Journal of KIIT. Vol. 21, No. 7, pp. 49-61, Jul. 31, 2023. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2023.21.7.49

동적 저궤도 위성 네트워크에서 온보드 강화학습 기반 라우팅을 위한 이종 프로세서 기반 추론 병렬화 기술

김도형*, 이민준**, 이헌철***, 원동식****, 한명훈*****

Heterogeneous Processor-based Inference Parallelization for On-Board Reinforcement Learning-based Routing in Dynamic LEO Satellite Network

Dohyung Kim*, Minjoon Lee**, Heoncheol Lee***, Dongshik Won****, and Myoung-Hun Han*****

이 연구는 2022년 정부(방위사업청)의 재원으로 국방과학연구소의 지원을 받아 수행된 연구임(UI220033VD)

요 약

본 논문에서는 동적 저궤도 위성 네트워크 라우팅 알고리즘의 OBC(On-Board Computer) 적용 문제를 다룬다. 저궤도 위성 간 연결상태가 동적으로 변하는 네트워크에서 라우팅을 위해 심층 강화학습을 적용할 수 있다. 하지만 심층 강화학습 모델 기반 추론 과정은 다수의 컨볼루션 층 연산으로 인해 과도한 수행시간이 소요되기 때문에 위성 탑재체의 실시간 OBC에 적용하기 어렵다. 본 논문에서는 이 문제를 해결하기 위해, 기존 중앙처리장치에서 순차적으로만 수행되는 추론 과정을 이종 프로세서를 이용한 병렬화 함으로써 수행시간을 단축시킬 수 있는 실용적 방법을 제안한다. 제안된 방법의 성능은 실제 이종 프로세서 기반 OBC를 이용하여 평가되었고, 라우팅 결과는 기존 방법과 동일하면서도 전체 수행시간은 대폭 단축시킬 수 있었다.

Abstract

This paper addresses the routing problem in dynamic low-orbit(LEO) satellite networks for on-board computer (OBC). Deep reinforcement learning can be applied for routing in networks with dynamic connectivity between LEO satellites. However, it is difficult to apply the inference process with deep reinforcement learning models to real-time OBCs because it causes excessive execution time due to the calculation of multiple convolution layers. To solve the problem, we propose a practical method based on heterogeneous processors which can reduce the execution time by parallelizing the inference process, which is performed sequentially in a Central Processing Unit. The performance of the proposed method was evaluated using an actual OBC based on heterogeneous processors, and the routing result was the same as that of the existing method, but the overall execution time was significantly reduced.

Keywords

heterogeneous processors, parallelization, deep reinforcement learning

* 금오공과대학교 전자공학부 학사과정
- ORCID: https://orcid.org/0009-0002-8415-851X
** 금오공과대학교 생산기술연구소 연구원
- ORCID: https://orcid.org/0000-0003-3308-9593
*** 금오공과대학교 IT융복합공학과 교수(교신저자)
- ORCID: https://orcid.org/0000-0003-2962-3474
**** 텔레픽스 주식회사 이사
- ORCID: https://orcid.org/0009-0000-3521-4680
***** 국방과학연구소 선임연구원
- ORCID: https://orcid.org/0009-0004-2571-0153

• Received: Jun. 08, 2023, Revised: Jun. 26, 2023, Accepted: Jun. 29, 2023
• Corresponding Author: Heoncheol Lee
Dept. of IT Convergence Engineering, School of Electronic Engineering
Kumoh National Institute of Technology, Korea
Tel.: +82-54-478-7476, Email: hclee@kumoh.ac.kr

Ⅰ. 서 론

최근, 저궤도 위성 네트워크의 라우팅 기법에 대한 연구들이 활발히 진행중이다. 저궤도 위성 네트워크(LEO-SN)는 수십 개에서 수천 개의 작은 위성들로 구성된 네트워크로, 이 위성들은 지구 궤도 주위를 상대적으로 낮은 고도에 위치하여 통신, 인터넷 및 관측 등 다양한 용도로 사용된다[1]. 저궤도 위성 네트워크의 주요 목적 중 하나는 지구 전역 커버리지 달성이다. 저궤도 위성을 여러 대 배치하여 위성 간 통신을 하면 전송 지연이 적으면서 전역 커버리지 달성이 용이해진다. 또한 이러한 위성들은 지상의 설치된 지점 간 통신을 향상시키거나 통신 기반이 존재하지 않는 위치에서도 사용할 수 있다는 장점이 있다.

하지만 저궤도 위성 네트워크의 라우팅에서는 고려해야 할 점이 많다. 우선 저궤도 위성의 빠른 실행속도이고, 빈번하게 변화하는 위성 토폴로지[2]로 지상에서 사용하는 라우팅 알고리즘을 그대로 사용할 수는 없다[3]. 따라서 저궤도 위성 네트워크의 라우팅 기법을 적용하기 위해서는 기존의 지상에서 사용하던 방식과는 다른 접근을 하여야 한다.

표 1은 저궤도 위성의 라우팅 알고리즘에 대한 기존 연구들이다. LEO-SN을 위한 라우팅 연구들은 지구 전역 커버리지 달성을 위해 활발하게 연구되었다. 표에서 제시되는 연구들은 LEO-SN에서 어떤 부분들을 고려하여 작성되었는지에 대해 보여준다. [4][5]의 경우, 강화학습에 기초하였으나 동적 환경 및 위성 링크 단절에 대한 내용이 포함되어 있지 않았다. [6][7]의 경우는 강화학습 및 동적 환경에 대한 내용이 포함되었지만 위성링크 단절에 대한 내용은 포함하지 않았다.

표 1. LEO-SN 라우팅 알고리즘 관련 논문
Table 1. Papers on LEO-SN routing algorithm

Related works	RL	Dynamic env.	Link dis-connection	Parallelization
[4][5]	O	X	X	X
[6]	O	O	X	X
[7]	O	O	X	X
[8]	O	O	O	X
Ours	O	O	O	O

[8]의 경우 위성 링크 단절은 고려하였지만 병렬화에 대한 내용은 포함되지 않았다. 우리가 제시하는 방법은 저궤도 위성 네트워크 라우팅에서의 보다 사실적인 표현을 위해 동적 환경 및 위성 링크 단절 상황을 고려하여 설계하였다.

저궤도 위성 네트워크에서는 강화학습 기반 라우팅 알고리즘의 중요성이 크다. 강화학습을 사용하여 라우팅 알고리즘을 작성할 경우 실시간으로 변화하는 환경에 대한 내성이 크기 때문이다[9]. 실제 저궤도 위성 네트워크에서는 위치가 실시간으로 변화하고 링크의 단절 구간도 존재한다. 이러한 점들을 고려하여 각각의 상황에 대처를 할 수 있는 알고리즘을 위해서는 강화학습이 필수불가결한 요소라고 볼 수 있다. 우리가 제시하는 기법은 강화학습 기반의 라우팅 알고리즘이며, 동적인 환경과 링크의 단절 구간을 표현할 수 있게 하였다.

