임베디드 시스템을 위한 딥러닝 연산 가속 기술 동향

홍승태 한국전자통신연구원 선임연구원

1. 머리말

최근 IT 기술의 발전으로 이미지 분류 및 글자/객체 인식 등 다양한 분야에서 딥러닝(deep learning) 기술이 활용되고 있다. 아울러 최근 심층신경망(DNN, Deep Neural Network)에 관한 연구가 활발히 이루어짐에 따라 딥러닝을 활용한 다양한 응용이 실생활에서 널리 사용되고 있다.

딥러닝 연산의 대부분은 벡터 또는 행렬 연산으로 구성된다. 이러한 벡터 또는 행렬 연산과 같은 일반적인 선형 대수 연산을 수행하기 위한 루틴 집합으로서 BLAS(Basic Linear Algebra Subprograms)가 존재한다. BLAS는 Netlib에서 정의한 명세(specification)가 일반적으로 사용되지만, CPU 또는 GPU와 같이 각 HW에 최적화된 다양한 BLAS 라이브러리가 존재한다. 아울러, BLAS 라이브러리와 별개로 cuDNN(NVIDIA CUDA Deep Neural Network Library)과 같이 심층신경망을 구성하는 각 계층(layer)에 대해서 GPU 기반 가속 연산을 수행하기 위한 라이브러리가 존재한다.

한편, 최근 스마트폰과 같은 온디바이스(on-device) 형태의 임베디드 시스템(embedded system)이 대중화됨에 따라, PC/Server 플랫폼뿐만 아니라 임베디드 시스템을 기반으로 딥러닝을 수행하기 위한 연구가 활발히 진행 중이다. 임베디드 시스템은 일반적인 PC/Server 플랫폼에 비해 매우 적은 용량의 메모리와 고정된 크기의 저장 장치, 그리고 저전력의 프로세스를 탑재하고 있다. 또한, 임베디드 시스템은 CPU와 GPU, 그리고 메모리 구조에서 PC/Server 플랫폼과 연산 처리 구조가 상이할 뿐만 아니라, 시스템의 사용 목적에 따라 다양한 종류의 HW 플랫폼이 존재한다. 또한, 대부분의 임베디드 시스템은 CUDA와 같은 NVIDIA GPU 기반의 병렬 처리 언어를 지원하지 않기 때문에, PC/Server 플랫폼 기반의 BLAS 라이브러리 및 GPU 기반 딥러닝 연산 가속 라이브러리를 활용하지 못하는 한계점이 존재한다. 따라서 임베디

드 시스템에서 딥러닝 연산을 효율적으로 수행하기 위해서는 임베디드 시스템에 특화된 딥러닝 연산 가속 기술을 활용해야 한다.

따라서 본고에서는 BLAS와 같은 딥러닝 연산 가속 기술의 연구 동향을 파악하고, 임베디드 시스템을 위한 병렬 처리 기술 및 딥러닝 연산 가속 기술 동향을 살펴본다.

2. 딥러닝 연산 가속 기술 연구 동향

2.1 딥러닝 연산 기본 구조

딥러닝 수행을 위한 심층신경망은 기본적으로 한 개의 입력 계층(input layer), 여러 개의 은닉 계층(hidden layer), 그리고 한 개의 출력 계층(output layer)으로 구성된다. 입력 계층은 학습 또는 추론을 수행하기 위한 입력 데이터를 가지고 있으며, 일반적으로 배열 형태의 부동 소수점 자료형이 입력 데이터로 사용된다. 은닉 계층은 입력 데이터에 대한 추론 연산을 수행하며, 이를 위한 가중치(weight)와 편향(bias)을 가지고 있다. 출력 계층은 입력에 대한 추론 확률을 값으로 가진다.

