventional Sum Pooling Method**
Input	conv_in[] conv_weight[], conv_bias[] input's height, width, channel kernel size output's height, width, channel
Output	conv_out[]
1.	**for** i = 0~output_channel
2.	**for** j = 0~output_width
3.	**for** k = 0~output_height
4.	float result = 0;
5.	**for** x = 0~input_channel
6.	**for** y = 0~kernel_size
7.	**for** z = 0~kernel_size
8.	**Calculate** in_idx
9.	**Calculate** weight_idx
10.	result += conv_in[in_idx] × conv_weight[weight_idx]
11.	**end**
12.	**end**
13.	**end**
14.	**Calculate** out_idx
15.	conv_out[out_idx] = result + conv_bias[i]
16.	**If**(conv_out[out_idx] <= 0)
17.	Conv_out[out_idx] = 0; //ReLU
18.	**end**
19.	**end**
20.	**end**
21.	**end**

3.3. Parallelized Sum Pooling

In the previous subsection, we discussed the limitations of parallelizing CNNs due to the loop dependency of Sum-pooling. In this section, the process of parallelizing the operation by releasing this loop dependency is introduced. In this paper, a method is proposed to reconstruct the computational structure of the sum-pooling method of CNN to parallelize it. First, the sum-pooling operation of a typical CNN consists of moving a kernel of a predefined size over the input data by the height and width of the input, iterating over the number of channels, accumulating the obtained values, and determining a single feature of the output. As a result, the number of iterations of a single 2D Convolution operation *iter* is calculated as follows:

$$iter = out_{channel} \times out_{width} \times out_{height} \times in_{channel} \times kernel_{size}^{2} \quad (1)$$

Figure 2 shows a schematic representation of the process and direction of the operation described above. The color change of the block means the direction of the operation. The iterative structure described above calculates one feature of the output and moves on to the next feature when the previous calculation is finished. In this paper, we propose a method to avoid loop dependency by calculating the value of all features belonging to a channel simultaneously based on one channel, rather than calculating one feature with a dependency on the output. Figure 3 shows a schematic representation of the sum-pooling method proposed in this paper.

In Figure 3, the direction of the operation is changed from the conventional Sum-pooling in Figure 2. Hear also the color change of the block means the direction of the operation. In the proposed method in Figure 3, the computation proceeds in the Channel direction so that the dependency can be ignored. This does not eliminate the data dependency, but it moves the loops with the data dependency away from each other, making it possible to ignore the dependency under specific conditions. The specific conditions are described in

Section 3.4. As a result, this computational structure is a hardware-friendly computational processing structure that cannot be beneficial to CPUs with sequential processing but can be greatly beneficial to hardware FPGAs with parallel processing characteristics.

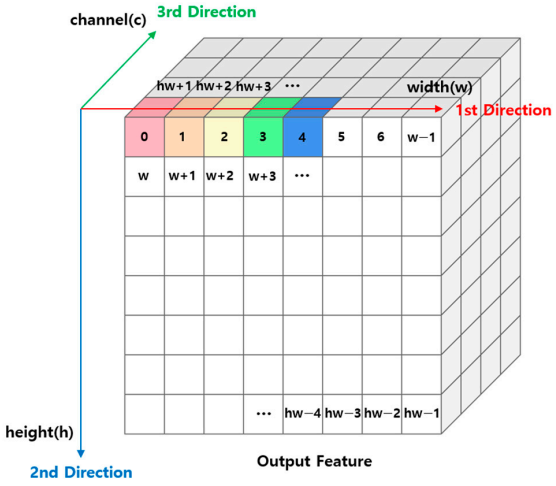

Figure 2. CNN computation structure with conventional sum-pooling.

