ETRI

2018
Electronics and
Telecommunications
Trends

Deep Neural Network &
Object Recognition 특집

딥 뉴럴 네트워크 지원을 위한 뉴로모픽 소프트웨어 플랫폼 기술 동향

Trends in Neuromorphic Software Platform for Deep Neural Network

유미선 (Misun Yu, msyu@etri.re.kr) 고신뢰 CPS 연구그룹 책임연구원
하영목 (Youngmok Ha, ymha@etri.re.kr) 고신뢰 CPS 연구그룹 연구원
김태호 (Taeho Kim, taehokim@etri.re.kr) 고신뢰 CPS 연구그룹 책임연구원/PL

Deep neural networks (DNNs) are widely used in various domains such as speech and image recognition. DNN software frameworks such as Tensorflow and Caffe contributed to the popularity of DNN because of their easy programming environment. In addition, many companies are developing neuromorphic processing units (NPU) such as Tensor Processing Units (TPUs) and Graphical Processing Units (GPUs) to improve the performance of DNN processing. However, there is a large gap between NPUs and DNN software frameworks due to the lack of framework support for various NPUs. A bridge for the gap is a DNN software platform including DNN optimized compilers and DNN libraries. In this paper, we review the technical trends of DNN software platforms.

* DOI: 10.22648/ETRI.2018.J.330402

* 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임[No.1711073912, 인공지능 시스템을 위한 뉴로모픽 컴퓨팅 SW 플랫폼 기술 개발].

Ⅰ. 개요

뉴로모픽 컴퓨터는 인간의 뇌를 모방한 컴퓨터로서 효율적인 인공지능 정보처리를 가능하게 한다. 이러한 뉴로모픽 컴퓨터를 구성하는 데에는 크게 두 가지 접근 방식이 있다.

첫 번째는 인간의 뇌가 패턴을 인식하는 방식에서 착안한 인공 뉴럴 네트워크(ANN: Artificial Neural Network) 혹은 딥 뉴럴 네트워크(DNN: Deep Neural Network)를 기반으로 하는 방식이다. 이 방식에서는 대규모 데이터 집합이 가지는 특성들을 효과적으로 모델링하기 위해 비선형 출력을 내는 노드들을 여러 층으로 쌓아 연결한다.

또 다른 접근 방식은 뇌의 생물학적 특성을 직접적으로 모방해 보다 원천적인 사고 능력과 뇌의 활동을 구현하는 방식이다. 이 방식은 주로 스파이킹 뉴럴 네트워크(SNN: Spiking Neural Network)에 쓰이고 있다. 〈표 1〉은 DNN과 SNN 기술에 대한 주요 특징 비교를 보여준다. 〈표 1〉에서 볼 수 있듯, SNN은 초 저전력, 강 인공지능을 구현할 것으로 기대되는 차세대 기술이다.

본고에서는 활발하게 연구가 이루어져 실용적으로 사용할 만한 성능을 보여주기 시작하고 있는 DNN을 위한 뉴로모픽 소프트웨어 플랫폼에 대해 기술한다.

DNN의 인기의 한 요인은 개발 프레임워크가 잘 갖추어져 있어 DNN 응용 소프트웨어 개발이 매우 간단하다는 것이다. 대표적인 DNN 응용 개발 프레임워크로는 Tensorflow[1], Caffe[2], PyTorch[3] 등이 있다.

또 하나의 요인으로는 DNN 처리 성능을 향상시키기 위한 DNN 가속 하드웨어인 뉴럴 프로세싱 유닛(NPU: Neural Processing Unit)이 다양하게 제공된다는 것이다. 대표적인 DNN 가속 하드웨어로는 GPU(Graphical Processing Unit)가 있다. GPU는 본래 DNN을 위해서 개발된 것은 아니지만, 그 효용성이 입증됨에 따라 현재 가장 많이 사용되고 있는 DNN 가속 하드웨어이다. DNN을 효율적으로 처리하기 위하여 GPU를 대체하는 하드웨어 개발이 진행되고 있는데, 구글의 클라우드 서비스를 위한 TPU(Tensor Processing Unit)와 DNN 가속 기능이 내장된 중국 화웨이의 모바일 프로세서인 Kirin970가 대표적인 예이다.