딥러닝의 연산은 기본적으로 순전파(forward propagation) 및 역전파(backward propagation)로 구성되어 있다. 순전파는 입력 계층에서 출력 계층 방향으로 연산을 수행하며, 역전파는 출력 계층에서 입력 계층 방향으로 연산을 수행하면서 가중치와 편향을 보정한다. 추론 시에는 순전파만 수행되며, 학습 시에는 순전파 이후 역전파를 수행한다. 역전파에서는 순전파에서 획득한 추론 결과와 실제 값과의 오차를 통해 가중치와 편향에 대한 경사도(gradient)를 계산한다. 역전파에서는 계산된 경사도를 이용하여 추론 결과와 실제 값과의 오차가 최소가 되도록 가중치와 편향을 보정한다. 이와 같이 심층신경망에서 순전파 수행 시에는 입력 데이터와 가중치, 그리고 편향값에 대한 행렬곱이 주로 이루어지며, 역전파 수행 시에는 은닉 계층의 가중치와 편향에 대한 경사도 계산이 주로 이루어진다. 이처럼 심층신경망을 사용하는 딥러닝 연산의 대부분은 벡터 또는 행렬 연산으로 구성된다.

2.2 딥러닝 연산 가속 기술

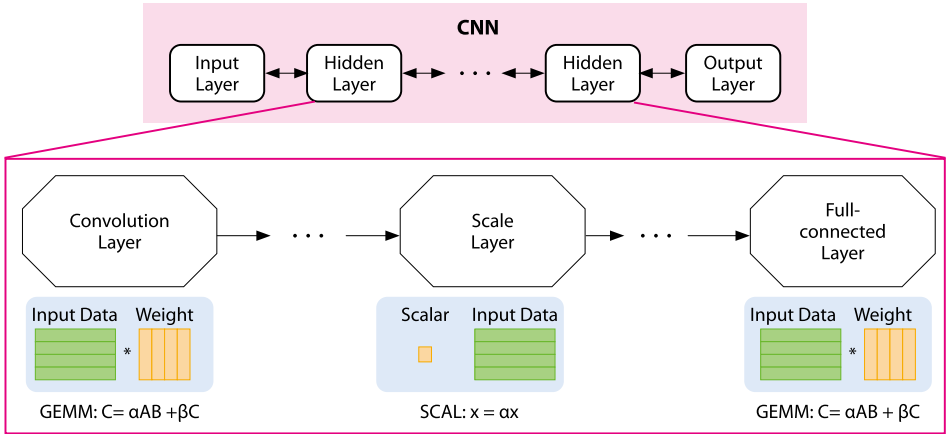
BLAS는 연산 데이터 형태에 따라 벡터-스칼라 연산 루틴(LEVEL 1), 행렬-벡터 연산 루틴(LEVEL 2), 행렬-행렬 연산 루틴(LEVEL 3)로 구분된다. BLAS의 각 루틴은 연산의 정밀도에 따라 단정밀도(single precision), 배정밀도(double precision), 단정밀 복소수(complex precision), 배정밀 복소수(double complex precision)로 구분된다. 이중 심층신경망에서는 주로 단정밀도 연산이 사용되며, 단정밀도 연산

<표 1> 딥러닝에서 사용되는 대표적인 BLAS 연산 루틴

루틴	연산명	연산 내용	비고
LEVEL 1	SCAL	$x = \alpha x$	x는 벡터, α는 스칼라
	AXPY	$x = \alpha x + y$	x, y는 벡터, α는 스칼라
LEVEL 2	GEMV	$y = \alpha Ax + \beta y$	x, y는 벡터, A는 행렬 α, β는 스칼라
LEVEL 3	GEMM	$C = \alpha AB + \beta C$	A, B, C는 행렬 α, β는 스칼라