하지만 고려해야 할 점은 이 뿐만이 아니다. 기본적으로 저궤도 위성에서는 컴퓨팅 자원을 활용하기 쉽지 않다. 우선 우주 방사선에 대한 문제를 고려하지 않을 수 없다. 우주 항공 및 핵 에너지 시스템과 같이 방사선에 노출되는 환경에서는 SEU(Single Event Upset)와 SEL(Single Event Latchup)이 발생할 가능성이 있다. 또한 극심한 환경변화가 주된 문제점인데, 위성이 궤도에 도착하기까지 진동과 열변형 등에 의해 컴퓨터가 손상될 수 있고 태양광 등으로 인해 극심한 온도 차가 나기 때문이다. 따라서 이러한 문제점들을 고려하여 지상컴퓨팅 자원 보다는 CPU/FPGA를 사용하는 경우가 많다.

하지만 CPU/FPGA의 경우에는 지상에서 사용하는 컴퓨팅 자원에 비해 성능이 떨어진다는 문제점이 존재한다. 특히 강화학습 기반의 라우팅 알고리즘은 많은 연산량을 필요로 하기 때문에 지상컴퓨팅 자원에 비해 연산 속도가 느려질 수 있다[10]. 따라서 OBC 강화학습 기반 라우팅을 사용하기 위해 시스템 병렬화/가속화가 굉장히 중요하다[11]. 최근에는 CPU/FPGA 뿐만 아니라 GPU 기반 알고리즘 가속화 방법도 개발되고 있다[12]-[14].

본 논문에서는 기존의 강화학습 기반 라우팅 알고리즘에서 연산의 많은 시간을 소모하는 convolution 함수를 loop unrolling 기법을 통해 병렬화하여 성능을 향상시키는 방법에 대해 소개한다.

본 논문은 이종 프로세서 기반 추론 병렬화의 전체 구조와 추론코드 추출 과정에 대해 설명한 뒤, 오버레이(Overlay) 기반 HW/SW 동시 설계와 DMA 블록 기반 데이터 처리에 대해 설명한다. 그리고 loop unrolling 기반 CNN 병렬 연산 과정에 대해 제시하고 실험 결과를 통해 본 시스템이 얼마나 가속화 되었는지에 대해 작성하였다.

II. 문제점 기술

2.1 저궤도 위성 네트워크에서의 라우팅 문제

저궤도 위성 네트워크에서의 라우팅 문제는 여러 위성으로 이루어진 네트워크에서 각 위성이 데이터를 전송하는 경로를 결정하는 일련의 문제를 일컫는다. 이를 수학적으로는 마르코프 결정 과정(Markov decision process)으로 표현할 수 있는데, 그 중에서도 그리드 마르코프 결정 과정(Grid MDP, Grid-Markov Decision Process)을 사용할 수 있다.

그리드 마르코프 결정 과정은 “First-order Markov Assumption”이라는 가정에 기반하여, 시간 t에서의 상태는 이전 시간인 t-1의 상태에만 영향을 받는다고 가정한다.

Grid MDP에서는 상태(State), 행동(Action), 보상(Reward), 상태 전이 확률(State transition probability)을 사용하여 문제를 모델링한다. 상태는 각 위성의 위치와 해당 위성에 저장된 패킷의 데이터를 나타내며, 행동은 위성이 데이터를 전송하는 방향을 의미한다. 보상은 특정 방향으로 데이터를 전송했을 때 얻는 점수를 나타내며, 상태 전이 확률은 패킷 전송에 따른 위치 변화를 표현한다.

Grid MDP는 일반적인 MDP에 비해 상태 공간이 Grid 형태로 구성되어 있어서 상태 전이와 보상에 대한 계산이 비교적 간단하다. 본 논문에서는 이러한 Grid MDP 환경에서 동작 가능한 심층 강화학습 기반 라우팅 알고리즘을 사용한다.

강화학습 알고리즘을 사용하여 Grid MDP에 적용시키면 전체 상태에 대해 정의하지 않아도 된다. 기존의 Grid를 이용한 MDP의 경우는 미리 상태, 행동, 보상, 상태 전이 확률을 전부 모델링 해 두어야만 원활하게 적용이 가능하지만 강화학습을 사용하여 진행하게 되면 사전에 모델링을 하지 않고도 직접 계산하여 결과를 도출해낼 수 있다. 대신, 상태와 보상에 대한 정보를 학습을 통해 실시간으로 수집하며, 이를 기반으로 강화학습 알고리즘이 최적의 라우팅 경로를 결정하는 데에 도움을 준다. 초반에는 학습 데이터가 존재하지 않기 때문에 낮은 적중률 및 성적을 보이지만 학습을 거듭할수록 점차 최적의 성적을 거두게 된다. 정리하자면, MDP를 사용하게 되면 학습 프로세스에서 가치 반복이나 정책 반복과 같은 다이나믹 프로그래밍 알고리즘을 사용하게 되어 최적 가치 함수 또는 정책을 찾게 되고, 강화학습을 사용하게 되면 상태 관찰 및 보상을 통해 에이전트가 직접 경험을 쌓아 이를 토대로 가치 함수나 정책을 학습하게 된다.

따라서 강화학습 알고리즘을 사용하게 되면, 기존의 방식에 비해 학습이라는 시간이 추가되게 되지만, 전체 상태에 대해 미리 정의하는 모델링 과정이 생략되게 된다. 추가로 MDP를 활용하는 방법의 경우 확실한 정책을 찾을 수 있는 반면 강화학습을 이용한 방법은 학습 결과가 불안정해질 수 있지만 본 논문에서 제시하는 동적 상황에서의 위성 라우팅 알고리즘에 적용하기에는 적합하다고 판단했다.

2.2 Dueling DQN 기반 학습 모델

Dueling DQN은 기존 DQN 알고리즘의 구조적인 부분을 개선하여 성능을 향상시킨 모델이다. DQN(Deep Q-Network)는 Q-Learning의 딥러닝 기반 확장으로, 강화학습 문제를 해결하기 위한 신경망 구조이다. 기존 DQN은 상태를 입력받아 Q함수를 출력하는 구조를 가지고 있다[15]. Dueling DQN의 주요한 특징으로 가치 함수(Value function)와 행동 가치 함수(Advantage function)를 명시적으로 분리하여 대상을 추정하는데 사용한다[16]. 이를 통해, 신경망은 상태 가치와 행동에 따른 가치의 차이를 추정할 수 있으며, 이로 인해 기존 DQN보다 더 큰 환경 내에서 학습 성능을 높일 수 있다. Dueling DQN의 구조는 기존 DQN과 동일한 입력부를 사용하나 출력부에 Value와 Advantage를 고려한 별도의

계층을 추가하는 것이 특징이다. 이렇게 구성된 Dueling DQN은 상태 가치와 행동 가치를 분리하고 하나의 종합된 Q-value로 합쳐 계산한다.

동적 저궤도 위성 네트워크와 이종 프로세서 환경에서 Dueling DQN 기반 라우팅 알고리즘을 사용할 때 연산 복잡성 문제를 반드시 고려해야 한다. Dueling DQN은 신경망에 의한 복잡한 연산 과정을 포함하므로, 신경망의 크기, 깊이를 적절히 조절해 연산량을 줄이는 방안을 반드시 고려하여야 한다.