〈표 1〉 DNN과 SNN 기술의 주요 특징 비교

ANN/DNN	SNN
– 가속 연산 지원 – 저전력 – 시각 인지 등에 강점 – 약 인공지능	– 계산 신경학 모델 – 이벤트 기반 연산 지원 – 초 저전력 – Life-long 러닝 – 강 인공지능

NPU의 활용성을 높이기 위해서는, 다양한 NPU를 지원하는 DNN 응용 개발 프레임워크가 반드시 필요하지만 이들 사이에는 큰 간극이 존재한다. 즉, 현재 개발자들이 주로 사용하고 있는 DNN 응용 개발 프레임워크가 다양한 종류의 NPU를 지원하지 못하고 있다. 이들 간극을 메우는 역할을 하는 것이 DNN 최적화 컴파일러 및 DNN 지원 라이브러리 등을 포함하고 있는 뉴로모픽 소프트웨어 플랫폼이다.

(그림 1)은 DNN 소프트웨어 플랫폼이 제공하는 기능을 보여준다. DNN 소프트웨어 플렛폼은 DNN연산을 최적화 할 수 있는 기능들을 포함해야 하고, 이를 라이브러리 형태로 제공해 손쉬운 개발을 지원해야 한다. DNN의 특성 상 훈련과 실행에 많은 연산이 필요하기

(그림 1) DNN 개발을 위한 소프트웨어 플랫폼의 요소

때문에 이러한 최적 연산 지원은 DNN 응용의 성능을 향상시키는 데 큰 역할을 수행한다. 더하여, DNN 소프트웨어 플랫폼은 DNN 프레임워크의 성능에 직접적인 영향을 미칠 수 있는 메모리 사용률, 전력 사용률, 실시간성 만족 등에 대한 최적화 요구사항을 만족시켜야 한다. 따라서 이러한 최적화 요구사항에 능동적으로 대응할 수 있는 컴파일러 및 운영체제 기술에 대한 지원도 포함하고 있어야 한다. 그리고 다양한 하드웨어를 지원하기 위한 하드웨어 지원 프레임워크 또한 필요하며 향후 개발될 하드웨어의 기능을 세부적인 지식이 없이도 쉽게 개발에 활용할 수 있도록 지원하는 컴파일러 수준의 확장성 높은 디바이스 드라이버 프레임워크도 제공해야 한다.

본 동향 분석 원고에서는 이러한 DNN 소프트웨어 플랫폼의 구성요소를 다음과 같은 순서로 설명한다. Ⅱ장은 딥뉴럴 네트워크 소프트웨어 플랫폼 기술의 구성을 사례를 통해 기술하고, Ⅲ장에 딥 뉴럴 네트워크 지원을 위한 라이브러리 기술을 설명한다. 그리고 Ⅳ장에서는 딥 뉴럴 네트워크 소프트웨어 플랫폼의 발전 방향에 대해 기술하며 본고를 마친다.

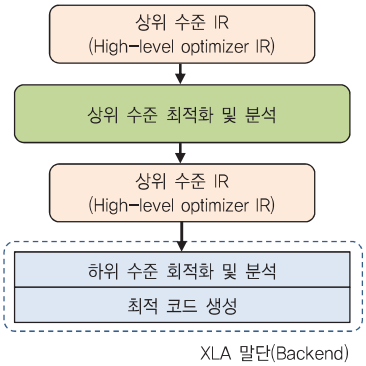
Ⅱ. 딥 뉴럴 네트워크 소프트웨어 플랫폼

DNN 소프트웨어 플랫폼은 DNN 프레임워크와 NPU 사이에 존재하며, DNN을 NPU에 최적화된 코드로 생성하는 기능을 제공하는 컴파일러 기반 통합 소프트웨어 스택이다. 구체적으로, 이러한 소프트웨어 플랫폼들은 중간언어(IR: Intermediate Representation) 제공과 컴파일러 수준 최적화를 통해, Tensorflow[1], Caffe[2], PyTorch[3] 등으로 구현된 DNN을 각 하드웨어 플랫폼 특성에 맞게 최적화하고, 그에 해당하는 실행 코드를 생성하는 역할을 수행한다. 현재 개발되고 있는 DNN 소프트웨어 플랫폼으로는 XLA[4], TVM[5], Glow[6], NNVM[7], DLVM[8] 등이 있다. 이 장에선 XLA, TVM, 그리고 Glow에 대해 소개한다.