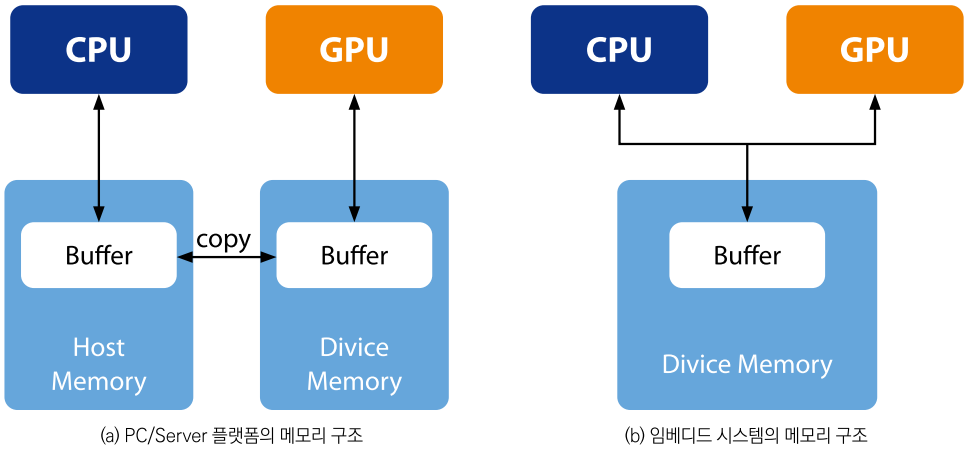
은 LEVEL 1에 14개, LEVEL 2에 16개, LEVEL 3 에 6개의 연산이 존재한다.

<표 1>은 딥러닝에서 사용되는 대표적인 BLAS 연산 루틴을 나타내며, [그림 1]은 대표적인 딥러닝 기술인 합성곱 신경망(CNN, Convolutional Neural Network)에서 BLAS 연산이 사용되는 예시를 나타낸다. AXPY는 스케일 계층(scale layer)과 요소곱 계층(eltwise layer), SCAL은 스케일 계층, GEMV는 배치 정규화 계층(batch normalization layer), 그리고 GEMM은 합성곱 계층(convolution layer)과 완전하게 연결된 계층(fully connected layer)에서 주로 사용된다. 한편, GEMV와 GEMM은 행렬의 전치(transpose) 여부에 따라 연산이 구분된다. 즉, GEMV는 행렬 A의 전치 여부(true or false)에 따라 2개의 연산(GEMV_T, GEMV_N)으로 구분되며, GEMM은 행렬 A, B의 전치 여부에 따라 4개의 연산(GEMM_NN, GEMM_NT, GEMM_TN, GEMM_TT)으로 구분된다.



※ 출처: "합성곱 신경망 기본 연산 인터페이스 구현 사례(기술보고서)", 한국정보통신기술협회, TTAR-00137

[그림 1] 합성곱 신경망에서 BLAS 연산 사용 예시



(a) PC/Server 플랫폼의 메모리 구조

(b) 임베디드 시스템의 메모리 구조

[그림 2] PC/Server 플랫폼과 임베디드 시스템의 메모리 구조 차이

3. 임베디드 기반 딥러닝 연산 가속 기술 동향

3.1 임베디드 시스템 기본 구조

임베디드 시스템은 일반적인 PC/Server 플랫폼에 비해 CPU와 GPU, 그리고 메모리 구조에서 상이한 특징을 가진다. [그림 1(a)]와 같이 PC는 일반적으로 CPU와 GPU가 독립된 별도의 메모리를 가지고 있으며, GPU에서 딥러닝 연산을 수행하기 위해서는 CPU에서 생성한 버퍼를 GPU에 복사해야 한다. 아울러, GPU에서 수행된 연산 결과를 가져오기 위해서도 위와 동일하게 GPU에서 CPU로 복사해야 한다. 이에 비해, 임베디드 시스템은 [그림 1(b)]와 같이 CPU와 GPU가 하나의 메모리 공간을 공유하는 통합 메모리(unified memory) 구조로 되어있다. 이러한 특징을 이용하여 임베디드에 특화된 딥러닝 연산 기술을 활용할 경우, CPU와 GPU 간에 데이터 전송 없이 데이터를 공유할 수 있다.

