공학석사학위논문

아타리 게임 플레이를 위한 심층 강화학습 모델의 FPGA 가속기 디자인

FPGA Accelerator Design of Deep Reinforcement Learning Model for Playing Atari Games

2018년 2월

서울대학교 대학원

컴퓨터공학부

박 지 영

아타리 게임 플레이를 위한 심층 강화학습 모델의 FPGA 가속기 디자인

FPGA Accelerator Design of Deep Reinforcement Learning Model for Playing Atari Games

지도교수 이 재 진

이 논문을 공학석사학위논문으로 제출함

2017년 10월

서울대학교 대학원

컴퓨터공학부

박 지 영

박지영의 석사학위논문을 인준함

2017년 12월

위 원 장 :	Bernhard Egger	(인)
부위원장 :	이 재 진	(인)
위 원 :	허 충 길	(인)

요약

딥 러닝 기술이 컴퓨터 비전, 자연어 처리 등 여러 어플리케이션에 적용되어 큰 성공을 거두고 있다. 특히 강화학습 알고리즘에 딥 러닝 기술을 접목시킨 심층 강화학습은 아타리 게임을 이미지만으로 학습시킬 수 있게 하면서 주목을 받게 되었다.

이러한 딥 러닝 어플리케이션들은 대부분 많은 계산량을 요구하며 주로 데이터 병렬성을 이용해 GPU로 가속화했는데, 최근 들어 저전력 시스템에 대한 요구가 증가하면서 FPGA에서의 딥 러닝 가속화에 대한 연구가 활발히 진행되고 있다. FPGA는 어플리케이션에 따라 하드웨어를 재구성을 할 수 있는 디바이스이기 때문에 GPU에 비해 전력을 적게 사용할 수 있고, 특히 임베디드 시스템에서 활용도가 높다.

본 논문은 FPGA를 기반으로 심층 강화학습을 가속화하는 방법을 제시했다. 특히 아타리 게임 플레이를 위한 CNN(convolutional neural network) 모델을 가속화했다. CNN을 포함한 대부분의 딥 러닝 어플리케이션들은 계산량 뿐만 아니라 메모리 전송량이 많아서 메모리 대역폭과 온-칩 메모리가 제한된 FPGA에서 효과적으로 가속화하기 어렵다. 그러나 아타리 게임 플레이를 위한 CNN 모델은 FPGA 온-칩 메모리에 모두 들어갈 수 있는 크기이기 때문에 데이터의 재사용을 최대화할 수 있었다. 또한 C++, OpenCL 등의 상위 레벨 언어의 고수준 합성 툴을 사용하지 않고 하드웨어 레벨 언어인 Verilog로 프로그래밍해서 하드웨어 자원을 최대로 활용할 수 있도록 했다. 결과적으로 하드웨어 자원 90%이상을 활용했고, GPU와 비교했을 때 14배 이상 성능 향상을 보였다.

주요어: FPGA, 심층 강화학습, CNN
학번: 2016-21205

목차

그림 목차

표 목차

제 1장 서론

딥 러닝 기술이 컴퓨터 비전, 자연어 처리, 음성 인식 등 여러 어플리케이션에 적용되어 큰 성공을 거두고 있다[1, 2, 3]. 특히 강화학습 알고리즘에 딥 러닝 기술을 접목시킨 심층 강화학습을 통해 아타리 게임을 이미지만으로 학습시킬 수 있게 되면서 강화학습이 주목을 받게 되었다[4, 5]. 강화학습은 학습 데이터에 레이블이 붙은 지도학습과 레이블이 붙지 않고 데이터 사이의 관계를 찾아내는 비지도학습과는 다르게 데이터가 학습 과정 중에 생성된다.

강화학습에서는 주어진 환경 혹은 상태(state)에서 에이전트가 어떤 행동(action)을 취하면 그에 따른 보상(reward)이 주어지고, 이 데이터를 바탕으로 어떤 행동을 취하면 미래에 받게 될 보상이 최대가 되는지에 대한 정책(policy)을 구하는 것이 목표다. 특히 구글 딥마인드에서 발표한 아타리 게임을 플레이하기 위한 심층 강화학습은 강화학습 알고리즘에 CNN(Convolutional Neural Network)을 적용해서 정책 함수를 근사했다. CNN은 컨볼루션 층(convolutional layer)과 완전 연결 층(fully connected layer)으로 구성된 인공신경망의 한 종류다.

이러한 딥 러닝 어플리케이션들은 대부분 많은 계산량을 요구하며 주로 데이터 병렬성을 이용해서 GPU로 가속화했는데[6], 최근들어 저전력 시스템에 대한 요구가 증가하면서 FPGA에서의 딥 러닝 가속화에 대한 연구가 활발히 진행되고 있다[7, 15, 22, 19]. FPGA는 어플리케이션에 따라 하드웨어를 재구성을 할 수 있는 디바이스이기 때문에 GPU에 비해 전력을 적게 사용할 수 있고, 특히 스마트폰이나 로봇 등 임베디드 시스템에서 활용도가 높지만[7], 프로그래밍이 어렵다는 단점이 있다. FPGA 칩의 대표적인 제조사인 인텔, 자일링스에서는 C++, OpenCL[8] 등의 상위 레벨 언어로 프로그래밍을 하면 하드웨어 레벨의 코드를 만들어주는 고수준 합성 툴

[9, 10]을 제공하지만 상위 레벨 언어로는 하드웨어 레벨에서의 동작을 효과적으로 표현하기 어렵고 결과적으로 성능을 최대로 내기 어렵다.