1. XLA

XLA(Accelerated Linear Algebra)는 구글의 Tensorflow 컴파일러 프레임워크로, 커스텀 연산자 의존도 감소와 실행 시간, 메모리 사용, 이식성에 대한 개선을 목적으로 개발되었다. 이 컴파일러 프레임워크는 상위 수준 DNN 표현인 Tensorflow에서 수행하지 못하는 하드웨어 수준 최적화를 지원함으로써, 메모리 사용과 실행 속도 측면에서 DNN의 성능을 높인다.

XLA가 수행하는 작업은 하드웨어에 독립적인 상위 수준 최적화 및 분석, 대상 하드웨어에 의존적인 하위 수준 최적화 및 분석, 그리고 대상 하드웨어에 대한 최적 코드 생성으로 이루어져 있다. (그림 2)는 XLA에서 수행하는 컴파일 과정을 보여준다. 여기서, 상위 수준 최적화 및 분석은 HLO(High-Level Optimizer) IR 또는 간단하게 HLO라고 불리우는 IR로 표현된 Tensorflow 그래프에 대한 연산자 합성, CSE(Common Sub-expression Elimination) 와 같은 최적화, 메모리 할당을 위한 버퍼 분석을 포함하고 있다. 최적 코드 생성과 함께 XLA 말단(backend)을 형성하고 있는 하위 수준 최적화 및 분석은 특정 하드웨어 프로그래밍 모델에 적합한 연산자 합성 또는 계산 분할 등의 최적화를 수행한다. XLA 최적 코드 생성은 이식성을 높이기 위해



(그림 2) XLA 컴파일 과정

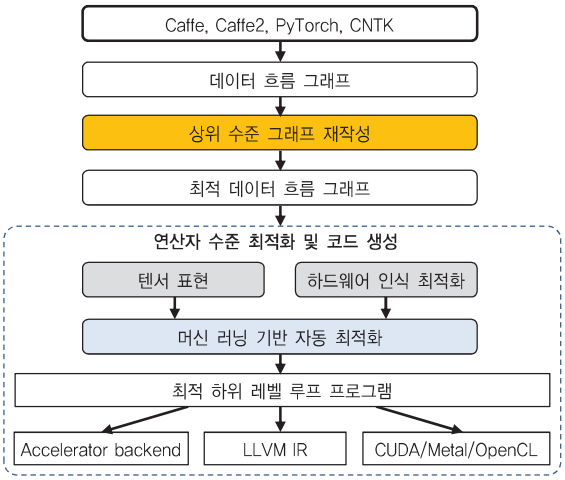
[출처] Tensorflow, "XLA: Domain-specific compiler for linear algebra to optimizes tensorflow computations," https://www.tensorflow.org/performance/xla/, CC BY 3.0.

LLVM[9]을 활용하고 있으며 여러 CPU ISA(Instruction Set Architecture)들과 NVIDIA GPU들에 대한 명령어 수준 최적화 및 코드 생성을 지원한다.

XLA는 JIT(Just-In-Time) 또는 AOT(Ahead-Of-Time) 컴파일을 모두 지원한다. JIT 컴파일은 사용자가 생성한 Tensorflow 그래프에 대한 분석과 커널 합성을 통한 메모리 대역폭 요구 및 실행 시간에 대한 최적화를 제공할 수 있고, AOT 컴파일은 실행 코드 크기와 실행 시간에 대한 최적화를 제공할 수 있다. XLA는 x86-64에 대해서는 JIT와 AOT 모두를 지원하며, NVIDIA GPU와 ARM에 대해서는 각각 JIT와 AOT만을 지원하고 있다.