한편, 임베디드 시스템은 앞서 설명한 바와 같이 CUDA와 같은 NVIDIA GPU 기반의 병렬처리 언어를 지원하지 않으며, 활용 목적에 따라 다양한 이기종 HW 플랫폼으로 구성되어 있다. 따라서 임베디드 시스템에서 행렬 연산과 같은 딥러닝 연산을 가속화하기 위해서는 OpenCL과 같은 병렬 처리 언어가 필수적이다. OpenCL은 크로노스 그룹에서 관리되는 개방형 범용 병렬 컴퓨팅 프레임워크로서, CPU와 GPU, 그 외 기타 프로세서들을 포함하는 다양한 이기종 하드웨어 환경에서 병렬 처리를 지원한다. 특히 OpenCL은 NVIDIA와 같은 특정 하드웨어에 종속되지 않기 때문에 다양한 종류의 하드웨어 플랫폼에서 실행 가능하다.

OpenCL은 호스트(host)인 CPU에서 실행되는 호스트 프로그램(host program)과 병렬처리 장치인 GPU에서 실행되는 커널(kernel)로 구성된다. 호스트 프로그램은 커널을 실행하기 위한 인덱스(index) 공간을 정의하며, 커널의 각 인스턴스(instance), 즉 워크 아이템(work-item)은 인덱스 공간의 각 점에 해당되어 실행된다. 워크 그룹(work-group)은 여러 워크 아이템들로 구성되며, 모든 워크 그룹은 서로 동일한 크기를 가진다. 워크 그룹의 크기는 전체 데이터 크기를 의미하는 전역 워크 그룹 크기(global work-group size)와 워크 그룹당 워크 아이템의 크기를 의미하는 지역 워크 그룹 크기(local work-group size)로 구분된다.

인덱스 공간은 동일한 크기를 가진 워크 그룹으로 분할되며, 각 워크 그룹은 병렬 처리 장치 내의 하나의 연산 유닛(compute unit)에서 병렬적으로 수행된다. 따라서 OpenCL에서 연산 수행 시 성능을 극대화하기 위해서는 각 워크 그룹을 연산 유닛에서 최대한 병렬적으로 수행할 수 있도록 지역 워크 그룹 크기를 설정해야 한다.

OpenCL은 데이터를 저장하기 위한 객체로써 메모리 객체(memory object)를 사용하며, OpenCL의 메모리 객체는 호스트인 CPU에서 생성되어, 사용자가 정의한 호스트 프로그램을 통해 GPU에서 수행되는 커널에 할당된다. OpenCL은 메모리 객체에 대한 매핑(mapping) 및 매핑 해제(unmapping) API를 통해 앞서 설명한 통합 메모리 구조를 이용한 데이터 공유 기능을 지원한다. 임베디드 시스템에서 이러한 OpenCL의 기능을 활용할 경우, CPU와 GPU 간 데이터 복사가 불필요하므로 딥러닝 연산 속도를 크게 향상할 수 있다.

3.2 OpenCL 기반 딥러닝 연산 가속 기술

OpenCL은 개방형 범용 병렬 컴퓨팅 프레임워

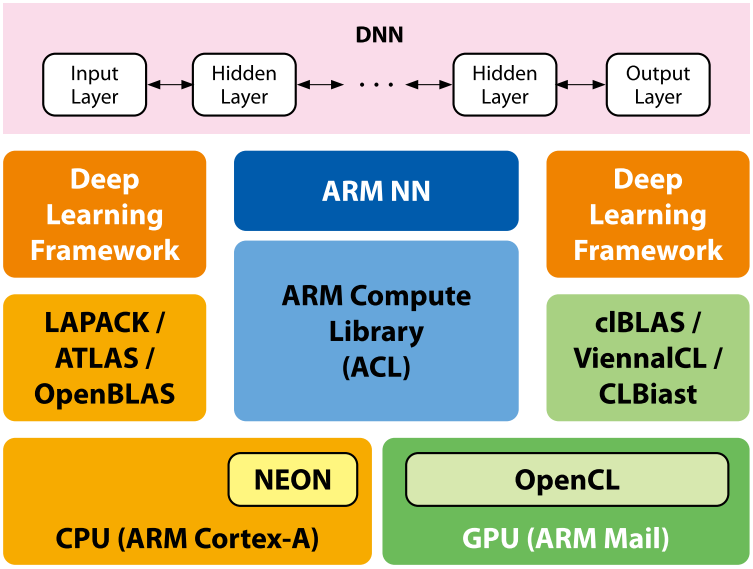
크이기 때문에 주로 오픈소스를 통해 BLAS 라이브러리에 대한 연구가 진행 중이다. 대표적인 OpenCL 기반 BLAS 라이브러리로는 clBLAS, ViennaCL, CLBlast가 존재하며, 현재 CLBlast가 가장 성능이 우수하다고 알려져 있다.