본 논문은 FPGA를 기반으로 심층 강화학습을 가속화하는 방법을 제시했다. 특히 아타리 게임 플레이를 위한 CNN 모델을 가속화했다. CNN을 포함한 대부분의 딥 러닝 어플리케이션들은 계산량 뿐만 아니라 메모리 전송량이 많아서 메모리 대역폭과 온-칩 메모리가 제한된 FPGA에서 효과적으로 가속화하기 어렵다. 특히 CNN의 컨볼루션 층은 계산량에 비해 메모리 전송량이 많지만 완전 연결층은 계산량에 비해 메모리 전송량이 많다. 메모리 전송에 걸리는 시간이 계산 시간에 비해 클 경우 통신과 계산을 중첩해서 최대한 성능을 높일 수 있지만 결국 메모리 대역폭에 의해 성능이 정해지게 되고 하드웨어의 계산 자원이 최대로 활용되지 못하게 된다. 이를 해결하기 위해 데이터 양자화, 압축 등의 연구가 이루어지고 있다[13, 17, 18, 7, 22, 23, 24].

그러나 아타리 게임 플레이를 위한 CNN 모델은 네트워크 파라미터가 FPGA 온-칩 메모리에 모두 들어갈 수 있는 크기이기 때문에 네트워크 파라미터들을 한번만 온-칩 메모리로 가져오면 이후에는 계속해서 재사용할 수 있었다. 이러한 경우 메모리 전송이 성능에 거의 영향을 미치지 않기 때문에 성능을 최대화하기 위해 하드웨어의 유휴 계산 자원이 없도록 하는 것이 중요하다.

모든 데이터 포맷은 부동 소수점을 사용했고 구현에 사용한 FPGA 칩에는 부동 소수점 연산을 할 수 있는 정해진 개수의 DSP(Digital Signal Processor)가 있기 때문에 모든 DSP가 병렬적으로 계산하도록 해야 했다. 특히 C++, OpenCL 등의 상위 레벨 언어의 고수준 합성 툴을 사용하면 데이터의 재사용과 하드웨어 자원 활용을 사용자의 의도대로 할 수 없기 때문에 하드웨어 레벨 언어인 Verilog로 프로그래밍해서 하드웨어 자원을 최대로 활용할 수 있도록 했다.

결과적으로 하드웨어 자원 90%이상을 활용했고, GPU와 비교했을 때 14배 이상 성능 향상을 보였다.

제 2장 관련연구

저전력 시스템에 대한 관심이 높아지면서 FPGA에서 딥 러닝 어플리케이션을 가속화하는 연구가 활발히 진행되고 있다. 그 중에서도 특히 컴퓨터 비전, 자연어 처리, 음성 인식 등 여러 분야에서 의미 있는 결과를 보여준 CNN을 가속화하려는 연구가 많다. FGPA에서 CNN을 가속화하는데 가장 큰 어려움은 제한된 계산 자원과 메모리 대역폭을 효율적으로 사용하는 것이다. 컨볼루션 연산을 어떤 식으로 병렬화 할 것인가에 대한 연구들이 이루어 졌다[24, 25, 26, 14]. [24], [25]에서는 주로 컨볼루션 필터 사이의 병렬성을 이용했고 [26]은 출력 값 사이(inter-output)와 출력 값 내(intra-output)의 병렬성을 이용했다. 본 연구에서는 주로 컨볼루션 필터 사이의 병렬성과 출력 값 사이의 병렬성을 이용했다.

네트워크 사이즈가 큰 모델들은 데이터 재사용을 최대화해서 오프-칩 메모리 접근을 최소화는 것이 필요했다[20, 21, 22]. [20]에서는 중간 결과물을 오프-칩 메모리로 보내지 않기 위해 여러 네트워크 층을 하나로 합쳤다. 또 네트워크 파라미터를 압축하거나[17, 18, 19, 7] 데이터를 양자화해서 고정 소수점을 사용한 연구들이 있다[22, 23, 24]. 고정 소수점을 사용하면 메모리 전송량을 크게 줄일 수 있고 계산 자원도 절약할 수 있다. 특히 [23]에서는 MNIST 숫자 인식 네트워크 학습 단계에서 32-bit 부동 소수점을 사용한 후 추론 단계에서 데이터 양자화를 통해 고정 소수점을 사용했는데 인식률에 거의 차이가 없었다.

온-칩 메모리가 충분히 큰 하드웨어에서 모든 네트워크 파라미터들을 온-칩에 저장해서 오-칩 메모리와의 통신을 없앤 연구들이 있다[15, 16, 11].

제 3장 강화학습

이 장에서는 강화학습에 대한 기본적인 배경과 강화학습에 딥러닝을 적용한 심층 강화학습에 대해 다룬다. 특히 최근 구글 딥마인드에서 발표한 아타리 게임 플레이를 위한 심층 강화학습 모델에 대해 알아본다[4, 5].

강화학습에서는 유한한 시간 동안 에이전트(agent)가 주어진 환경의 상태(state)와 상호작용하면서 매 시간 어떤 행동(action)을 취한다. 게임을 예로 들면 에이전트는 게임 플레이어, 상태는 매 프레임마다 변하는 게임 이미지, 행동은 게임마다 정해진 조작 방법으로 볼 수 있다. 여기서 에이전트 혹은 게임 플레이어는 매 시간 현재 상태에 따라 어떤 행동 혹은 조작을 할지 정해야하는데 이를 정책(policy)이라고 한다. 이처럼 정해진 정책에 따라 에이전트가 행동을 취하면 환경으로부터 행동에 대한 보상(reward)을 받는다. 환경은 에이전트의 행동에 의해 다음 상태로 바뀌게 되고 특정 상태가 될 때까지 이를 반복한다. 이는 게임이 승패가 결정 나거나 정해진 시간이 다 될 때까지 플레이되는 것과 비교할 수 있다.

강화학습의 목표는 에이전트가 현재 상태에서 어떤 행동을 취할 것인지에 대한 정책을 학습해서 환경으로부터 최대의 보상을 받을 수 있도록 하는 것이다.