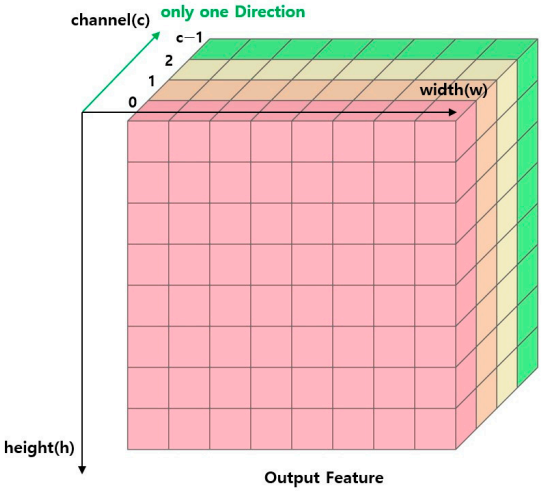

Figure 3. Proposed CNN computation structure with sum-pooling.

3.4. Buffer-Based Dependency Avoidance for Adaptive Pipelining

The implementation of the proposed method described in Section 3.3 is described in this part. In the previous section, it was mentioned that dependencies between loops in the Parallelized Sum pooling method are not eliminated but can be ignored under certain conditions. First, Algorithm 2 shows a pseudo-code that implements the structure described in Section 3.3 for the HLS tool. HLS is an abbreviation for High Level Synthesis. Verilog or VHDL languages are less productive in the hardware design process. Using higher abstraction-level languages, such as C/C++, has several advantages in terms of productivity. HLS is a tool that automatically generates RTL-level code just as a compiler compiles when C/C++ codes used for algorithmic technology, rather than Verilog/VHDL, which was traditionally used in hardware design. Parallelization techniques using HLS tools are already well researched due to their high productivity and portability [27]. The iterations for the output's width and output's height are moved to the lowest level, and a buffer equal to the size of the height multiplied by the width is placed under the top loop. The buffer in line 2 will serve as a buffer to store as many results as there are features in the output of one channel. Next, in Algorithm 2, the pipelining directive is placed on line 10, so it will attempt to pipeline lines 12 through 18. However, in this synthesis, the

dependency on the accumulating sum in line 13 is determined to be valid, so pipelining does not proceed. Therefore, the directive in line 11 explicitly removes the dependency on the result. In this process, lines 12 through 18 are pipelined and can be parallelized.

Algorithm 2	**The Proposed Parallelized Sum Pooling Method**
Input	conv_in[] conv_weight[], conv_bias[] input's height, width, channel kernel size output's height, width, channel
Output	conv_out[]
1.	**for i = 0~output_channel**
2.	float result [output_width × output_height] = 0;
3.	**for** x = 0~input_channel
4.	**for** y = 0~kernel_size
5.	**for** z = 0~kernel_size
6.	**Calculate** weight_idx
7.	float temp = conv_weight
8.	**for** j = 0~output_width
9.	**for** k = 0~output_height
10.	**#Pipeline II = 1**
11.	**#ignore dependence variable = result**
12.	**Calculate** in_idx
13.	result += conv_in[in_idx] × conv_weight[weight_idx]
14.	**Calculate** out_idx
15.	**Calculate** conv_out[out_idx]
16.	**If**(conv_out[out_idx] <= 0)
17.	Conv_out[out_idx] = 0; //ReLU
18.	**end**
19.	**end**
20.	**end**
21.	**end**
22.	**end**
23.	**end**
24.	**end**

For deep and complete pipelining, it is necessary to analyze the execution time of lines 12 to 18. If the latency of 12 to 18 exceeds the size of the buffer, deep and complete pipelining is not possible because the dependencies between the data become valid again. In this context, deep and complete pipelining means that the Initiation Interval is equal to 1. This condition is defined as follows:

$$C_{com} \leq buffer_{size} \tag{2}$$

In this application, the execution cycle *Ccom* for lines 12–18 is 14 clocks, which means that implementing a buffer of 14 clocks or more will allow for deep and complete pipelining.

4. Experimental Results

4.1. Experimental Setups

In this paper, the Dueling-DQN-based routing algorithm was evaluated in the following on-board environment. First, the training to obtain the .pt file containing the weight information was performed on a Jupyter notebook v7.0.8 in the Anaconda v24.3.0 virtual environment, and the training was performed on a desktop PC with i7-10700 and Nvidia Geforce RTX 3070. The model for training was a 15 × 15 grid map consisting of four channels: agent position, goal position, obstacles, and boundaries. The total number of training runs was 100, with a maximum number of steps per run of 300 and a batch size

of 1024. In the convolution process, each kernel was set to 6 × 6, 4 × 4, 3 × 3, and 2 × 2, resulting in 15 × 15 × 4 inputs and 4 × 4 × 32 data outputs.