2. TVM

특정 DNN 프레임워크가 특정 NPU에 종속되지 않도록 지원하는 대표적인 방법으로 TVM이 있다. TVM(Tensor Virtual Machine)은 Halide[10] 계산과 스케줄을 분리해 병렬성을 높이는 파이프라인 개념을 차용한 컴파일러 기반 DNN 소프트웨어 플랫폼으로 Caffe, PyTorch, CNTK[11] 등으로 구현된 DNN과 NPU를 이어주는 역할을 한다. 이 플랫폼은 텐서 표현 언어와 기계 학습 기반 자동 최적화를 특징으로 한다.



(그림 3) TVM 개요[5]

(그림 3)은 TVM이 수행하는 전체적인 컴파일 과정을 보여준다. TVM은 그래프 수준 DNN 모델 최적화를 수행한 후, 그 결과로 생성된 최적 그래프를 연산자 수준 최적화 모듈에 전달한다. 이 연산자 수준 최적화 모듈에서는 연산자를 텐서 표현 언어를 이용하여 표현하고 대상 하드웨어에 맞게 최적화하는 작업을 수행한다. 하드웨어에 탑재하기 위한 최적화에는 기계 학습 기반 자동 최적화를 사용한다.

가. 상위 수준 그래프 최적화

TVM 상위 수준 그래프 최적화는 연산자 합성과 데이터 레이아웃 변환에서 차별성을 가진다. TVM에서는 연산자 합성을 위하여 그래프 연산자를 네 개 카테고리(injective, reduction, complex out-fusable, opaque)로 분류하고, 이들에 대한 연산자 합성 규칙을 정의한다. 예를 들어, injective에 해당하는 연산자들은 함께 합성되어 또 다른 injective 연산자로 표현될 수 있고 opaque에 해당하는 연산자는 다른 연산자들과 합성될 수 없다. 이러한 합성은 주어진 그래프를 단순화한다. 데이터 레이아웃 변환은 주어진 그래프에 있는 텐서를 저장하는 방법을 정의한다. 이 변환은 메모리 계층과 메모리 구조 등에 맞게 하드웨어에서 빠르게 참조될 수 있는 형태로 데이터를 메모리에 배치하는 역할을 한다.

나. 텐서 연산 생성

TVM은 어떤 연산자에 대하여 각 하드웨어에 유효한 여러 구현 방법을 고려한 후, 가장 적절한 구현 방법을 찾아 효율적인 코드를 생성한다. TVM은 여기에 Halide 계산/스케줄 분리 원칙을 차용하며, 텐서 표현과 하위 수준 코드를 매핑하는 스케줄러를 사용한다. 이 스케줄러는 프로그램의 논리적 동치를 유지하는 기본 변환으로 이루어지며, 여러 하드웨어에 대한 이식성을 높이기 위한 다양한 변환들을 고려하여 매핑을 수행한다.

다. 자동 최적화

실행시간에서의 최적 스케줄을 찾기 위해 TVM은 루프 순서 조정이나 면적 분할(Tiling)과 관련된 여러 가지 스케줄 파라미터를 고려한다. 스케줄 수가 많고, 그 파라미터가 다양한 경우 스케줄 공간(Schedule space)이 수십억 개에 달할 정도로 매우 커질 수 있다. TVM에선 최적 스케줄을 찾기 위해 기계 학습 모델을 사용한다.

3. Glow

Glow(Graph-Loweing)는 PyTorch를 비롯한 여러 DNN 프레임워크들을 지원하기 위하여 페이스북에서 개발한 컴파일러 기반 소프트웨어 스택이다. Glow는 DNN 표현 방식 중의 하나인 ONNX[12]를 사용하여, 다양한 DNN 프레임워크를 지원하고 있다. 또한, 데이터 흐름 그래프를 상위 수준 IR과 하위 수준 IR로 분리하여 두 단계로 표현하는 점에서 기존 방식보다 상세한 최적화를 가능하게 한다.

Glow에서 사용되는 IR들 중 상위 수준 IR은 하드웨어 독립적인 최적화를 위한 것으로, 데이터 흐름 노드를 기반으로 한 그래프 표현 방식을 사용하고 있다. 이 IR에서는 입력과 출력에 있어 어떤 텐서 타입을 사용하는지 명시해야 하며, 그에 대한 타입 확인은 컴파일러가 한다.