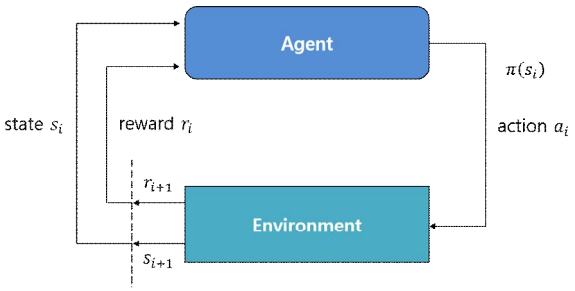

그림 3.1 강화학습

강화학습을 모델링하는 방법에는 크게 값-기반(value-based) 방법과 정책-기반(policy-based) 방법이 있다[5]. 값-기반의 방법은 현재 상태에서 어떤 행동을 취했을 때 미래에 받게 될 보상의 합을 예측하는 행동-값 함수(action-value function)를 정의한다. 각 상태와 행동에서 행동-값 함수의 값이 최대가 되도록 하는 최적의 함수를 찾는 것이 목표다. 정책-기반 방법에서는 현재 상태에서 각 행동들을 취할 확률 값을 계산하는 정책 함수를 직접적으로 정의한다.

심층 강화학습은 이러한 행동 값 함수나 정책 함수를 인공 신경망으로 근사한다. 본 논문에서 FPGA로 가속화한 심층 강화학습 모델의 인공 신경망은 CNN(convolutional neural network)이고 그림 3.2과 같이 2개의 컨볼루션 층과 2개의 완전 연결 층으로 이루어졌다. 네트워크는 크기가 4x84x84인 이미지를 입력으로 받는다. 첫 번째 컨볼루션 층은 크기가 8x8인 필터(filter)가 16개 있고 스트라이드(stride)가 4이다. 두 번째 컨볼루션 층은 크기가 4x4인 필터가 32개 있고 스트라이드가 2이다. 다음 완전 연결 층은 256개의 뉴런(neuron)을 갖고 마지막은 가능한 행동 수만큼의 뉴런을 갖는 완전 연결 층으로 끝난다.

표 3.1은 각 층의 입력 데이터, 파라미터 크기와 계산 량을 나타낸다. 컨볼루션 층은 파라미터 크기에 비해 계산량이 많고 완전 연결 층은 반대로 계산량에 비해 파라미터가 많다.

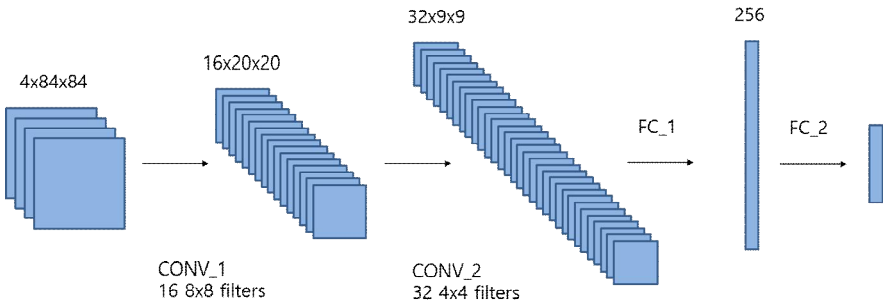

그림 3.2 심층 강화학습 CNN 모델

	입력(KB)	파라미터(KB)	계산량(KFLOP)
CONV_1	111	16	3200
CONV_2	25	32	1296
FC_1	11	2592	1296
FC_2	1	2	1
합계	**148**	**2642**	**5793**

표 3.1 CNN 모델 입력, 파라미터, 계산량

제 4장 구현 상세

이 장에서는 심층 강화학습의 FPGA 구현을 상세하게 살펴본다. 4.1-4.2절에서 시스템의 전체적인 아키텍처와 메모리 시스템에 대해 설명하고 4.3절에서는 모든 부동 소수점 계산에서 사용하는 기본 단위인 매트릭스 연산 모듈에 대해 알아본다. 4.4-4.5절에서는 매트릭스 연산 모듈을 이용해 CNN을 이루는 컨볼루션 층과 완전연결 층이 어떻게 구현되는지 설명한다. 또한 데이터 포맷을 부동 소수점이 아닌 고정 소수점으로 해서 연산을 하면 전력이나 성능 면에서 장점이 있을 수 있는데 이에 대해 4.6절에서 알아본다. 마지막 4.7절에서는 OpenCL 고수준 합성 툴을 이용해 프로그래밍 했을 때의 장단점을 하드웨어 레벨에서 Verilog로 프로그래밍 할 때와 비교한다.

4.1 아키텍처

그림 4.1은 시스템의 전체적인 아키텍처를 보여준다. 호스트 메인 보드에 FPGA 보드가 가속기 형태로 꽂혀 있고 PCIe Gen3 x8 버스를 통해 약 6GB/s로 통신한다. 호스트 CPU는 아타리 게임 시뮬레이터를 실행시키면서 현재 게임의 상태 이미지(state)를 받고 다음 행동(action)을 주는 역할을 한다. 호스트에서 시뮬레이터로부터 받은 현재 상태 이미지와 미리 학습된 네트워크 파라미터들을 FPGA 보드의 DRAM으로 보내면 컨트롤 로직이 데이터를 온-칩 버퍼로 복사한다. 오프-칩 메모리인 DRAM에서 온-칩 버퍼로의 메모리 대역폭은 약 10GB/s인 반면 온-칩 버퍼에서 매트릭스 연산 모듈로의 대역폭은 약 150GB/s로 훨씬 크기 때문에 재사용하는 데이터를 온-칩 버퍼에 최대한 올리게 된다. 메모리 시스템에 대해서는 4.2절에서 더 자세히 다룬다.

온-칩 버퍼에 필요한 데이터가 모두 올라가면 컨트롤 로직이 데

이터를 매트릭스 연산 모듈에 패치하면서 컨볼루션 층과 완전 연결 층의 계산을 시작하고 결과를 다시 호스트 인터페이스로 보낸다. 이를 바탕으로 호스트가 시뮬레이터에 행동을 주고 다음 상태를 다시 FPGA로 보내는 것을 반복한다.