In addition, for the heterogeneous device co-design of CPU and FPGA, the Board of PYNQ-Z2, which has both PS and PL, was selected. The PYNQ-Z2 is based on the ZYNQ XC7z020-1CLG400C chipset, which includes a 650 MHz ARM Cortex-A9 dual-core processor and an Artix-7 level FPGA. Figure 4 shows the real experimental environment as configured. The FPGA development platform was Xilinx Vivado HLS 2018.3, Vivado 2018.3. First, the code for parallelization was configured in Vivado HLS 2018.3, and the .v file containing the IP information was extracted by synthesizing the configured code. The previously developed IP was then imported into Vivado 2018.3 to design the entire hardware platform. Finally, the .hwh, .tcl, and .bit files containing information about the hardware platform were extracted. PYNQ also provides various tools and libraries, of which we used the overlay library to develop through python. Figure 5 shows the development flow described above schematically.

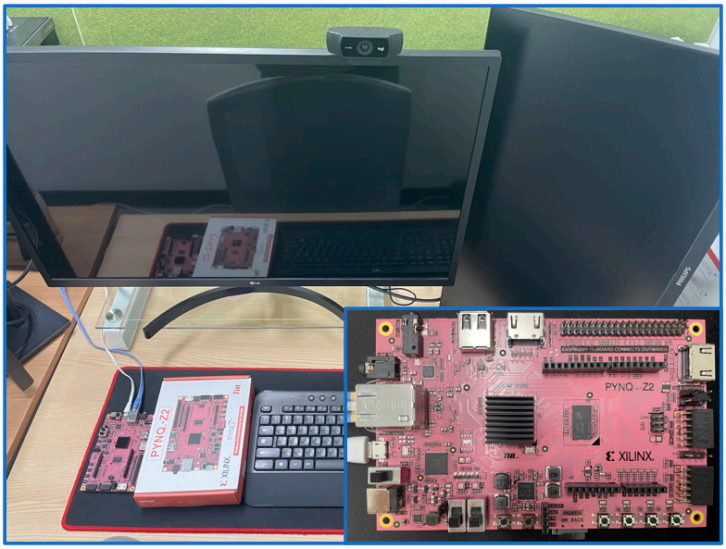

Figure 4. Configured real experimental environment.

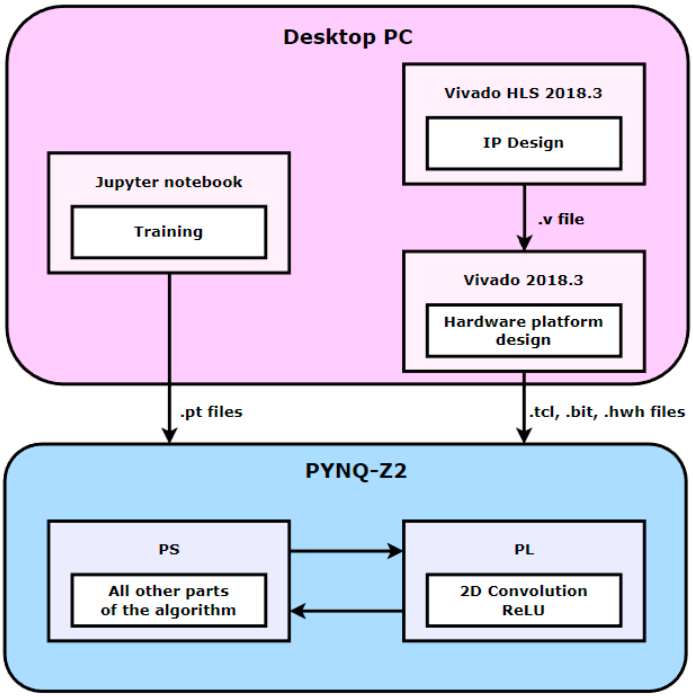

Figure 5. The overall flow of development.