상위 수준 IR 표현이 완료되면, Glow는 변수 최적화 등을 마친 후 '노드 세분화(Node lowering)'를 수행한다. 컴파일러는 이 단계에서 상위 수준 연산자들을 낮은 수준 선형 대수 연산 노드들로 나누어진다. 이렇게 생성된 그래프는 명령어 스케줄 결정에 영향을 주거나 특정 하드웨어에 의존적인 하위 수준 추가 최적화를 수행한다.

위 단계가 끝나게 되면, 코드들은 IRGen으로 불리우는 하위 수준 IR로 표현되게 된다. 이 하위 수준 IR은 'in-memory form'을 사용한다. 하위 수준 IR은 상위 수준 IR과 같이 입·출력에 연관된 텐서 타입을 명시하게 되어있다. 이 단계에서, 각 상위 노드들은 하나 또는 여러 개의 명령어로 번역되며, 이러한 표현은 상위 수준 그래프 포맷에서는 할 수 없는 주소 참조 기반 메모리 최적화를 가능하게 한다. 추가로, 이러한 명령어 기반 IR은 컴파일러가 메모리 수행 지연을 숨기기 위한 스케줄 생성을 가능하게 한다.

위 단계가 모두 완료된 후에는 하드웨어에 맞춘 최적화된 코드가 생성된다.

Ⅲ. 딥 뉴럴 네트워크 지원 라이브러리

최신 DNN 알고리즘들은 수천만에서 많게는 1억 개 이상의 파라미터를 사용한다. 이러한 수많은 파라미터를 계산하고 훈련하기 위해서는 대량의 행렬 곱셈(Matrix Multiplication)이 필요하다. 예를 들어, 이미지 인식 및 분류에 있어서 높은 인식률을 나타내고 있는 합성곱 신경망(CNN: Convolutional Neural Net)의 경우, 완전 연결 계층(FC: Fully-Connected Layer)을 구현하기 위해 수많은 행렬 연산을 반복적으로 수행해야 한다. 그러므로 DNN 응용의 수행 속도를 향상시키지 위해서는 행렬 연산을 최적화할 수 있는 하드웨어와 라이브러리의의 사용 필수적이다.

그래픽 프로세싱 유닛(GPU: Graphic Processing Unit)는 이러한 행렬 연산을 효율적으로 수행할 수 있는 하드웨어 구조를 제공한다. GPU에서 응용 프로그램을 수행할 때, 해당 응용 프로그램은 여러 개의 스레드 블록으로 구성되고, 또 하나의 스레드 블록은 여러 개의 스레드로 구성되게 된다. 그리고 하나의 하나의 스레드 블록은 하나의 스트리밍 멀티프로세서(SM: Streaming Multiprocessor)에 할당된다. 이때, 하나의 SM에 속한 스레드 블록 내의 스레드들은 스트리밍 프로세서(SP: Streaming Processor)에 의해 동시에 수행된다. 즉, GPU를 이용하면 행렬 연산을 여러 스레드로 나누어 병렬적으로 수행하는 것이 가능하게 된다.

GPU를 이용하여 DNN 응용 개발이 가능하도록 지원

하는 대표적인 플랫폼으로는 NVIDIA에서 개발한 Unified Device Architecture(CUDA)[13] 기반 플랫폼과 Khronos에서 개발한 Open Computing Language (OpenCL)[14] 기반 플랫폼이 있다. CUDA 기반 플랫폼은 NVIDIA에서 개발한 GPU에서만 사용 가능한 폐쇄적인 플랫폼이며 OpenCL 기반 플랫폼은 다양한 하드웨어 지원을 목적으로 만들어진 플랫폼이다.

위에서 설명한 두 종류의 GPU 기반의 DNN 개발 플랫폼(CUDA 기반 플랫폼과 OpenCL기반 플랫폼)은 DNN 응용을 개발하기 위해 GPU 하드웨어와 프로그래밍 언어 및 컴파일을 지원하고 있으며, 현재 주로 사용되고 있는 DNN 응용 소프트웨어 프레임워크가 이들 라이브러리를 기반으로 구축되어 있기 때문에 라이브러리와 호환성을 유지하는 것이 필요하다. 따라서, 본 문서에서는 현재 범용적으로 사용되고 있는 GPU 기반 플랫폼에 대해 중점적으로 설명한다.