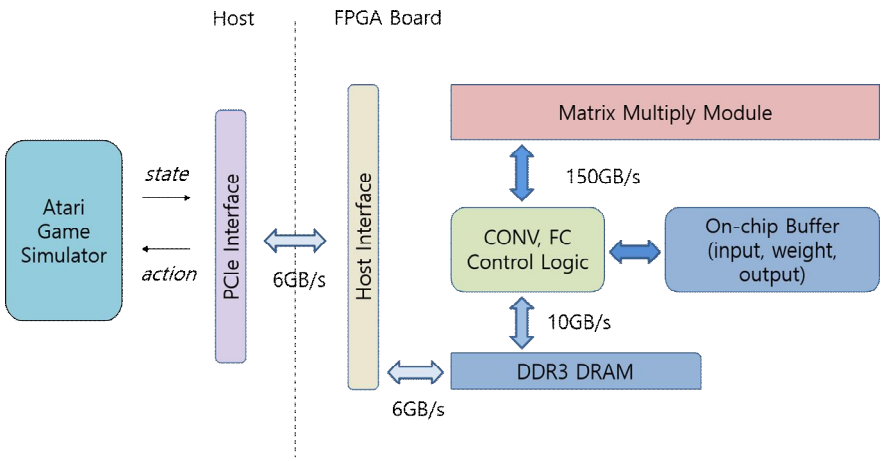

그림 4.1 시스템 아키텍처

4.2 메모리 시스템

메모리 시스템은 크게 오프-칩 메모리인 DRAM과 온-칩 메모리로 이루어졌다. 호스트 인터페이스가 데이터를 DRAM으로 전송하고 나면 필요한 데이터를 온-칩 메모리로 가져온 후 계산을 시작한다. DRAM의 저장 공간은 2GB로 충분히 크지만 온-칩 메모리는 6MB 뿐이기 때문에 효율적으로 사용하는 것이 중요하다. 온-칩 메모리에 패치한 후에 계산에 사용해야하는 데이터에는 크게 입력 값, 네트워크 파라미터, 각 층의 계산 결과 값이 있는데 본 논문에서 구현한 네트워크(표 3.1)는 이 값을 모두 온-칩 메모리에 저장할 수 있다. 특히 네트워크 파라미터 값은 한번만 온-칩 메모리에 저장하고 나면 계속해서 사용할 수 있다.

온-칩 메모리는 온-칩 버퍼 모듈을 만들어서 사용한다. 온-칩 버퍼는 매 사이클마다 읽기(read) 혹은 쓰기(write)를 할 수 있고 총 2개의 포트를 갖고 있다. 또 메모리 워드(word) 크기를 설정해서 한 번에 처리할 데이터의 크기를 버퍼마다 다르게 할 수 있다.

온-칩 버퍼의 레이아웃은 표 4.1와 같다. 1개의 입력(input) 버퍼, CNN 4개 층의 각 파라미터(weight) 버퍼와 출력(output) 버퍼가 있고 모든 데이터는 32-bit 부동 소수점 포맷이다.

	type	word (bit)	depth	size (KB)
CONV_1	input	2688	336	112
	weight	512	256	16
	output	640	320	25
CONV_2	weight	1024	256	32
	output	288	288	10
FC_1	weight	4096	5184	2654
	output	512	16	1
FC_2	weight	1024	256	32
	output	1024	1	1

표 4.1 온-칩 버퍼 레이아웃

4.3 매트릭스 연산 모듈

아타리 게임 플레이를 위한 심층 강화학습 알고리즘은 기본적으로 CNN을 사용해 정책(policy) 함수를 근사한다[4, 5]. CNN은 컨볼루션 층과 완전 연결 층으로 이루어진 네트워크인데 이 두 가지 층의 계산 모두 매트릭스 곱셈 형태로 표현할 수 있다.

그림 4.2는 매트릭스 연산 모듈의 구조를 보여준다. 16개의 부동 소수점 곱셈 연산 유닛(multiplier)과 누산 유닛(accumulator)으로 구성된다. 부동 소수점 곱셈 연산 유닛의 경우 DSP 1개를 필요로 하기 때문에 매트릭스 연산 모듈 1개당 16개의 DSP를 사용하고 전체 256개의 DSP가 FPGA 칩에 내장되어 있기 때문에 총 16개의 매트

릭스 연산 모듈을 생성할 수 있다.

매트릭스 연산 모듈은 매 클락 사이클마다 1개의 실수 값과 16개의 실수 벡터의 값을 각각 곱한 후 16개의 결과 값을 각각 누산 유닛을 통해 누적시킨다. 누적시키는 횟수 또한 입력으로 받아서 계산하려는 매트릭스의 크기에 따라 조절할 수 있도록 했다.

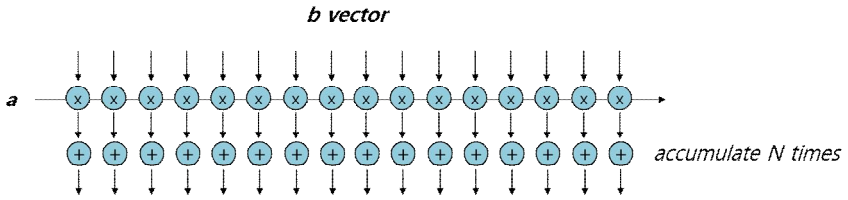

그림 4.2 16-width 매트릭스 연산 모듈

성능을 최대화하려면 매 사이클 연산에 필요한 입력 데이터와 네트워크 파라미터 데이터를 온-칩 버퍼에서 가져 온 후 매트릭스 연산 모듈에 전달해줘야 한다. 특히 컨볼루션 층은 입력 데이터의 접근 패턴이 일정하지 않아서 온-칩 버퍼에서 가져온 데이터를 적절하게 shift한 후에 매트릭스 연산 모듈에 전달해 줘야 한다. 입력 데이터는 주로 이미지 포맷에 맞춰서 온-칩 버퍼에 저장되기 때문에 컨볼루션 연산을 하려는 원도우(window)에 따라서 shift의 크기가 결정된다. 그러나 레지스터의 shift 연산은 FPGA 로직을 많이 사용하기 때문에 라우팅에 성공하려면 최적화가 필요하다. 최적화 방법으로는 가능한 레지스터 shift의 수를 줄이는 것과 shift 레지스터 수 자체를 줄이는 것이 있다. (그림 4.3)