4.2. Training Results

The training results of Dueling-DQN in the environment described above are as follows. First, the starting position of the agent was fixed to (2,2), and the target position was fixed to (10,10). The training was executed a total of 100 times, and the training results are shown in Figure 6. From Figure 6, it is seen that at the beginning of the training, the agent repeatedly obtained the lowest score of −300, but after the initial score, it gradually found the path over time. In the experiment, at the 40th training, the best point (shortest path) was obtained for the first time and eventually converged to the best point.

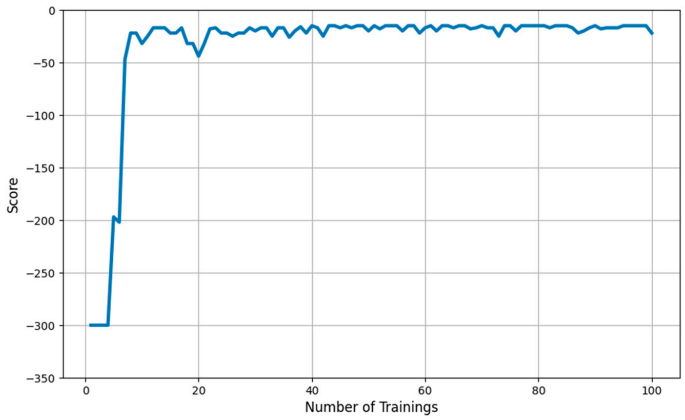

Figure 6. Dueling-DQN-based routing algorithm training results. (Score means point obtained by the agent until it reaches the goal location. Higher scores indicate that the goal location is reached faster).

Figure 7 shows these results in a grid environment, where blue represents the agent, red represents the goal location, black represents the obstacles (link disconnections), and the gray line represents the path obtained. In Figure 7b,d, there is a collision between the obstacle and the path; however, this can be ignored, as the path that was free of obstacles at the time of movement is marked with dynamically moving obstacle positions. As a result, the routing algorithm based on the Dueling-DQN obtained the shortest path.

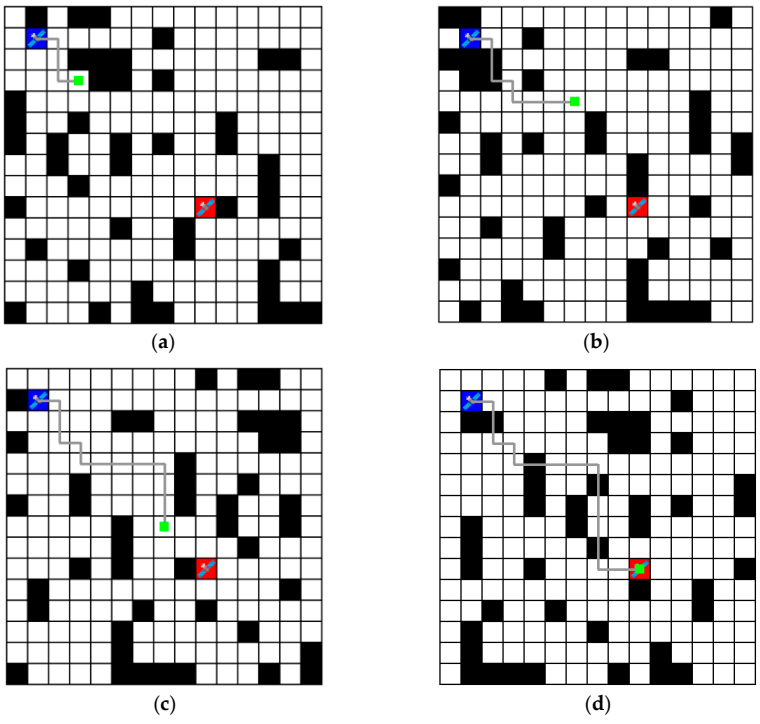

Figure 7. Routing paths represented in a grid environment. (The blue box means agent. The red box means goal location. The black box means obstacles (link disconnections). The gray line means path

obtained. The green dot means location of the agent after each step. (**a**) shows the result after 4 steps. (**b**) shows the result after 8 steps and shows the routing path over the obstacles; however, this is due to the movement of the obstacles and can be ignored. (**c**) shows the result after 12 steps. (**d**) shows the final result after 16 steps).