CLBlast는 C++11으로 작성된 OpenCL 기반 BLAS 라이브러리로써, 다양한 HW 플랫폼에서 OpenCL의 성능을 최대한 활용할 수 있도록 설계되었다. 특히, CLBlast는 다양한 하드웨어에서 최적화된 성능을 보장하기 위해, OpenCL의 파라미터(parameter)를 자동으로 튜닝(tuning)하는 도구인 오토 튜너(auto-tuner)를 제공한다. CLBlast는 BLAS의 모든 연산 루틴과 정밀도를 지원하지는 않으나, 딥러닝에서 사용하는 대부분의 주요 연산 루틴 및 정밀도를 지원한다. 아울러, 최근 딥러닝 분야에서 연산 비용 감소를 위해 주로 사용하는 정밀도인 반정밀도(half precision)를 추가적으로 지원한다.

한편, cuDNN과 같이 계층 레벨의 순전파 및 역전파 연산을 지원하는 OpenCL 기반 딥러닝 연산 가속 기술로는 Arm Compute Library(ACL)과 clDNN이 있다. ACL은 ARM에서 관리하고 있는 ARM CPU 및 GPU를 위한 딥러닝 가속 라이브러리이다. ACL은 ARM CPU 및 GPU에만 동작이 가능하며, ArmNN을 이용할 경우 Caffe 또는 Tensorflow 모델을 사용할 수 있으나 ACL 자체만으로는 기존 딥러닝 프레임워크와 연동이 어려운 한계점이 존재한다. clDNN은 인텔이 개발한 OpenCL 기반의 딥러닝 연산 가속 라이브러리로써, 현재는 인텔 CPU에 내장된 GPU에서만 동작하도록 구현되어 있다.

3.3 OpenCL 활용 딥러닝 프레임워크

현재 가장 대표적인 딥러닝 프레임워크로는 PyTorch, Tensorflow, Keras, Theano, Caffe, MXnet, Deeplearning4j, Darknet이 있다. 이러한 딥러닝 프레임워크는 딥러닝 연산 가속을 위해 CUDA 및 cuDNN을 사용하기 때문에 임베디드 시스템에서는 활용이 어렵다. 따라서 임베디드 시스템에서 딥러닝 프레임워크가 효율적



[그림 3] 임베디드 시스템에서 딥러닝 수행을 위한 구성도

으로 수행되기 위해서는 앞서 설명한 OpenCL 기반의 딥러닝 연산 가속 기술이 연동되어야 한다. 현재 OpenCL 기반의 딥러닝 연산 가속 기술이 지원되는 딥러닝 프레임워크로는 OpenCL Caffe, DeepCL, TensorFlow Lite, Darknet on OpenCL이 존재한다.

OpenCL Caffe는 Caffe의 GitHub 저장소에서 관리되기 때문에 기존 Caffe의 인터페이스를 그대로 활용 가능한 장점이 존재하며, ViennalCL, clBLAS 그리고 CLBlast와 연동하여 딥러닝 연산을 수행할 수 있다. DeepCL은 EasyCL과 clBLAS와 연동 가능하며, C++ 및 Python API를 제공한다. TensorFlow Lite는 Tensorflow에서 학습한 모델을 경량화하여 스마트폰과 같은 안드로이드(Android) 기반의 운영체제를 사용하는 임베디드 시스템에서 추론을 수행하기 위한 Tensorflow의 서브 구성요소이다. Tensorflow Lite는 Google의 Neural Networks API (NNAPI), GPU 대리자(delegates)를 통해 딥러닝 연산을 가속화한다. Darknet on OpenCL은 CUDA 및 cuDNN를 활용하는 기존 Darknet을 OpenCL을 활용할 수 있도록 수정한 버전이다.