첫 번째 방법은 하나의 레지스터가 여러 크기의 shift를 하지 않도록 하는 것이다. 특히 컨볼루션 층에서는 계산하려는 윈도우의 위치에 따라 입력 데이터 값을 가지고 있는 레지스터를 스트라이드(stride) 크기의 배수만큼 shift해야 하는 경우가 있다. 이때 모든 크기의 shift가 가능하도록 하드웨어를 만들면 비효율적이기 때문에 스트라이드 크기의

shift만 가능하도록 하드웨어를 만들고 나머지 크기의 shift는 한 개의 shift를 반복적으로 사용해서 한다. 그러나 단순히 한 개의 shift를 반복적으로 사용하는 것은 로직을 적게 사용할 수 있지만 성능이 저하될 수 있기 때문에 shift한 레지스터를 재사용할 수 있도록 컨볼루션 연산 순서를 바꿔주는 것이 필요하다. 예를 들어, 첫 번째 윈도우의 컨볼루션 연산을 할 때는 shift가 필요 없고, 두 번째 윈도우는 shift를 한번 해야 한다. 이때 세 번째 윈도우의 컨볼루션 연산을 할 때 두 번째 윈도우 연산을 할 때 사용한 레지스터를 그대로 재사용하면 shift를 한번만 하면 되기 때문에 성능 저하가 생기지 않는다.

두 번째 방법은 shift 레지스터의 수 자체를 줄이는 것이다. 매 사이클 매트릭스 연산 모듈에 필요한 입력 데이터와 네트워크 파라미터 데이터를 전달하려면 레지스터를 2개 사용해서 더블 버퍼링(double buffering) 해야 한다. 이때 2개의 레지스터 모두 shift를 할 수 있도록 하드웨어를 구성하지 않고 한 개만 할 수 있도록 해서 로직 사용량을 줄일 수 있다.

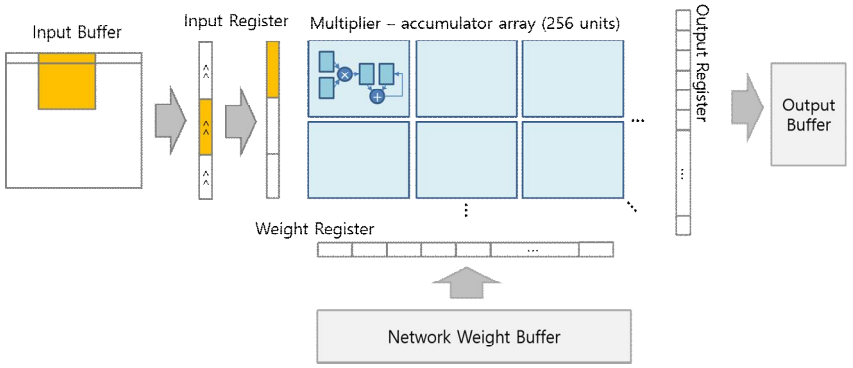

그림 4.3 컨볼루션 연산

4.4 컨볼루션 층

컨볼루션 층의 연산은 입력 데이터와 네트워크 파라미터의 매트릭스 곱셈 연산으로 표현할 수 있다.

- CONV_1: 첫 번째 컨볼루션 층은 크기가 4x84x84인 이미지를 입력으로 받는다. 필터 크기가 8x8이고 입력 데이터의 채널 크기가 4이기 때문에 각 필터의 파라미터 크기는 4x8x8(=256)이다. 또 이러한 필터가 16개(출력 채널) 있기 때문에 전체 파라미터의 크기는 256x16이고 [입력 채널, 필터 높이, 필터 너비, 출력 채널] 순서로 저장된다. 입력 데이터와 파라미터의 온-칩 버퍼 레이아웃은 표 4.1와 같다. 1개 필터의 컨볼루션 연산은 크기가 1x256인 입력 매트릭스와 크기가 256x1인 파라미터 매트릭스의 곱으로 표현할 수 있다. 각 필터들의 컨볼루션 연산은 데이터 병렬성(data parallelism)이 있기 때문에 16개 필터의 컨볼루션 연산을 동시에 할 수 있고 이는 크기가 1x256인 입력 매트릭스와 크기가 256x16인 파라미터 매트릭스의 곱으로 나타낼 수 있다. 이러한 형태의 연산은 4.3절에서 다룬 매트릭스 연산 모듈 1개를 사용해서 계산할 수 있다. 이 경우 매트릭스 연산 모듈에 매 사이클마다 1개의 입력 데이터와 16개의 파라미터 데이터를 주게 되어 총 256 사이클이 소요된다.

- CONV_2: 두 번째 컨볼루션 층의 입력은 크기가 16x20x20이다. 필터 크기가 4x4이고 입력 데이터의 채널 크기가 16이기 때문에 각 필터의 파라미터 크기는 16x4x4(=256)이다. 또 이러한 필터가 32개 있기 때문에 전체 파라미터의 크기는 256x32이고 저장 순서는 첫 번째 컨볼루션 층과 같다. 각 필터들의 컨볼루션 연산은 첫 번째 컨볼루션 층과 마찬가지로 크기가 1x256인 입력 매트릭스와 크기가 256x32인 파라미터 매트릭스의 곱으로 나타낼 수

있는데 이 경우 매트릭스 연산 모듈 2개를 사용해서 병렬화 하면 256 사이클이 소요된다.

위의 설명에서는 모든 필터에 대해 한 번의 컨볼루션 연산을 해서 각각 16x1x1, 32x1x1의 출력 값을 얻게 되는 상황만을 고려했고 매트릭스 연산 모듈을 각각 1, 2개씩만 사용했다. 그러나 실제로는 입력 데이터를 각각 스트라이드 4, 2로 슬라이딩하면서 20x20, 9x9번의 컨볼루션 연산을 해야 한다. 이 연산들은 모두 데이터 병렬성이 있기 때문에 매트릭스 연산 모듈을 모두 사용하도록 병렬화할 수 있다. 이때 첫 번째 컨볼루션 층은 1x256x256x16 매트릭스 곱셈 연산 16개를 동시에 계산하고 이를 총 (20x20)/16(=25)번 반복하면 계산이 끝난다. 두 번째 컨볼루션 층은 1x256x256x32 매트릭스 곱셈 연산 8개를 동시에 계산하고 이를 총 (9x9)/8(=11)번 반복하면 계산이 끝난다.