4.3. Inference Results

As described earlier, the results of inference in the actual on-board environment using the .pt file obtained from the training results and the .hwh, .bit, and .tcl files obtained from the hardware design are as follows. Figure 8 shows the results of inference using PYNQ-Z2, organized by execution time. The experiment was run 10 times for each method, and all results showed that the optimal routing path was found accurately. First, PYNQ-Z2 used only PS for inference, and the average execution time was 0.9396 s. Next, when PS and PL were co-designed, but the naive code was executed on PL without any parallelization process, the execution time was 1.4012 s on average. Finally, when the PS and PL were co-designed, and the 2D convolution operation was parallelized by applying the proposed method, the average execution time was 0.2991 s. When checking the experimental results, it can be seen that the naive method of using the same computation structure in the PL as in the PS without parallelization actually increased the execution time by about 1.51 times compared to the PS only. However, when the parallel 2D convolution operation was used following the proposed method, it was accelerated by about 3.10 times compared to the PS only and by about 4.68 times compared to the naive method. This shows that the proposed method improves real-time performance while maintaining accurate results.

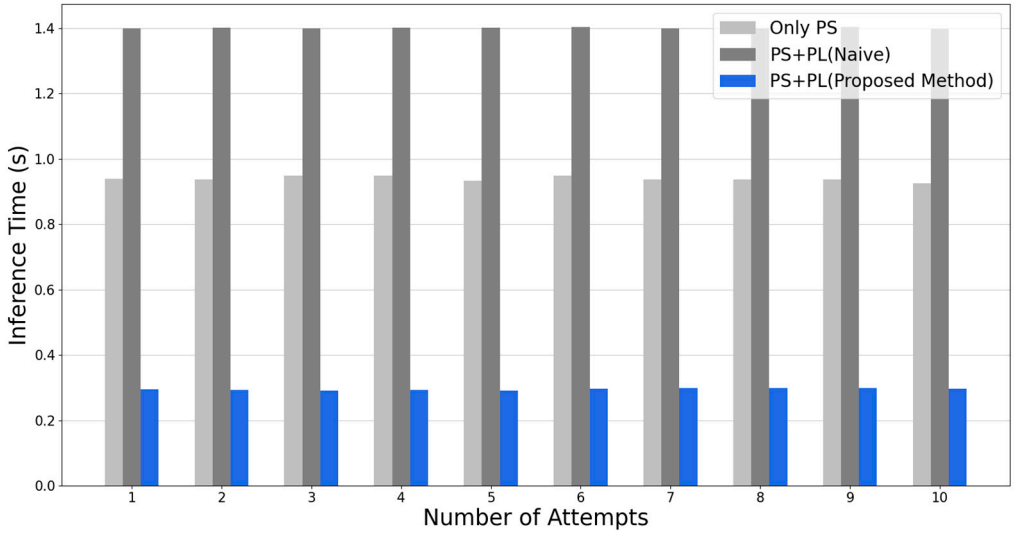

Figure 8. Comparing the inference time of several methods, including the proposed method. (The longer inference time can cause problems with the real-time performance of satellite routing algorithms.).

Finally, Table 2 summarizes the resources of the boards used to design the naive and proposed methods. Both the proposed method and naïve code required PS and PL to perform the work. The ZYNQ XC7z020-1CLG400C has a maximum hardware resource of LUT 53200, LUTRAM 17400, FF 106400, DSP 220, and BRAM 140. Table 2 shows that the proposed method utilized 25.32% of the total resources for LUTs, 4.76% for LUTRAM, 15.51% for FF, 25.36% for BRAM, and 9.09% for DSP, with a slight increase in the usage of the LUTs and BRAM compared to the naive code. However, since the proposed method significantly reduced the inference time, the results of almost the same or slight increase in resource usage compared to the naive code indicate that the proposed method is an efficient method.