다음 절에서는 위에서 설명한 두 가지 GPU 기반 플랫폼과 각 플랫폼상에서 효율적인 DNN 응용을 개발하기 위해 사용되는 선형대수 및 DNN 라이브러리에 대해 설명한다.

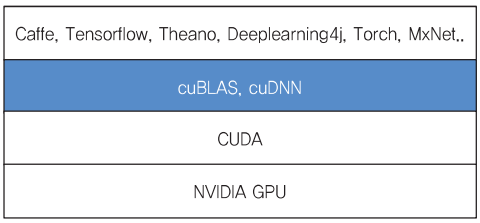
1. BLAS

BLAS(Basic Linear Algebra Subprograms)[15]는 행렬곱, 벡터 내적과 같은 선형 대수 연산 수행을 위한 함수들의 정의로, 대수 연산 라이브러리의 업계 표준이며 고성능 컴퓨팅을 요구하는 다양한 분야에서 가장 널리 사용되고 있다.

BLAS를 CUDA나 OpenCL과 같은 특정 플랫폼상에서 구현한 BLAS 라이브러리는 행렬 곱셈 성능이 중요한 부분을 차지하는 DNN 응용 개발의 필수 요소이다.

2. CUDA 기반 플랫폼

CUDA는 NVIDIA GPU만을 위해 설계되었기 때문에 타사의 GPU와는 함께 사용할 수 없다. 그러나 C/C++, Python과 같은 익숙한 프로그래밍 언어를 이용해 고성능 DNN 프로그램을 구현할 수 있도록 지원하기 때문에 현재 대표적인 DNN 컴퓨팅 플랫폼으로 자리잡고 있다.



(그림 4) CUDA 기반 딥 뉴럴 네트워크 응용 개발 프레임워크

(그림 4)는 CUDA 기반 플랫폼 기반의 DNN 응용 개발 프레임워크의 구조를 보여준다. 파란색으로 표시된 부분은 CUDA를 이용해 고속 DNN 연산을 지원하는 라이브러리인 cuBlas[16]와 cuDNN[17]이다. Caffe나 Tensorflow와 같은 주요 딥러닝 프레임워크들이 모두 CUDA 기반 라이브러리인 cuBlas와 cuDNN을 사용하고 있다.

가. cuBLAS

cuBlas(CUDA Basic Linear Algebra Subprogram)는 NVIDIA에서 개발한 CUDA 가속 기능을 포함하고 있는 BLAS 라이브러리이다. cuBlas는 최근에 NVIDIA GPU에 추가된 행렬 계산 전용 프로세서인 텐서 코어(Tensor Core)를 사용한 행렬 곱 연산을 지원하고 있다. 텐서 코어는 행렬 곱셈+덧셈(Matrix-Multiply-and-Accumulate) 장치로서 80개의 SM이 장착된 V100 GPU의 경우 한 사이클에 8,196번의 부동소수점(Floating Point) 연산이 가능한 장치이다. 그러므로 NVIDIA GPU를 장착한 디바이스 상에서 DNN 응용을 개발하는 경우 cuBlas를 사용하면 상당한 성능 가속 효과를 볼 수 있다.

나. cuDNN

cuDNN은 CUDA 기반 DNN 라이브러리이다. 즉,

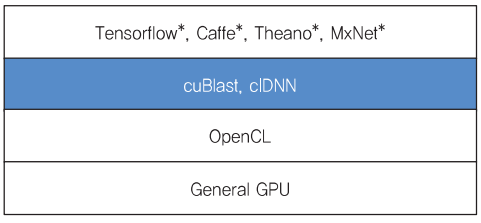
cuDNN은 DNN 응용 개발에 자주 사용되는 기능들을 멀티 GPU, 멀티 스레드를을 기반으로 구현한 라이브러리이다. 컨볼루션(Convolution), 풀링(Pooling), 표준화(Nomarlization), 활성화(Activation)와 같은 기능에 대한 포워드 및 백워드 학습을 지원하며 cuBlas와 마찬가지로 CNN과 RNN(Recurrent Neural Networks)을 가속하는데 텐서 코어를 사용하고 있다.