4.5 완전 연결 층

완전 연결 층의 연산 또한 입력 데이터와 네트워크 파라미터의 매트릭스 곱셈 연산으로 표현할 수 있다.

- FC_1: 첫 번째 완전 연결 층의 입력은 크기가 32x9x9(=2592)이고 출력 값은 256개이다. 1개의 출력 값 계산을 크기가 1x2592인 입력 매트릭스와 크기가 2592x1인 파라미터 매트릭스의 곱으로 나타낼 수 있다. 각 출력 값은 독립적으로 계산할 수 있기 때문에 전체 계산은 크기가 1x2592인 입력 매트릭스와 크기가 2592x256인 파라미터 매트릭스의 곱으로 표현할 수 있다. 이 경우 매트릭스 연산 모듈 16개를 사용할 수 있는데 매 사이클 1개의 입력 데이터와 256개의 파라미터 데이터를 주게 되어 총

2592 사이클이 소요된다.

- FC_2: 마지막 완전 연결 층의 입력 크기가 256이고 출력은 게임마다 다르게 주어지는 행동 혹은 액션 수와 같다. 전체 계산은 첫 번째 완전 연결 층과 마찬가지로 크기가 1x256인 입력 매트릭스와 크기가 256x(액션 수)인 파라미터 매트릭스의 곱으로 나타낼 수 있다. 아타리 게임의 액션 수는 모두 32개 이하이기 때문에 매트릭스 연산 모듈 2개를 사용하면 256 사이클이 소요된다.

첫 번째 완전 연결 층은 매트릭스 연산 모듈을 모두 사용하도록 구현했지만 마지막 완전 연결 층은 계산량이 적어 성능에 미치는 영향이 거의 없기 때문에 최대 2개의 모듈만 사용하도록 구현했다.

4.6 부동 소수점 vs 고정 소수점

실수를 표현할 때 소수점의 위치를 고정시키지 않고 지수로 그 위치를 표시하는 부동 소수점과 달리 고정 소수점은 소수점의 위치를 고정시켜서 실수를 표현한다. 본 논문에서는 이미지, 네트워크 파라미터 등의 데이터 포맷으로 32-bit 부동 소수점을 사용했는데 고정 소수점을 사용하는 많은 연구[22, 23, 24]들이 있다. 고정 소수점은 부동 소수점에 비해 데이터 량을 줄여서 메모리 전송에 드는 비용을 줄일 수 있고 연산도 더 간단해서 리소스 사용량이나 전력을 아낄 수 있지만 부동 소수점에 비해 수의 표현 범위가 좁고 정밀도가 낮아서 학습 단계에서는 쓰기 어렵고 추론 단계에서만 주로 사용된다. 그렇기 때문에 추론 단계에서도 고정 소수점을 사용하려면 부동 소수점으로 학습된 네트워크 파라미터들을 고정 소수점 포

맷으로 변환해야한다. 부동 소수점을 고정 소수점으로 변환하면 기본적으로 데이터의 손실이 생길 수밖에 없기 때문에 손실을 최소화하기 위해 부동 소수점 데이터 값의 범위와 허용 오차 범위를 고려해서 고정 소수점 bit수와 소수점의 위치를 정해야한다. 이 과정은 모든 경우에 좋은 결과를 얻을 수 있는 방법이 없고 어플리케이션마다 데이터의 특성이 다르기 때문에 고정 소수점의 적절한 bit수와 소수점의 위치는 실험적으로 찾아낼 수밖에 없다.

4.7 Verilog vs OpenCL

FPGA에서 딥러닝을 가속화한 대부분의 연구들은 하드웨어 제조사에서 제공하는 고수준 합성(high level synthesis) 툴을 사용해서 프로그래밍 했지만[7, 19, 20, 22] 본 연구에서는 하드웨어 레벨의 언어인 Verilog를 사용해서 구현했다. OpenCL 같은 상위 레벨 언어로 프로그래밍한 후 고수준 합성 툴을 사용하면 프로그래밍이 간단하다는 장점이 있지만 하드웨어 자원을 최대로 활용해서 성능을 최적화하기 어렵다. 특히 딥 러닝 어플리케이션들은 데이터 병렬성이 충분하기 때문에 계산을 위한 하드웨어 자원을 모두 사용할 수도 있지만 고수준 합성 툴을 사용하면 이를 충분히 활용하지 못한다는 것을 알 수 있다[7, 19, 22].

제 5장 실험 결과

FPGA에서 딥 러닝을 가속화한 기존 연구에서는 대부분 네트워크 크기가 큰 어플리케이션을 다뤘고[23, 24, 25] 메모리 전송량이나 전력 효율을 높이기 위해 고정 소수점 연산을 사용했다. 또한 프로그래밍의 편리성을 위해 고수준 합성 툴을 사용하는 경우가 많았다. 본 연구에서는 아타리 게임 플레이를 위한 심층 강화학습 모델을 FPGA에서 가속화 했다. 32-bit 부동 소수점을 사용했고 하드웨어 레벨의 언어인 Verilog로 프로그래밍 했다. 이 모델은 CNN으로 구성됐고 각 층의 입력, 파라미터의 크기와 계산량은 표 3.1과 같다. 입력과 모델의 크기가 약 3MB로 작기 때문에 모두 온-칩 메모리에 올려놓고 계산에 사용할 수 있었다.

5.1 성능 비교 및 분석

심층 강화학습 모델에서 사용한 CNN의 추론 성능을 GPU구현과 비교했고 기존 연구들과도 비교해봤다. 실험에는 Stratix V GX FPGA가 내장된 Terasic DE5-Net 보드와 Geforce GTX Titan X GPU를 사용했다.