Table 2. Algorithm's hardware resource usage.

	Available	Utilization (%)	
		Naïve Method	Proposed Method
LUT	53,200	24.17	25.32
LUTRAM	17,400	4.22	4.76
FF	106,400	15.65	15.51
BRAM	140	22.50	25.36
DSP	220	9.09	9.09

5. Conclusions

This paper proposed a practical co-design method with heterogeneous processors to parallelize and accelerate a part of the multiple convolutional layer operations on an FPGA for onboard real-time routing in dynamic low-orbit satellite networks. Previous work has considered reinforcement learning, dynamic environments, and satellite link outages, but none has considered parallelization. This paper is different in that it includes the previous topics and proposes an FPGA-accelerated CNN with parallelized sum pooling for an onboard real-time routing method. In the sum-pooling structure of conventional CNNs, convolution and ReLU operations require excessive execution time, which is not suitable for the real-time requirements of dynamically changing low-orbit satellite network environments. The proposed method solved this problem by accelerating the algorithm through the co-design of CPU and FPGA, specifically by changing the computation structure of CNN. Experimental results with an OBC with heterogeneous processors including an FPGA showed that the proposed method was about 3.10 times faster than the conventional method while achieving the same routing results.

Author Contributions: All authors contributed to the present paper with the same effort in describing the problem, finding the available literature, and writing the paper. H.K., J.P., and H.L. designed and implemented the proposed method to resolve the problem. D.W. and M.H. addressed the topic and proposed the idea to resolve it. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Agency for Defense Development, funded by the Korean government (Defense Acquisition Program Administration) (UI220033VD).

Data Availability Statement: Data is unavailable due to the restrictions of the organization supporting this work.

Conflicts of Interest: Authors Juhyeon Park and Dongshik Won were employed by the company LIGNEX1 and TelePIX, respectively. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Cakaj, S.; Kamo, B.; Lala, A.; Rakipi, A. The Coverage Analysis for Low Earth Orbiting Satellites at Low Elevation. *Int. J. Adv. Comput. Sci. Appl.* **2014**, *5*, 6–10. [CrossRef]
2. Wang, J.; Li, L.; Zhou, M. Topological dynamics characterization for LEO satellite networks. *Comput. Netw.* **2007**, *51*, 43–53. [CrossRef]
3. Kempton, B.; Riedl, A. Network Simulator for Large Low Earth Orbit Satellite Networks. In Proceedings of the ICC 2021—IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–6. [CrossRef]
4. Hu, Y.; Li, V.O.K. Satellite-based Internet: A tutorial. *IEEE Commun. Mag.* **2001**, *39*, 154–162. [CrossRef]
5. Uzunalioglu, H. Probabilistic routing protocol for low Earth orbit satellite networks. In Proceedings of the ICC '98. 1998 IEEE International Conference on Communications. Conference Record. Affiliated with SUPERCOMM'98 (Cat. No.98CH36220), Atlanta, GA, USA, 7–11 June 1998; Volume 1, pp. 89–93. [CrossRef]
6. Markovitz, O.; Segal, M. Advanced Routing Algorithms for Low Orbit Satellite Constellations. In Proceedings of the ICC 2021—IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–6. [CrossRef]
7. Uzunalioğlu, H.; Akyildiz, I.F.; Bender, M.D. A routing algorithm for connection-oriented Low Earth Orbit(LEO) satellite networks with dynamic connectivity. *Wirel. Netw.* **2000**, *6*, 181–190. [CrossRef]