2.3 온보드 강화학습 기반 라우팅의 문제점

온보드 강화학습 기반 라우팅이 동적 저궤도 위성 네트워크에서 효과적인 솔루션을 제공할 수 있지만, 여전히 몇 가지 문제점이 존재한다. 그 중 핵심 문제는 강화학습의 수행 시간 때문에 발생하는 실시간성 문제이다.

강화학습 알고리즘은 주기적으로 업데이트되는 위성 네트워크의 동적 상황에 대응하여 라우팅 결정을 내리는 데 사용되며, 이 과정에서 다양한 요인들을 고려해야 한다. 이러한 요인들에는 대역폭, 지연 시간, 오류율, 위성의 이동 궤도 등이 포함된다. 이러한 변수들 변화에 신속하게 대응하면서 최적의 라우팅 솔루션을 제공하는 것이 온보드 강화학습 기반 라우팅의 목표이다.

그러나 강화학습 기반 라우팅 알고리즘의 훈련 및 추론 과정에서 소비되는 시간이 많을수록 실시간성이 저하되며, 동적 저궤도 위성 네트워크의 변경 요인들에 빠르게 대응하는 데 어려움이 발생한다. 이로 인해 강화학습 기반 라우팅 알고리즘이 최적의 솔루션을 찾지 못하거나 지연이 발생하게 된다. 이러한 실시간성 문제를 해결하기 위해 본 논문에서는 이종 프로세서 기반 추론 병렬화 기술을 제안한다. Dueling DQN 알고리즘을 사용한 온보드 강화학습 기반 라우팅에서 PL에서 연산을 병렬화하여 추론 과정을 가속화하는 방식을 사용하게 된다. 이를 통해 강화학습의 수행 시간을 줄이고, 실시간성 문제를 해결할 수 있을 것으로 생각된다.

본 논문의 후속 챕터에서는 이종 프로세서 기반 추론 병렬화 기술의 세부 사항과 구현, 그리고 성능 향상에 대해 더 깊게 다루게 된다. 그 결과, 본 논문의 제안 방법을 도입함으로써 동적 저궤도 위성 네트워크에서의 온보드 강화학습 기반 라우팅의 실시간성과 성능 개선을 기대할 수 있다.

III. 제안하는 기법

3.1 이종 프로세서 기반 추론 병렬화

본 논문에서는 다음과 같은 순서로 이종 프로세서 기반 추론 병렬화를 진행한다. 우선 PS에서 그리드 환경 생성 및 5가지 행동에 대해 정의하고, reward를 설정한다. 행동은 현재 에이전트의 위치에서 상, 하, 좌, 우, 정지로 총 5가지 행동에 대해 정의한다[17]. Reward는 5가지 행동에 관계 없이 매 time step마다 1점을 감점하며, 도착지에 도착하였을 경우 1점을 상점으로 부여하고 과정이 종료된다. 만약 에이전트가 이동 중 장애물(위성 링크 단절)에 부딪힐 경우 다음 time step에는 동작을 하지 못한다. 즉, 장애물에 부딪히는 경우는 2 time step을 소모하기 때문에 총 -2점 감점이라고 생각할 수 있다.

이렇게 PS에서 환경 생성 및 행동과 보상에 대해 설정하고 나면 Dueling DQN 파트로 넘어온다. 본 논문에서 제시하는 Dueling DQN은 우선 main layer를 통해 2D convolution과 ReLU 과정을 총 4번 진행한다. 이 과정을 PL로 데이터를 보내서 연산한다. 전체 프로세스에서 convolution 연산을 담당하는 부분이 가장 크기가 크고 시간이 오래 걸리기 때문에 해당 부분을 PL에서 연산하여 병렬화를 진행한다. PL에서 convolution 및 ReLU 과정을 거치고 나면 다시 PS로 데이터를 보내 advantage layer와 value layer에서 linear 연산을 수행한다. 이렇게 도출된 연산 값 중 최고치에 해당하는 행동을 판단한 뒤, 그 행동을 수행하고 수행 후의 위치와 보상 값을 계산한다. 이렇게 나온 위치와 보상 값을 다시 현재 state로 넘겨 위의 과정을 반복하며 최적의 라우팅 경로를 찾아낸다. 그림 1은 이종 프로세서 기반 추론 병렬화 과정을 도식화한 것이다.

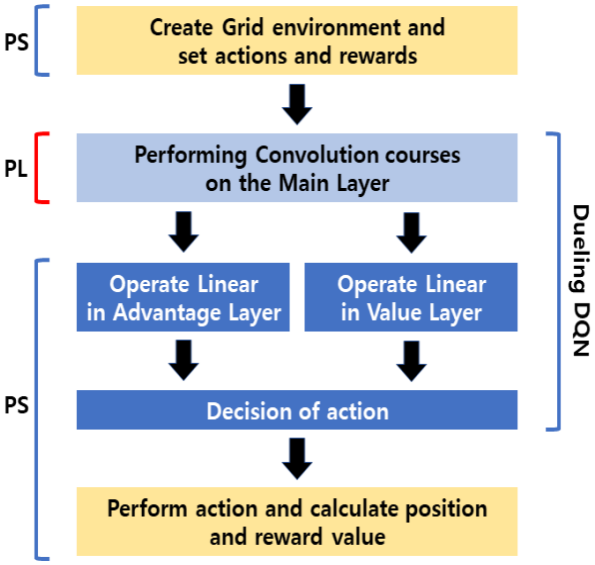


그림 1. 이종 프로세서 기반 추론 병렬화 과정 도식화
Fig. 1. Heterogeneous processor-based inference parallelization process schematic

3.2 추론 코드 추출

본 논문에서는 온보드 강화학습 기반 라우팅을 위하여 Dueling DQN 코드를 Jupyter 환경에서 학습시키고, 가중치 파일을 Pynq 보드로 전송하여 추론 과정을 진행한다. 이를 위해 이종 프로세서 기반 추론 병렬화 기술을 제안한다. 추론 코드 추출 과정은 다음과 같이 진행된다.

먼저, Dueling DQN에 사용된 신경망 구조를 확인한다. 이를 통해 각 계층별로 가중치와 계수를 별도의 파일 형태로 저장하며, 이를 .pt 파일 형식으로 변환한다. 저장된 .pt 파일을 Pynq 보드로 업로드하여 추론 과정 시작에 앞서 필요한 정보를 불러오도록 한다.

또한, 추론 과정은 병렬화하여 처리되도록 한다. 이를 통해 Pynq 보드의 이종 프로세서를 활용하여 병렬 연산이 가능한 구조로 코드를 수정한다. 이를 통해 효율적인 병렬 처리가 가능하도록 설계한다.

Convolution 연산을 담당할 코드는 HLS tool에서 직접 설계한다. 기본적으로 본 논문에서 제시하는 방법은 4번의 convolution과 ReLU 과정을 포함하기 때문에 HLS에서 이를 코드로 작성하여 IP화한다. 그 후 IP를 가지고 block design하여 bitstream, blockdesign, hardware hondoff 파일을 추출한 뒤 Pynq에 업로드한다.

이렇게 가중치 파일과 block design을 통해 추출한 파일이 업로드가 완료되면 Pynq를 LAN port를 통해 접속한다. Jupyter 환경을 사용하였으며 .ipynb 파일을 생성한 뒤 라이브러리와 함수를 호출한다. 그 후 기존의 Dueling DQN의 추론 코드에서 main layer 부분을 PL로 병렬화하는 코드를 통해 진행한다.