3. OpenCL 기반 플랫폼

OpenCL은 CUDA와 달리 특정 제조사의 GPU 상에서 동작하는 프로그램 작성을 지원하는 것이 목적이 아니라 다양한 제조사가 만든 이기종 하드웨어 플랫폼(CPU, GPU, DSP, FPGA 등) 상에서 동작하는 프로그램 작성을 지원하기 위해 만들어졌다.

(그림 5)는 OpenCL 플랫폼 기반의 DNN 응용 개발 프레임워크의 구조를 보여준다. 파란색으로 표시된 부분은 OpenCL을 이용하여 고속 딥러닝 연산을 지원하는 라이브러리들인 clBlast[18]와 clDNN[19]이다.

OpenCL은 개방된 프레임워크로 향후 확장 및 발전 가능성이 높기는 하나 현재 대부분의 딥러닝 프레임워크가 OpenCL 기반 플랫폼을 완전히 지원하지 못하고 있으며, 성능 또한 제한이 있는 것으로 알려지고 있다. 그러나 OpenCL은 개방된 개발 플랫폼을 기반으로 하고 있으며 여러 다양한 하드웨어 플랫폼을 대상으로 꾸준히 연구와 개발이 이루어지고 있으므로 향후 하드웨어 지원 및 성능 면에서 발전 가능성이 높을 것으로 예상된다.



(그림 5) OpenCL 기반 딥뉴럴네트워크 응용 개발 프레임워크

가. clBlast

clBlast는 GPU를 포함한 다양한 플랫폼을 지원하기 위해 C++11과 OpenCL로 작성된 BLAS 라이브러리이다. cuBlas와 달리 특정 제조사의 하드웨어에 의존적이지 않으며 오픈소스로 공개되어 있다.

clBlast의 가장 큰 장점은 특정 하드웨어에 최적화된 OpenCL 커널을 생성하기 위한 복잡하고 다양한 파라미터 튜닝을 자동으로 수행하는 도구인 auto-tuner[20]가 제공된다는 것이다. 이와 함께, auto-tuner를 이용하여 특정 딥러닝 응용의 특성(예: 뉴럴 네트워크 계층 설정)에 적합하게 clBlast의 설정을 변경하여 해당 응용에 최적화된 성능을 내도록 할 수 있다.

clBlast는 OpenCL 기반 Tensorflow와 Caffe의 실험 버전에 사용되고 있으며 libgpuarray0 라이브러리와 함께 PyTorch에 사용되고 있다.

나. clDNN

clDNN은 인텔이 개발한 OpenCL 기반의 오픈소스 DNN 라이브러리이다. 현재는 인텔 GPU 상에서만 동작하도록 구현되어 있으며 C/C++을 이용한 CNN을 구현을 지원한다. clDNN과 함께 배포되는 OpenVino 도구를 이용하면 Caffe나 Tensorflow와 같은 주요 딥러닝 프레임워크에서 훈련된 모델을 사용 가능하다.

Ⅳ. 결론

대규모 데이터와 결합된 DNN 기술은 정보통신(ICT) 산업뿐 아니라 제조업·서비스업 등 다양한 산업 분야에서 자동화와 최적화를 한 단계 상승시킬 수 있는 핵심적인 기술로 주목받고 있다. 본 원고에서는 DNN 응용의 성능과 효율성을 향상시킬 수 있는 소프트웨어 플랫폼, 특히, 컴파일러에 기반한 DNN 소프트웨어 플랫폼과 라이브러리 동향에 대해 살펴보았다. 현재 이들 소프트웨어 플랫폼은 특정 하드웨어에 의존적이나, 오픈소스 기

반의 개방형 구조를 기반으로 지원 및 하드웨어 기능을 점진적으로 확장해 가는 추세이다. 이러한 개방형 플랫폼을 기반으로 향후 다양한 하드웨어 플랫폼에서 최적화된 성능을 지원하는 DNN 컴파일러와 라이브러리가 지속적으로 개발될 것으로 예상된다. 또한, 서로 다른 특성(메모리, 배터리, 계산 속도 등)을 지닌 이기종 하드웨어가 끊임없이 개발되고 있으므로, 다양한 하드웨어 상에서 공통적으로 동작 가능한 DNN 응용 개발 지원을 위한 소프트웨어 플랫폼과 이를 운영하기 위한 운영체제 기술의 개발도 더욱 활성화될 것으로 예상된다.