[그림 2]는 OpenCL을 활용할 수 있는 대표적인 HW 플랫폼인 ARM CPU 및 GPU를 탑재한 임베디드 시스템에서 딥러닝 수행을 위한 구성도를 나타낸다. 현재 대부분의 CPU 기반 BLAS 라이브러리는 ARM 기반의 CPU를 지원하며, OpenBLAS는 자체 스레드(thread) 기반 병렬 처리를 통해 멀티 코어 기반 CPU에 최적화된 성능을 제공한다. 아울러, ARM의 Mali GPU를 지원하는 OpenCL 기반 BLAS 라이브러로는 clBLAS, ViennalCL, CLBlast를 활용할 수 있다. ACL의 경우 OpenCL 기반 딥러닝 연산 가속과 함께 ARM CPU 기반 단일 명령 다중 데이터 처리(SIMD, Single Instruction Multiple Data) 기술인 NEON 기반 연산 가속 기술을 지원한다. 임베디드 시스템을 위한 딥러닝 프레임워크로는 OpenCL Caffe, DeepCL, TensorFlow Lite, 그리고 Darknet on OpenCL 등이 있으며, TensorFlow Lite의 경우 안드로이드 환경에서만 OpenCL 기반 딥러닝 연산 가속을 지원한다.

4. 맺음말

임베디드 시스템은 일반적인 PC/Server 플랫폼에 비해 시스템 자원이 제한적일 뿐만 아니라, 시스템 구조이 상이하다. 따라서, 임베디드 시스템에서 딥러닝을 효율적으로 수행하기 위해서는 임베디드 시스템의 특징을 파악하고, 임베디드 시스템에 최적화된 딥러닝 연산 가속 기술을 활용해야 한다.

본고에서는 기존의 딥러닝 연산 기술을 살펴보고, 임베디드 시스템의 구조적 특징과 함께 임베디드 시스템을 위한 딥러닝 연산 가속 기술을 살펴보았다. 특히, 임베디드 시스템에서 GPU 기반 딥러닝 연산 가속을 위한 OpenCL 및 OpenCL 기반 연산 가속 기술의 연구 동향을 살펴보았다.

OpenCL 기반 딥러닝 연산 가속 기술의 경우 개방형 병렬 처리 프레임워크인 OpenCL의 특징상 오픈소스를 기반으로 주로 연구가 진행 중이며, 현재 모든 딥러닝 프레임워크가 OpenCL을 지원하지 못하는 한계점이 존재하나 향후 확장 및 발전 가능성이 크다고 할 수 있다.

※ 본 연구는 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No.2017-0-00142,스마트기기를 위한 온디바이스 지능형 정보처리 가속화 SW플랫폼 기술 개발)

참고문헌

[1] 홍승태 외, '합성곱 신경망 기본 연산 인터페이스 구현 사례(기술보고서)', 한국정보통신기술협회, 기술보고서, TTAR-10.0137, 2020. 10

[2] http://www.netlib.org/blas/

[3] http://www.netlib.org/lapack/

[4] http://math-atlas.sourceforge.net/

[5] https://www.openblas.net/

[6] https://www.khronos.org/opencl/

[7] https://developer.nvidia.com/cublas

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

[9] https://github.com/NVIDIA/cutlass

[10] https://github.com/clMathLibraries/clBLAS

[11] http://viennacl.sourceforge.net

[12] https://github.com/CNugteren/CLBlast

[13] https://github.com/arm-software/ComputeLibrary

[14] https://github.com/ARM-software/armnn

[15] http://deepcl.hughperkins.com

[16] https://github.com/hughperkins/EasyCL

[17] https://developer.arm.com/architectures/instruction-sets/simd-isas/neon

[18] https://github.com/intel/clDNN

[19] https://github.com/ganyc717/Darknet-On-OpenCL