3.3 오버레이(Overlay) 기반 HW/SW 동시 설계

Pynq는 python을 사용하여 프로그래밍, 하드웨어 가속, 주변 장치 제어 등을 할 수 있도록 도구와 라이브러리를 제공한다. 그 중 Overlay 라이브러리는 이러한 임베디드 시스템에서 FPGA를 프로그래밍하고 제어하기 위한 도구이다[18].

오버레이 기반 HW/SW 동시 설계란, 하드웨어와 소프트웨어를 효과적으로 연계하여 동시에 설계하고 구현하는 개발 방식을 말한다. 오버레이는 FPGA 하드웨어의 복잡한 기능들을 소프트웨어 측면에서 추상화된 인터페이스로 제공함으로써, 개발자가 하드웨어와 소프트웨어 간의 상호 작용을 쉽게 관리할 수 있게 돕는다.

Overlay를 사용하면 기존의 python 코드가 수정되었을 때, 코드 전체를 수정하는 것이 아닌 HLS 코드만 재작성하여 사용할 수 있다는 장점이 존재한다. 본 논문에서 제시하는 방법은 PS에서 연산하는 부분과 PL에서 연산하는 부분이 명확하게 구분되어 나누어져 있는데[19], 만약 PS에서 동작하는 코드가 수정되었으면 그 부분만 수정하면 되고 PL에서 동작하는 코드가 수정되었으면 그 부분의 block design만 새롭게 하면 된다는 장점이 있다. 이를 통해 사용자는 복잡한 하드웨어 디자인 과정과 세부 사항을 크게 나누어 집중할 필요 없이 전체 시스템 개발에 초점을 맞출 수 있다. 또한 여러 프로젝트에서 중복된 코드나 모듈을 공유하여 시간을 절약할 수 있으며, 기존의 Overlay에 새로운 하드웨어 기능을 추가하여 확장할 수도 있다.

3.4 DMA 블록 기반 데이터 처리

DMA(Direct Memory Access) 블록 기반 데이터 처리는 프로세서의 개입 없이 메모리와 주변장치 간의 데이터 전송을 효율적으로 처리하는 기술이다. 이 방식으로 데이터 처리를 실행하면, 프로세서의 부담을 줄이고 시스템의 전체적인 성능을 향상시킬 수 있다[20].

일반적인 데이터 처리는 중앙 처리 장치를 통해 메모리와 주변장치 간의 데이터 이동이 발생한다. 이와 달리 DMA 블록 기반 데이터 처리는 특별한 하드웨어 컨트롤러인 DMA 컨트롤러가 직접 메모리에서 데이터를 읽고 주변 기기로 전송하거나, 주변 장치로부터 데이터를 읽고 메모리로 쓰는 작업을 처리한다[21].

DMA 블록 기반 데이터 처리를 하면 독립적인 DMA 컨트롤러가 데이터 전송을 수행하기 때문에 전체 시스템의 성능이 향상될 수 있다. 또한 프로세서와 DMA 컨트롤러가 독립적으로 작동하므로 시스템의 병렬 처리 능력이 증가해 더 많은 작업을 동시에 처리할 수 있다.

그림 2는 본 논문에서 제시하는 방법에 따라 설계한 block design이다. 좌상단의 위치한 block은 DMA block이다. DMA는 PS-PL은 대용량 전송을 위해 선택한 매개체로 stream 방식으로 데이터를 송수신하며 data와 last로 나뉜다. data는 전송할 float형 데이터고 last는 마지막을 나타내는 불리안형 변수이다. 우리는 총 9개의 DMA를 사용하고, 그 중 8개는 데이터 읽기 전용 DMA이고 1개는 입출력을 모두 담당한다. 9개의 데이터를 DMA를 통해 입력받는 방식인데, 이 때 현재 상태와 목적지, 장애물, 경계면 정보를 담은 입력 데이터 1개와 convolution 연산에 필요한 weight 값과 bias 값을 총 8개 입력으로 받는다. DMA를 통해 데이터를 입력받고 나면 PL에서 연산을 수행한 뒤 출력 데이터 하나를 DMA를 통해 다시 내보낸다. 그림 3은 DMA를 통해 PS와 PL간 데이터 흐름에 대해 도식화 한 그림이다. M은 Memory-Mapped to Stream을 뜻하며 Memory-Mapped된 데이터를 스트림 형태로 전송하는 기능을 하며, 읽기 채널에 해당한다. S는 Stream to Memory-Mapped을 뜻하며 스트림 형태의 데이터를 메모리 Memory-Mapped 된 형태로 전송하는 기능을 가르킨다. 이는 쓰기 채널에 해당된다[22].