CNN 각 층의 계산량과 FPGA, GPU에서의 성능은 표 5.1과 같다. FPGA는 입력 데이터와 네트워크 파라미터가 모두 온-칩 메모리에 올라가 있을 때의 계산 성능이고 GPU는 Tensorflow[12]에서 같은 모델을 계산했을 때의 시간 측정치이다. CONV_1, FC_1의 경우 모든 DSP 자원이 충분히 사용돼서 최대치의 성능이 나오는 반면 CONV_2는 전체 계산이 매트릭스 연산 모듈 수로 나누어떨어지지 않아 낭비되는 부분이 생겨 최적화된 성능을 얻지 못했다. FC_2도 매트릭스 연산 모듈을 2개만 사용해서 구현했지만 계산량이 극히 적어서 전체적인 성능에 미치는 영향이 작았다.

	FPGA (Stratix V)		
	계산량 (KFLOP)	실행 시간 (ms)	계산 성능 (GFLOPS)
INPUT	-	0.018	-
CONV_1	3200	0.064	50.0
CONV_2	1296	0.039	33.2
FC_1	1296	0.026	49.8
FC_2	1	0.003	0.3
Total	**5793**	**0.15**	**38.6**
GPU (Titan X)		**2.11**	**2.75**
Speedup		**14x**	

표 5.1 CNN 각 층의 FPGA, GPU 성능

GPU 구현의 경우 계산 성능이 매우 낮게 나왔는데 이는 네트워크 사이즈가 작고 메모리 전송량에 비해 계산량이 적어 유휴 하드웨어 계산 자원이 많기 때문이다. 실제로 GPU 이용률(utilization)을 측정해보면 15% 정도로 매우 낮았다.

표 5.2는 기존 연구들과의 비교를 보여준다. 대부분의 연구에서 고정 소수점을 사용했고 계산 자원의 개수에 차이가 많이 나기 때문에 정확한 비교에 어려움이 있다. [22]에서는 32-bit 부동 소수점을 사용했지만 메모리 전송에 비해 계산량이 많은 컨볼루션 층만 고려했다. 특히 본 연구에서는 부동 소수점 덧셈, 곱셈 연산을 할 수 있는 하드웨어 자원이 각각 256개 있고 [22]에서는 560개씩 있어서 이론 성능을 계산할 수 있는데 실제 성능 측정치와 비교해보면 본 연구가 하드웨어 자원을 잘 활용했다는 것을 알 수 있다.

	[21]	[22]	[19]	[7]	**본 연구**
년도	2013	2015	2015	2016	2017
FPGA	Virtex-6 VLX240T	Virtex-7 VX485T	Stratix-V GSD8	Zynq XC7Z045	Stratix-V GX
DSP	768	2800	1963	874	256
Clock(MHz)	150	100	120	150	100
정밀도	fixed	32-bit float	8-16bit fixed	16-bit fixed	32-bit float
계산량(GOP)	5.48	1.33	30.9	30.76	0.0058
이론 성능(GOP/s)	-	**112**	-	-	**51.2**
성능(GOP/s)	17	**61.6**	117.8	136.9	**38.6**

표 5.2 기존 연구들과의 성능 비교

5.2 하드웨어 자원 사용률 비교 및 분석

기존 연구들의 대부분은 하드웨어 제조사에서 제공하는 고수준 합성 툴을 사용했다. OpenCL 등의 상위 레벨 언어로 프로그래밍하면 Verilog와 같은 하드웨어 레벨로 프로그래밍 하는 것보다 효율성 면에서는 좋을 수 있지만 하드웨어 자원을 직접 정의하고 어떤 식으로 동작할지 정하는 것보다는 활용률이 떨어질 수밖에 없다. 특히 성능을 최대로 내려면 계산 자원과 메모리 대역폭을 낭비하지 않는 것이 중요하고 특히 딥 러닝 어플리케이션들은 데이터 병렬성이 충분해서 이론적으로는 계산 자원을 최대한 사용할 수 있다. 표 5.3은 본 연구와 기존 연구들의 하드웨어 자원 사용률을 보여준다. [22, 19] 에서는 온-칩 메모리와 로직을 충분히 활용하지 못했고 [7]는 비교적 활용률이 높지만 DSP를 100% 활용하지 못했다.

	Resource	DSP	Block RAM	LUT
[22]	유효량(Available)	2800	2060	303600
	사용량(Used)	2240	1024	186251
	사용률(Utilization)	**80%**	**50%**	**61%**
[19]	유효량(Available)	256	2560	622000
	사용량(Used)	246	1673	153000
	사용률(Utilization)	**96%**	**65%**	**25%**
[7]	유효량(Available)	874	560	218700
	사용량(Used)	780	486	182616
	사용률(Utilization)	**89%**	**87%**	**83%**
본 연구	유효량(Available)	256	2560	234720
	사용량(Used)	256	2454	197215
	사용률(Utilization)	**100%**	**96%**	**84%**

표 5.3 기존 연구들과의 하드웨어 자원 사용률 비교

제 6장 결론

FPGA에서 CNN과 같은 딥 러닝 어플리케이션을 가속화한 기존 연구들은 대부분 데이터 압축이나 양자화를 통해 절대적인 메모리 전송량을 최소화하거나 메모리 재사용을 최대화하는 것에 집중해야했다. 그러나 데이터 압축이나 양자화는 학습 단계에서는 사용하기 어렵고 추론 단계에서도 적용하기 쉽지 않다. 결국 FPGA의 제한된 메모리 대역폭에서 많은 데이터 전송을 필요로 하는 CNN을 가속화하기는 쉽지 않다.

본 연구에서는 딥 러닝 어플리케이션 중에서도 아타리 게임 플레이를 위한 심층 강화학습 모델을 가속화했다. 이 모델은 CNN 기반이지만 네트워크 크기가 크지 않고 입력 데이터나 파라미터를 온-칩 메모리에 모두 저장할 수 있기 때문에 계산을 최적화하는 것이 중요했다. 모든 계산은 FPGA 칩에 내장된 DSP를 최대로 활용했고 32-bit 부동소수점을 사용했다. 특히 하드웨어 레벨 언어인 Verilog로 직접 프로그래밍해서 하드웨어 계산 자원의 사용률을 최대로 할 수 있었다.