약어 정리

ANN	Artificial Neural Network
CPU	Central Processing Unit
CSE	Common Subexpression Elimination
DNN	Deep Neural Network
DSP	Digital Signal Processors
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HW	Hardware
ICT	Information and Communications Technology
IR	Intermediate Representation
ISA	Instruction Set Architecture
NPU	Neural Processing Unig
RNN	Recurrent Neural Networks
SNN	Spiking Neural Network
SW	Software
TVM	Tensor Virtual Machine

참고문헌

[1] M. Abadi et al., "TensorFlow: A System for Large-scale Machine Learning," In *Proc. USENIX Conf. Oper. Syst. Des. Implementation*, Berkeley, CA, USA, 2016, pp. 265-283.

[2] Y. Jia et al., "Caffe: Convolutional Architecture for Fast Feature Embedding," In *Proc. ACM Int. Conf. Multimedia*, New York, NY, USA, 2014, pp. 675-678.

[3] A. Paszke et al., "Automatic Differentiation in PyTorch," In *Conf. Neural Inform. Process. Syst.*, Long Beach, CA, USA, 2017.

[4] Tensorflow, "XLA: Domain-Specific Compiler for Linear Algebra to Optimizes Tensorflow Computations," https://www.tensorflow.org/performance/xla/

[5] T. Chen et al., "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," Feb. 2018, arXiv: 1802.04799v2.

[6] N. Rotem et al., "Glow: Graph Lowering Compiler Techniques for Neural Networks," May 2018, arXiv: 1805.00907.

[7] P.G. Allen et al., "NNVM Compiler: Open Compiler for AI Frameworks," 2017. http://tvmlang.org/2017/10/06/nnvm-compiler-announcement.html

[8] R. Wei, L. Schwartz, and V. Adve, "DLVM: A Modern Compiler Infrastructure for Deep Learning Systems," Nov. 2017, arXiv: 1711.03016v5.

[9] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," In *Proc. Int. Symp. Code Generation Optimization*, San Jose, CA, USA, Mar. 20-24, 2004, pp. 75-86.

[10] J. Ragan-Kelley et al., "Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines," In *Proc. ACM SIGPLAN Conf.* Programming *Language Des. Implement.*, Seattle, WA, USA, June 2013, pp. 519-530.

[11] F. Seide and A. Agarwal, "CNTK: Microsoft'S Opensource Deep-Learning Toolkit," In *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 2135-2135.

[12] ONNX. https://onnx.ai/

[13] D. Kirk, "NVIDIA Cuda Software and GPU Parallel Computing Architecture," In *Proc. Int. Symp. Memory Manag.*, Montreal, Canada, Oct. 2007, pp. 103-104.

[14] J.E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Comput. Sci. Eng.*, vol. 12, no. 3, 2010, pp. 66-73.

[15] L.S. Blackford et al., "An Updated Set of Basic Linear Algebra Subprograms (BLAS)," *ACM Trans. Math. Softw.*, vol. 28, no. 2, 2002, pp. 135-151.

[16] NVIDIA, "cuBLAS," https://developer.nvidia.com/cublas

[17] NVIDIA, "cuDNN," https://developer.nvidia.com/cudnn

[18] C. Nugteren, "CLBlast: A Tuned OpenCL BLAS Library," In *Proc. Int. Workshop OpenCL*, Oxford, UK, May 2018,

pp. 5:1-5:10.

[19] Intel, "Intel Open Sources OpenCL Deep Neural Network Library for Intel GPUs," 2017. https://software.intel.com/en-us/forums/opencl/topic/735271

[20] C. Nugteren and V. Codreanu, "CLTune: A Generic Auto-Tuner for OpenCL Kernels," In *Proc. Int. Symp. Embedded Multicore/Many-Core Syst.-on-Chip*, Turin, Italy, Sept. 23-25, 2015, pp. 195-202.

[21] ANACONDA, "Library to Manipulate Arrays on GPU," https://anaconda.org/anaconda/libgpuarray