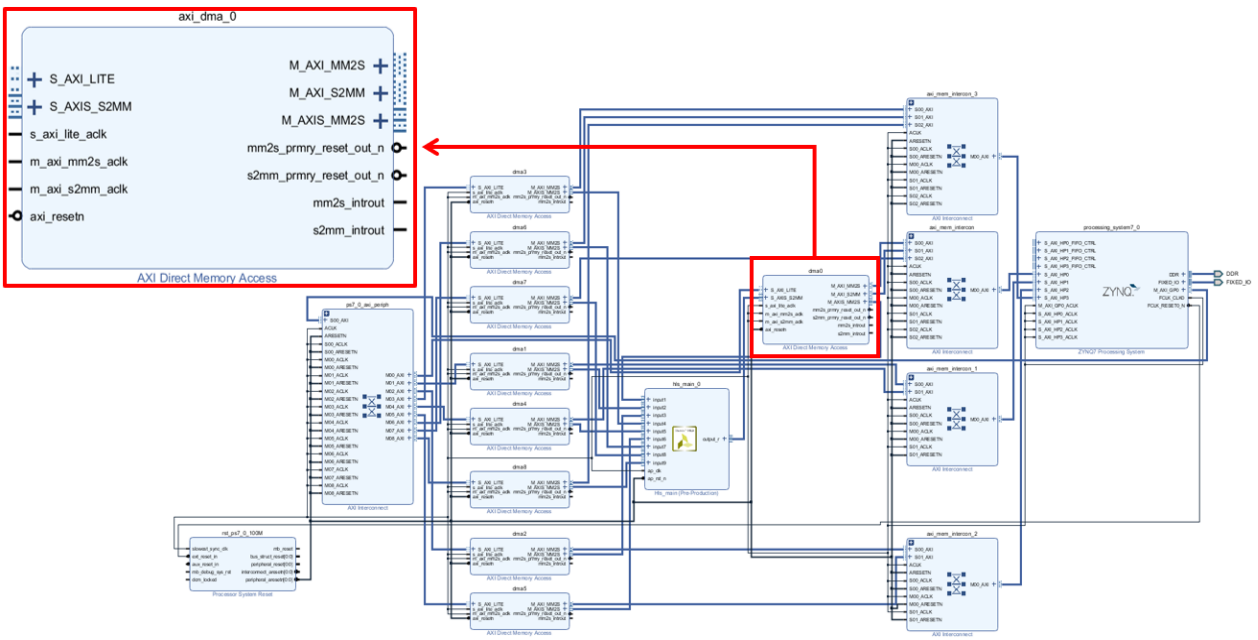


그림 2. 본 논문에서 제시하는 블록 디자인
Fig. 2. Block design presented in this paper

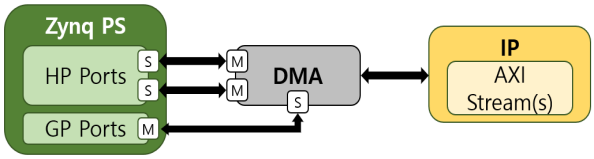


그림 3. DMA 데이터 흐름 도식화
Fig. 3. Schematic DMA dataflow

3.5 Loop unrolling 기반 CNN 병렬 연산

우리가 제시하는 Dueling DQN에서, 연산에 상당한 시간을 소모하는 부분은 바로 convolution 함수다. Convolution은 수 많은 입력 데이터와 weight, bias를 입력 받아서 연산을 진행한다. 특히 convolution 구조 자체가 6중 반복문으로 되어있기 때문에 PL에서 병렬화 연산을 할 때 굉장한 부담을 준다. 따라서 본 논문에서 는 입력 채널과 출력 채널을 각각 배수로 고정하고 해당 loop에 대해 loop unrolling 기법을 사용한다.

Loop unrolling은 컴퓨터 프로그래밍에서 성능 최적화를 위해 사용되는 기술 중 하나다. 해당 기술은 반복문 내의 연산을 반복 실행 횟수를 줄이고, 대신 각 반복에서 수행하는 연산을 늘려서 전체 실행 시간을 단축시키는 것을 목표로 한다[23]. Loop unrolling을 수행하는 과정에서는, loop의 몸체가 더 많은 연산을 수행하도록 코드가 수정된다[24]. 이렇게 하면 loop를 돌면서 발생하는 overhead가 줄어들되, 실행 시간이 단축되므로 전체 성능이 향상된다. 또한 loop unrolling 기법을 사용할 경우 pipelining이나 vectorization과 같은 고급 병렬 처리 기술을 효과적으로 사용할 수 있다.

그림 4는 해당 논문에서 사용한 기법을 도식화한 예시이다. 기존의 convolution 함수에 있는 변수들의 종속성을 배열을 통해 제거하여 loop unrolling을 진행한 뒤, pipelining을 적용하여 기존보다 높은 성능을 낼 수 있다. 본 논문에서 사용하는 Dueling DQN의 경우 현재 위치, 목표 위치, 장애물, 경계면의 총 4개의 채널은 입력으로 받는다. Convolution 과정을 진행할 때 입력 채널과 출력 채널만큼 반복문을 거치기 때문에 입력 채널과 출력 채널을 4의 배수, 8의 배수로 고정하여 loop unrolling이 용이한 상태로 바꾼다.

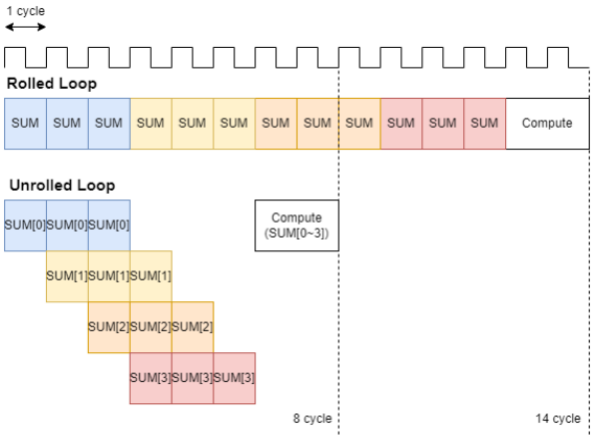


그림 4. 본 논문에서 사용한 Loop unrolling 기법 도식화
Fig. 4. Schematic illustration of loop unrolling technique used in this paper

이렇게 unrolling 된 코드를 가지고 pipelining을 하게 되면 기존보다 클럭 당 연산을 많이 수행할 수 있게 되어 최종적으로는 연산 수행 시간이 감소하게 된다.

IV. 결과 및 분석

4.1 구현 환경

해당 실험을 진행하기 위해 보드는 PYNQ-Z2 (Python Productivity for Zynq) 버전을 사용하였다. 보드에 Jupyter notebook을 설치하여 환경을 구성하였으며 PYNQ OS 2.4 및 Pytorch 1.2.0a 버전을 사용하였다. 학습은 Anaconda 가상환경을 이용한 Jupyter notebook에서 진행하였다. 학습에 사용한 PC의 사양은 Intel core i7-10700 및 Nvidia Geforce RTX 3070이 탑재된 Desktop을 사용하였다.

학습과 추론에 사용한 모델은 총 15x15의 grid 맵으로 총 4개의 채널로 구성된다. 각각의 채널은 에이전트의 위치, 목표 위치, 장애물, 경계면을 뜻하며 에이전트의 위치 및 목표 위치는 전체 15x15 맵 기준으로 구성했으며 장애물 및 경계면의 경우 현재 에이전트의 위치를 기준으로 학습하였다. 학습 횟수는 총 100회 반복하였으며 한 번의 학습 당 최대 step 수는 300, batch size는 1024로 설정하였다.

Convolution 과정의 경우 커널 사이즈를 6x6, 4x4, 3x3, 2x2로 설정하였으며 최종적으로 나오는 출력의 경우 15*15*4의 데이터를 입력으로 넣어 4*4*32의 데이터가 나온다. Loop unrolling을 사용하기 위해 입력 채널 및 출력 채널은 각각 4와 8의 배수로 고정하여 진행하였다.

4.2 결과 및 분석

4.2.1 학습 및 추론 결과

그림 5는 Dueling DQN 학습 결과이다. 학습에 앞서 에이전트의 위치는 (2, 2)로 고정하였고, 목표 위성의 위치는 (10, 10)으로 고정하였다. 학습은 총 100회 진행되었으며, 학습의 결과는 초반에는 상점을 받지 못해 최하점인 -300점을 계속해서 획득한 것을 알 수 있고 상점을 받은 순간부터는 경로를 점점 찾아가는 것을 볼 수 있다. 커널 사이즈를 크게 가져가서 속도에서의 이점을 가지고 대신 초반 수렴이 늦어지는 경향을 보이는데, 그럼에도 불구하고 50회를 채우기 전 -14점(에이전트의 초기 위치에서 목표 지점까지의 최단 경로)을 얻은 것을 확인할 수 있다.

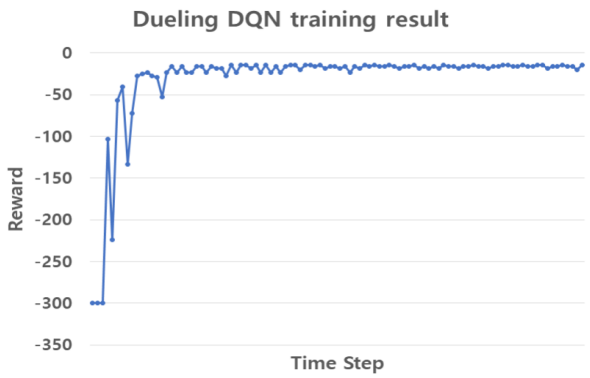


그림 5. Dueling DQN 학습 결과
Fig. 5. Dueling DQN training result

그림 6에서 청색은 에이전트, 적색은 목표 위치를 나타낸다. 지도에 나타난 검정색 부분은 모두 장애물(위성 링크 단절)을 의미한다. 강화 학습을 통해 목표한 위치까지 장애물들을 회피하며 최단 경로를 얻어낸 것을 확인할 수 있다.