실험에서 GPU 구현과 성능을 비교한 결과 추론 성능이 14배 이상 높았다. GPU 구현의 경우 네트워크 사이즈가 충분히 크지 않으면 메모리 전송량에 비해 계산량이 너무 작아 하드웨어 계산 자원의 이용률이 너무 낮았다. 반면 FPGA 구현은 하드웨어 계산 자원을 100% 사용했고 온-칩 메모리와 하드웨어 로직 또한 90% 가까이 활용해서 이론치에 가까운 성능을 낼 수 있었다.

기존의 연구들은 대부분 CNN의 추론을 가속화하는 것에 집중했는데 강화학습 알고리즘은 에이전트가 추론과 학습을 동시에 진행해야 하는 경우가 있을 수 있다. 본 연구도 추론만 고려했지만 32-bit 부동소수점 연산을 사용했기 때문에 학습 과정에 그대로 적용한 후속 연구를 진행할 수 있겠다.

참고문헌

[1] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 1106?1114, 2012.

[2] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In Proc. ICASSP, 2013.

[3] The IBM 2016 English Conversational Telephone Speech Recognition System

[4] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. Nature, 518(7540):529?533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

[5] V. Mnih, A. Puigdomenech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. ArXiv preprint arXiv:1602.01783, 2016.

[6] https://developer.nvidia.com/cudnn

[7] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang and Huazhong Yang, “Going deeper with embedded fpga platform for convolutional neural network,” in Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016, pp. 26 - 35.

[8] www.khronos.org/opencl
[9] https://software.intel.com/en-us/intel-opencl
[10] https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html
[11] Jinhwan Park, Wonyong Sung, “FPGA based implementation of deep neural networks using on-chip memory only,” ICASSP ’16: the 41st IEEE International Conference on Acousics, Speech and Signal Processing (ICASSP), 2016. 03
[12] https://www.tensorflow.org/
[13] Yann LeCun, J. S. Denker, S. Solla, R. E. Howard, and L. D. Jackel, “Optimal Brain Damage,” NIPS ’89: Advanced in Neural Information Processing Systems (NIPS), 1990
[14] Clement Farabet, Cyril Poulet, Jefferson Y. Han, and Yann LeCun, “CNP: An FPGA-based Processor for Convolutional Networks,” FPL ’09: 2009 International Conference on Field Programmable Logic and Applications (FPL), 2009. 09, pp32-37
[15] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam, “ShiDianNao: Shifting vision processing closer to the sensor,” ISCA ’15: Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA), 2015. 06, pp92-104
[16] Tao Luo, Shaoli Liu, Ling Li, Yuqing Wang, Shijin Zhang, Tianshi Chen, Zhiwei Xu, Olivier Temam, and Yunji Chen, “DaDianNao: A Neural Network Supercomputer,” IEEE Transactions on Computers, vol. 66, no. 1, pp. 73-88, 2017
[17] SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE
[18] Stephen J. Hanson and Lorien Y. Pratt, “Comparing Biases for Minimal Network Construction with Back-Propagation,” NIPS ’88: Advances in Neural Information Processing Systems, 1988

[19] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jaeson Seo, and Yu Cao, “Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks,” FPGA ’16: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2016. 02, pp16-25
[20] Manoj Alwani, Han Chan, Michael Ferdman, and Peter Milder, “Fused-Layer CNN Accelerators,” MICRO ’16: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016. 10,
[21] Maurice Peemen, Arnaud A.A. Setio, Bart Mesman, and Henk Corporaal, “Memory-centric accelerator design for Convolutional Neural Network,” ICCD ’13: 2013 IEEE 31st International Conference on Computer Design (ICCD), 2013. 11, pp13-19
[22] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong, “Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks,” FPGA ’15: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2015. 02, pp161-170
[23] Large-Scale FPGA-based Convolutional Networks - Yann LeCun
[24] Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto, and Hans Peter Graf, “A Massively Parallel Coprocessor for Convolutional Neural Networks,” ASAP ’09: 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2009. 07, pp53-60
[25] Srihari Cadambi, Abhinandan Majumdar, Michela Becchi, Srimat Chakradhar, and Hans Peter Graf, “A programmable parallel accelerator for learning and classification,” PACT ’10:

Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2010. 09, pp273-284
[26] Srimat Chakradhar, Murugan Sankaradas Venkata Jakkula, and Srihari Cadambi, “A dynamically configurable coprocessor for convolutional neural networks,” ISCA ’10: Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA), 2010. 06, pp247-257

Abstract

FPGA Accelerator Design of Deep Reinforcement Learning Model for Playing Atari Games

Park Jiyoung
Department of Computer Science
and Engineering
The Graduate School
Seoul National University

Deep learning technology has been successfully applied to various applications such as computer vision and natural language processing. In particular, deep reinforcement learning combining deep learning with reinforcement learning algorithms has attracted attention by allowing Atari games to be learned with images alone.

Most of these deep learning applications require large amounts of computation and have been accelerated on GPUs, mainly using data parallelism. In recent years, there has been a growing demand for low-power systems, and studies are actively being conducted to accelerate deep learning on FPGAs. Since FPGA is a device that can reconfigure hardware according to applications, it can use less power than GPU, especially in embedded system.

This paper presents a method to accelerate deep reinforcement learning based on FPGA. In particular, we accelerated the CNN (convolutional neural network) model for playing Atari game. Most deep learning applications, including CNN, have large memory footprint as well as computational complexity, making it difficult to accelerate effectively on FPGAs with limited memory bandwidth and on-chip memory. However, the CNN model for playing Atari was able to fit in the FPGA on-chip memory, minimizing data transfer. As a result, it utilized more than 90% of hardware resources, and showed a 14x performance improvement compared to GPU.

Keywords: FPGA, Deep Reinforcement Learning, CNN

Student Number: 2016-21205