8. Zhu, Y.; Qian, L.; Ding, L.; Yang, F.; Zhi, C.; Song, T. Software defined routing algorithm in LEO satellite networks. In Proceedings of the 2017 International Conference on Electrical Engineering and Informatics (ICELTICs), Banda Aceh, Indonesia, 18–20 October 2017; pp. 257–262. [CrossRef]
9. Ekici, E.; Akyildiz, I.F.; Bender, M.D. A distributed routing algorithm for datagram traffic in LEO satellite networks. *IEEE/ACM Trans. Netw.* **2001**, *9*, 137–147. [CrossRef]
10. Jiang, C.; Zhu, X. Reinforcement Learning Based Capacity Management in Multi-Layer Satellite Networks. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 4685–4699. [CrossRef]
11. Wang, X.; Dai, Z.; Xu, Z. LEO Satellite Network Routing Algorithm Based on Reinforcement Learning. In Proceedings of the 2021 IEEE 4th International Conference on Electronics Technology (ICET), Chengdu, China, 7–10 May 2021; pp. 1105–1109. [CrossRef]
12. Shi, X.; Ren, P.; Du, Q. Heterogeneous Satellite Network Routing Algorithm Based on Reinforcement Learning and Mobile Agent. In Proceedings of the 2020 IEEE Globecom Workshops, Taipei, Taiwan, 7–11 December 2020; pp. 1–6. [CrossRef]
13. Zuo, P.; Wang, C.; Yao, Z.; Hou, S.; Jiang, H. An Intelligent Routing Algorithm for LEO Satellites Based on Deep Reinforcement Learning. In Proceedings of the 2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall), Norman, OK, USA, 27–30 September 2021; pp. 1–5. [CrossRef]
14. Zuo, P.; Wang, C.; Wei, Z.; Li, Z.; Zhao, H.; Jiang, H. Deep Reinforcement Learning Based Load Balancing Routing for LEO Satellite Network. In Proceedings of the 2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring), Helsinki, Finland, 19–22 June 2022; pp. 1–6. [CrossRef]
15. Lee, J.H.; Chai, K.Y. A Study on the low-earth orbit satellite based non-terrestrial network systems via deep-reinforcement learning. In Proceedings of the Korean Institute of Communication Sciences Conference, Yeosu, Republic of Korea, 17–19 November 2021; pp. 1306–1307.
16. Ma, Y.; Cao, Y.; Vrudhula, S.; Seo, J. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 45–54. [CrossRef]
17. Hoozemans, J.; Peltenburg, J.; Nonnemacher, F.; Hadnagy, A.; Al-Ars, Z.; Hofstee, H.P. FPGA Acceleration for Big Data Analytics: Challenges and Opportunities. *IEEE Circuits Syst. Mag.* **2021**, *21*, 30–47. [CrossRef]
18. Biookaghazadeh, S.; Ravi, P.K.; Zhao, M. Toward Multi-FPGA Acceleration of the Neural Networks. *ACM J. Emerg. Technol. Comput. Syst.* **2021**, *17*, 1–23. [CrossRef]
19. Rahman, A.; Lee, J.; Choi, K. Efficient FPGA Acceleration of Convolutional Neural Networks Using Logical-3D Compute Array. In Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 1393–1398. [CrossRef]
20. Guo, K.; Zeng, S.; Yu, J.; Wang, Y.; Yang, H. [DL] A Survey of FPGA-based Neural Network Inference Accelerators. *ACM Trans. Reconfigurable Technol. Syst.* **2017**, *12*, 1–26. [CrossRef]
21. Garcia, F.; Rachelson, E. Markov Decision Processes. In *Markov Decision Processes in Artificial Intelligence*; Wiley: Hoboken, NJ, USA, 2013; pp. 1–38. [CrossRef]
22. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement Learning: A Survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285. [CrossRef]
23. Arulkumaran, K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Process. Mag.* **2017**, *34*, 26–38. [CrossRef]
24. Ban, T.-W. An Autonomous Transmission Scheme Using Dueling DQN for D2D Communication Networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 16348–16352. [CrossRef]
25. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; Volume 48, pp. 1995–2003.
26. Villanueva, A.; Fajardo, A. Deep Reinforcement Learning with Noise Injection for UAV Path Planning. In Proceedings of the 2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS), Kuala Lumpur, Malaysia, 20–21 December 2019; pp. 1–6. [CrossRef]
27. de Fine Licht, J.; Besta, M.; Meierhans, S.; Hoefler, T. Transformations of High-Level Synthesis Codes for High-Performance Computing. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1014–1029. [CrossRef]