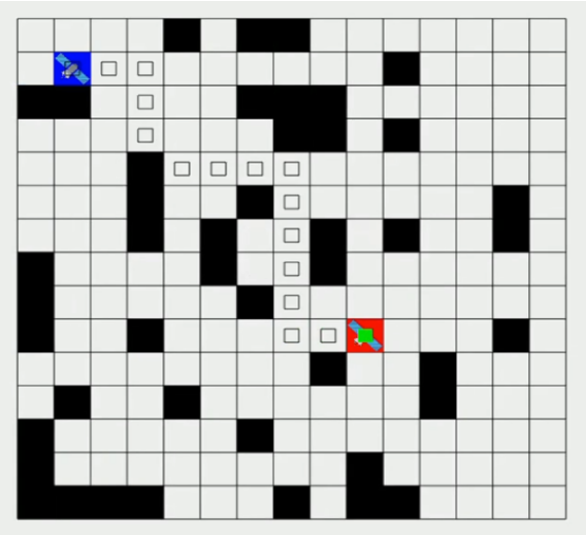


그림 6. 에이전트의 경로
Fig. 6. Path of the agent

4.2.2 순차적 추론 결과

순차적 추론은 FPGA의 PL 부분을 사용하지 않고 PS에서만 모든 연산을 돌렸을 때의 결과를 나타낸다. 추론은 총 10번 시도하였으며 모두 추론에 성공하였다. 점수는 -14점으로 해당 환경에서 얻을 수 있는 최고 점수에 도달한 것을 알 수 있다. 또한, 평균 추론 시간은 0.96796658초로 측정되었으며 항상 같은 성능을 낼 수 있는 것은 아니다 보니 0.02초 정도의 오차가 발생하기는 했다. 그림 7은 순차적 추론 결과이다. 좌측은 이동 경로이고 우측은 순차적 추론을 10번 진행한 결과이다.

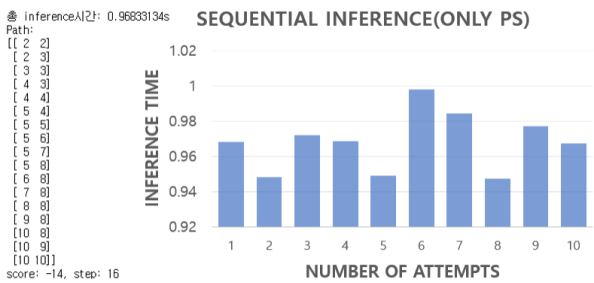


그림 7. 순차적 추론 결과
Fig. 7. Sequential inference result

4.2.3 병렬적 추론 결과

다음은 순차적 추론과의 결과 차이 확인을 위하여 병렬적 추론을 진행한 결과이다.

본 논문에서는 pytorch 라이브러리의 conv2d 함수를 활용한 병렬적 추론 결과를 얻기 위해 conv2d 함수를 HLS 코드로 변환하였다. 이후, vivado를 통해 HLS 코드를 IP화하고 블록디자인을 완료하여 bitstream, blockdesign, hardwarehandoff 파일을 보드에 업로드 한 뒤 사용하였다.

병렬화하여 추론을 진행한 결과를 보면 1.2417초 정도의 평균 추론 시간을 얻은 것을 확인할 수 있다. PS에서 순차적 추론을 할 때보다 평균 추론 시간이 오래 걸린 이유는 convolution 연산을 하기 위해 초기 상태와 weight, bias 값을 받아와서 연산을 진행하게 된다. 하지만 HLS 코드에서 내부적으로 연산을 진행할 때에는 float32 부동소수점 형태를 사용하게 되는데, 총 4번의 convolution 연산을 진행하기 때문에 굉장히 많은 연산시간을 필요로 하며, 또한 최단 경로로 진행한다고 하더라도 총 16번의 PL-PS 데이터 전송이 발생한다. 이와 같은 이유로 overhead가 굉장히 크고 순차적 추론에 비해 연산 시간이 늘어난다.

기존 convolution 함수를 HLS로 그대로 구현하게 되면 latency가 200만 clk이 넘게 측정된다. 또한 자원 사용량이 20%를 넘기는 것이 없으며 사용할 수 있는 자원의 양보다 훨씬 적은 양만을 사용하여 연산을 수행하기 때문에 효율이 떨어진다. 그림 8은 병렬적 추론을 진행한 결과이고, 상단은 latency 및 자원 사용량이다.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.567	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2075976	2075976	2075976	2075976	none

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	767
FIFO	-	-	-	-
Instance	0	20	3901	8181
Memory	43	-	129	20
Multiplexer	-	-	-	1640
Register	-	-	957	-
Total	43	20	4987	10608
Available	280	220	106400	53200
Utilization (%)	15	9	4	19

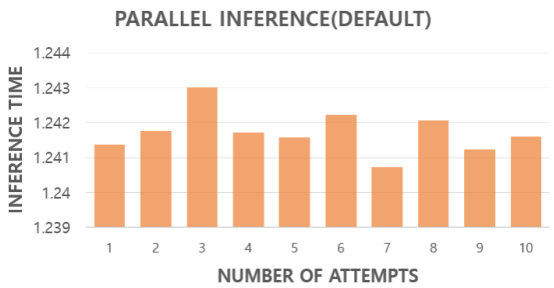


그림 8. 병렬적 추론 결과
Fig. 8. Parallel inference result

4.2.4 Loop unrolling 범위에 따른 수행시간 비교

다음은 병렬적 추론 결과에 loop unrolling을 적용한 결과이다. 기존의 convolution 함수를 그대로 구현한 것이 아닌, 6중 반복문에서 입력 채널과 출력 채널만큼 반복하는 부분을 loop unrolling 한 결과이다. 왼쪽의 그림은 loop unrolling을 출력 채널에 대해 8번 적용한 결과이고 우측의 그림은 출력 채널에 대해 8번, 입력 채널에 대해 4번 적용한 결과이다. 본 논문에서 사용하는 코드는 입력 채널은 4의 배수, 출력 채널은 8의 배수로 고정되어있어 이와 같은 기법이 적용 가능하다. 만약 채널이 나누어 떨어지지 않다면 다른 부분에서 loop unrolling을 진행하여야 한다.

평균 추론 시간을 비교하면 기존의 병렬적 추론 결과의 경우 1.2417초로 기존의 순차적 추론 결과보다 낮은 성능을 보였었던데 반해 loop unrolling을 하나의 반복문에 대해 적용한 결과는 0.4236초, 두 개의 반복문에 대해 적용한 결과는 0.2824초로 측정되었다. 이는 각각 2.93배, 4.39배 가속화되었다. 그림 9은 loop unrolling에 따른 추론 시간을 총 10회 측정하여 비교한 그림이다.

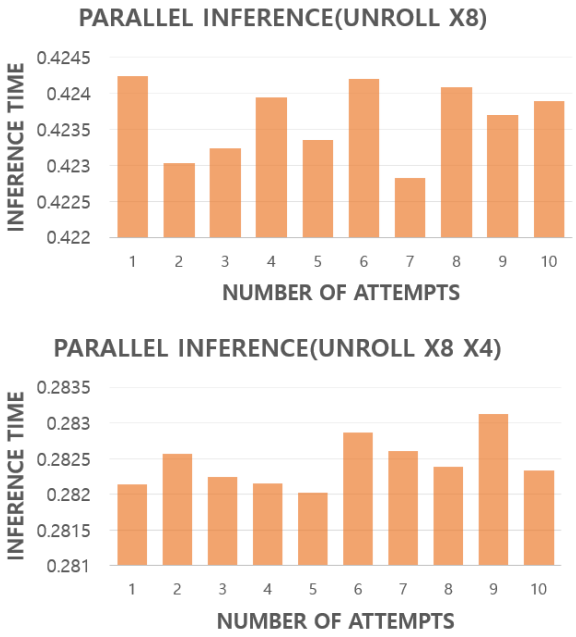


그림 9. Loop unrolling에 따른 추론 시간 비교
Fig. 9. Comparison of inference time according to loop unrolling

4.2.5 Loop unrolling 범위에 따른 자원 사용량 비교

다음은 loop unrolling 범위에 따른 자원 사용량의 비교이다. 그림 10은 loop unrolling에 따른 자원 사용량 비교에 대한 그림이다. 우선 자원 사용량을 보면 LUT(Look Up Table)가 loop unrolling을 사용하지 않았을 때 25% 수준이었던 것에 반해 반복문 두 개에 대해 loop unrolling을 진행하였을 때에는 50% 수준까지 LUT 사용량을 상승시켰다. 그 외에도 FF(Flip-Flop), DSP(Digital Signal Processing) 또한 각각 26%, 32% 수준으로 올라왔다. 이렇게 하드웨어 자원 사용량이 늘어났다는 것은 즉 데이터를 효율적으로 처리할 수 있어 성능이 향상될 수 있음을 의미한다.

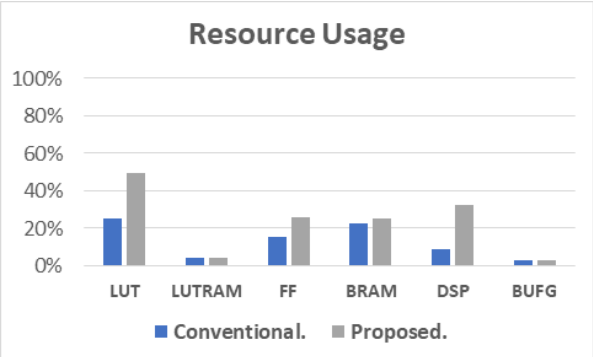


그림 10. Loop unrolling에 따른 자원 사용량 비교
Fig. 10. Comparison of resource usage according to loop unrolling

표 2는 HLS에서 테스트한 latency의 비교 테이블이다. Latency의 경우 출력 채널과 입력 채널에 모두 unrolling을 적용한 결과, 33만~37만 clk으로 기존 병렬화 코드의 207만과 비교 시 약 5.5~6.1배 단축된 것을 확인할 수 있었다. 이렇게 latency가 큰 폭으로 단축된 이유는 제안된 방법에 의해 반복문 제어 오버헤드가 크게 감소되었기 때문이다. 반복문은 반복 횟수에 따라 제어문을 실행하고 반복하는데 많은 오버헤드를 발생시킨다. Loop unrolling은 반복문을 여러 번 실행하는 대신 한 번에 여러 반복문을 처리하도록 변경함으로써 오버헤드를 줄일 수 있었다. 또한 loop unrolling을 통해 반복문의 반복 횟수를 줄여서 파이프라이닝을 더 효율적으로 수행할 수 있기 때문에 연산 효율이 증가했다.

표 3은 기존 순차적 추론 방식 및 병렬적 추론 방식과 함께 loop unrolling 기법을 사용하였을 때의 평균 추론 시간 비교 테이블이다.

표 2. 지연 시간 비교 결과
Table 2. Comparison results of latency times

Latency	Conventional	Proposed
Min	2,075,976	339,472
Max	2,075,976	374,948

표 3. 지연 시간 비교 결과
Table 3. Comparison results of latency times

Method	Inference time(s)
Sequential	0.96796658
Parallel	1.24172873
Loop unrolling	0.28244445

V. 결 론

본 논문에서는 동적 저궤도 위성 네트워크에서 온보드 강화학습 기반 라우팅을 위한 이종 프로세서 기반 추론 병렬화 기술에 대해 제안하였다. 먼저 본 논문에서는 Dueling DQN 기반 강화학습 알고리즘을 동적 환경 및 위성 링크 단절을 고려하여 설계하였다. LEO-SN 환경에서 동적 환경 및 위성 링크 단절은 반드시 고려해야 할 문제이다. 또한 Overlay 기반 HW/SW 동시 설계 및 DMA를 통해 데이터를 처리하였다. 특히 loop unrolling 기법을 통한 파이프라이닝 극대화로 기존의 순차적 추론 및 병렬적 추론보다 향상된 결과를 제시하였으며 이를 latency, 추론 시간, 자원 사용량 등의 관점에서 해석하였다. 우리가 제안한 기법이 저궤도 위성 네트워크 라우팅 알고리즘 뿐 아니라 이종 프로세서 기반 인공지능 등 다양한 분야에서 추가적인 연구가 이루어지길 기대한다.

References

[1] Y.-E. Lee and K.-I. Kim, "Cross-Point Based Routing Protocol in Low Earth Orbit

Communication Networks", Journal of the Conference of the Society of Information Processing, Vol. 28, No. 2, pp. 72-74, 2021.

[2] R. Chen, W.-N. Wang, X. Zhao, and G. Zhao, "Waypoint segment routing algorithm for LEO satellite network", IET Communications, Vol. 16, No. 18, pp. 2133-2144, Nov. 2022. https://doi.org/10.1049/cmu2.12466.

[3] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning", In Proc. of the AAAI conference on artificial intelligence, Vol. 30, No. 1, pp. 2094-2100, Mar. 2016. https://doi.org/10.1609/aaai.v30i1.10295.

[4] B.-S. Roh, M.-H. Han, D.-W. Kum, and K.-S. Jeon, "A Study on the Reinforcement Learning Routing for LEO Satellite Network", Proc. of the Korean Institute of Communication Sciences Conference, pp. 537-538, 2022.

[5] X. Wang, Z. Dai, and Z. Xu, "LEO Satellite Network Routing Algorithm Based on Reinforcement Learning", In 2021 IEEE 4th International Conference on Electronics Technology(ICET), Chengdu, China, pp. 1105-1109, May 2021. https://doi.org/10.1109/ICET51757.2021.9451072.

[6] P. Zuo, C. Wang, Z. Yao, S.g Hou, and H. Jiang, "An Intelligent Routing Algorithm for LEO Satellites Based on Deep Reinforcement Learning", In 2021 IEEE 94th Vehicular Technology Conference(VTC2021-Fall), Norman, OK, USA, pp. 1-5, Sep. 2021. https://doi.org/10.1109/VTC2021-Fall52928.2021.9625325.

[7] P. Zuo, C. Wang, Z. Wei, Z. Li, H. Zhao, and H. Jiang, "Deep Reinforcement Learning Based Load Balancing Routing for LEO Satellite Network", In 2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring), Helsinki, Finland, pp. 1-6, Jun. 2022. https://doi.org/10.1109/VTC2022-Spring54318.2022.9860582.

[8] J. H. Lee and K. Y. Chai, "A Study on the low-earth orbit satellite based non-terrestrial network systems via deep-reinforcement learning", Proc. of the Korean Institute of Communication Sciences Conference, pp. 1306-1307, 2021.

[9] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning", In International conference on machine learning, Vol. 48, pp. 1995-2003, 2016.

[10] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA-Based Neural Network Inference Accelerator", arXiv preprint arXiv:1712.08934, Dec. 2017. https://doi.org/10.48550/arXiv.1712.08934.

[11] J. Wang, W. Tong, and X. Zhi, "Model Parallelism Optimization for CNN FPGA Accelerator", Algorithms 2023, Vol. 16, No. 2, pp. 110, Feb. 2023. https://doi.org/10.3390/a16020110.

[12] D. Kim, Y. Han, H. Lee, Y. Kim, H.-H. Kwon, C. Kim, and W. Choi, "Accelerated Particle Filter With GPU for Real-Time Ballistic Target Tracking", IEEE Access, Vol. 11, pp. 12139-12149, Jan. 2023. https://doi.org/10.1109/ACCESS.2023.3238873.

[13] S. Lee, H. Lee, Y. Kim, J. Kim, and W. Choi, "GPU-accelerated PD-IPM for real-time model predictive control in integrated missile guidance and control systems", Sensors 22, Vol. 22, No. 12, Jun. 2022. https://doi.org/10.3390/s22124512.

[14] Y.-H. Han, H.-C. Lee, H.-H. Gwon, W.-S. Choi, and B.-R. Jeong, "Parallelized Particle Swarm Optimization with GPU for Real-Time Ballistic Target Tracking", IEMEK Journal of Embedded Systems and Applications, Vol. 17, No. 6, pp. 355-365, Dec. 2022. https://doi.org/10.14372/IEMEK.2022.17.6.355.

[15] A. Cigliano and F. Zampognaro, "A Machine Learning approach for routing in satellite Mega-Constellations", In 2020 International Symposium on Advanced Electrical and

Communication Technologies (ISAECT), Marrakech, Morocco, pp. 1-6, Nov. 2020. https://doi.org/10.1109/ISAECT50560.2020.9523672.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning", arXiv:1312.5602, Dec. 2013.

[17] D. Kim and H. Jung, "Performance Analysis of Target Tracking AI based on Unity ML-Agents", Journal of JKIIT, Vol. 19, No. 12, pp. 19-26. Dec. 31, 2021. https://doi.org/10.48550/arXiv.1312.5602.

[18] Python productivity for Zynq(PYNQ Overlays), https://pynq.readthedocs.io/en/v2.0/pynq_overlays.html [accessed: Jun. 2, 2023]

[19] H. Watanabe, M. Tsukada, and H. Matsutani, "An FPGA-based on-device reinforcement learning approach using online sequential learning", In 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Portland, OR, USA, pp. 96-103, Jun. 2021. https://doi.org/10.1109/IPDPSW52791.2021.00022.

[20] M. Vohra and S. Fasciani, "PYNQ-Torch: a framework to develop PyTorch accelerators on the PYNQ platform", In 2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), Ajman, United Arab Emirates, pp. 1-6, Dec. 2019. https://doi.org/10.1109/ISSPIT47144.2019.9001806.

[21] L. Stornaiuolo, M. Santambrogio, and D. Sciuto, "On how to efficiently implement deep learning algorithms on pynq platform", In 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Hong Kong, China, pp. 587-590, Jul. 2018. https://doi.org/10.1109/ISVLSI.2018.00112.

[22] Python productivity for Zynq(PYNQ DMA), https://pynq.readthedocs.io/en/v2.3/pynq_libraries/dma.html [accessed: Jun. 3, 2023]

[23] M. Booshehri, A. Malekpour, and P. Luksch, "An improving method for loop unrolling", arXiv:1308.0698, Aug. 2013. https://doi.org/10.48550/arXiv.1308.0698.

[24] A. Balamane and Z. Taklit, "Using Deep Neural Networks for Estimating Loop Unrolling Factor", arXiv:1911.03991, Nov. 2019. https://doi.org/10.48550/arXiv.1911.03991.

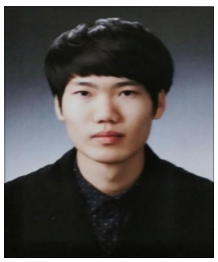
저자소개

김 도 형 (Dohyung Kim)

2018년 3월 ~ 현재 : 금오공과대학교 전자공학부 학사과정

관심분야 : Algorithm Acceleration with GPU/FPGA, Reinforcement Learning

이 민 준 (Minjoon Lee)

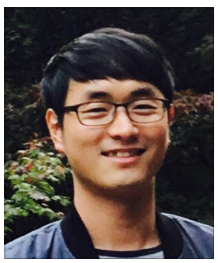


2023년 2월 : 금오공과대학교 전자공학부(공학사)

2023년 3월 ~ 현재 : 금오공과대학교 생산기술연구소 연구원

관심분야 : GPU/FPGA 기반 알고리즘 병렬화/가속화

이 헌 철 (Heoncheol Lee)



2006년 8월 : 경북대학교 전자전기컴퓨터학부(공학사)

2008년 8월 : 서울대학교 전기컴퓨터공학과(공학석사)

2013년 8월 : 서울대학교 전기컴퓨터공학과(공학박사)

2013년 9월 ~ 2019년 2월 : 국방과학연구소 선임연구원

2019년 3월 ~ 현재 : 금오공과대학교 전자공학부 IT융복합공학과 조교수

관심분야 : SLAM, 자율주행, 인공지능, 알고리즘 가속화

원 동 식 (Dongshik Won)

2011년 12월 : 미네소타대학교
전기공학부(전기공학사)
2014년 2월 : 한국과학기술원
전기및전자공학부(공학석사)
2014년 2월~2021년 1월 :
국방과학연구소 선임연구원
2021년 1월~2023년 3월 :
글래스돔코리아 기술이사
2019년 2월 ~ 현재 : 한국과학기술원 박사과정
2023년 4월 ~ 현재 : 텔레픽스 주식회사 이사
관심분야 : 위성 미션 해석 및 설계, 전자광학 시스템,
추적및항법, 우주미션시스템을 위한 인공지능/기계학습

한 명 훈 (Myoung-Hun Han)

2007년 2월 : 중앙대학교
컴퓨터공학(공학사)
2009년 8월 : 중앙대학교
컴퓨터공학(공학석사)
2021년 8월 : 중앙대학교
컴퓨터공학(공학박사)
2014년 10월 ~ 현재 :
국방과학연구소 선임연구원
관심분야 : 네트워크, M&S, LEO, 인공지능