Journal of The Institute of Electronics and Information Engineers Vol.58, NO.1, January 2021 https://doi.org/10.5573/ieie.2021.58.1.33
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2021-58-1-4

CPU-GPU 협력 스케줄링을 통한 임베디드 시스템 내 다중 DNN 모델들의 성능 향상

(Performance Improvement of Multiple DNN Models inside Embedded Systems through CPU-GPU Collaborative Scheduling)

김 명 선*

(Myungsun Kim©)

요 약

DNN(deep neural network) 모델의 활용은 여러 응용 분야의 성능을 향상시켜왔다. 성능 향상을 위하여 DNN 모델 자체에 대한 연구는 활발하게 진행되어 오고 있지만 이를 효율적으로 수행할 수 있는 시스템에 대한 연구는 상대적으로 미비하다. 높은 인식 성공률과 다양한 응용의 요구사항들을 만족하기 위하여 로보틱스 및 자율주행 자동차와 같은 임베디드 시스템 내부에서도 여러 종류의 DNN 모델들이 활용되고 그 숫자 자체도 증가되고 있다. 따라서 GPU와 같은 DNN 가속 장치를 여러 응용들이 공유해서 사용할 때 GPU로 인한 성능 병목 현상이 나타날 수밖에 없다. 본 논문에서는 이러한 문제를 풀고자 CPU와 GPU를 동시에 DNN 모델 연산에 활용할 수 있는 스케줄링 프레임워크를 제안한다. 이 기법은 DNN 모델들의 각각의 레이어의 연산 특성에 알맞은 코어 타입을 선택하여 수행을 맡기고 서로 다른 타입의 코어 간에 데이터 전송 오버헤드를 최소화하는 방법을 사용한다. 제안된 기법을 실제 상용 보드에서 실험한 결과 적용 전 대비 최대 71.1% 향상되었다.

Abstract

The use of deep neural network (DNN) models has improved the performance of many applications. To get the better performance, research on the DNN model itself is actively progressing, but research on a system that can perform this efficiently is relatively insufficient. In order to satisfy the high recognition success rate and the requirements of various applications, various types of DNN models are used inside embedded systems such as robotics and autonomous vehicles, and the number itself is increasing. Therefore, when a DNN accelerator such as a GPU is shared and used by multiple applications, a performance bottleneck due to the GPU is bound to appear. In this paper, to solve this problem, we propose a scheduling framework that can utilize both CPU and GPU for DNN model computation. This technique uses a method of minimizing data transmission overhead between cores of different types by selecting a core type suitable for the computational characteristics of each layer of DNN models. As a result of experimenting with the proposed technique on an actual commercial board, it is up to 71.1% higher than before applied.

Keywords : Deep neural network, CPU-GPU, Collaborative scheduling, Embedded system

Ⅰ. 서 론

최근 DNN(deep neural network) 기술의 발전은 스마트폰, 로보틱스, 자율주행 자동차 등에 사용되는 다양한 응용들의 성능을 크게 향상시켜왔다[1~2]. 응용들은 점점 더 진화된 DNN 모델들을 사용하게 되고, 진화된 모델들은 증가된 연산 복잡도와 모델 다양화의 특성을 나타내게 되었다. 또한 개인 정보 보호 및 통신 네트워크 환경에 의한 지연시간 이슈 등을 피하기 위하여 서버에 의존하지 않고 시스템 내부의 컴퓨팅 자원 자체를 활용하는 임베디드 DNN 컴퓨팅은 점점 더 매력적인

* 정회원, 한성대학교 IT융합공학부 (Department of IT Convergence Engineering, Hansung University)
© Corresponding Author(E-mail : kmsjames@hansung.ac.kr)
※ 본 연구는 한성대학교 교내학술연구비 지원과제임.
Received ; September 4, 2020 Revised ; September 12, 2020
Accepted ; September 23, 2020

연구 분야로 대두되고 있다.

임베디드 DNN 컴퓨팅 요구사항을 만족하기 위하여 NVIDIA(사)의 AGX Xavier[3] 모듈과 같은 시스템이 등장하였고 이는 오토노머스 머신 분야에 널리 사용되는 시스템이다. 이는 멀티코어로 구성된 CPU와 임베디드 시스템용 GPU인 iGPU(integrated GPU)[4]로 구성되어 있다. 그림 1은 이 시스템 환경에서 DNN 연산에 핵심적으로 사용되는 콘볼루션 연산 결과를 CPU와 GPU를 각각 사용했을 때의 결과를 나타낸다. 그림의 x축은 콘볼루션의 연산을 구성하는 MAC(multiply and accumulate) 횟수를 나타낸다. 그림에서 알 수 있듯이 연산의 규모가 증가할수록 CPU는 연산 시간 증가 폭이 크고 그 절대적인 연산 시간 값 자체도 매우 크다. 반면에 GPU는 연산 규모가 증가해도 CPU 대비 상대적으로 작은 연산 시간 증가 폭을 나타내고 절대적인 연산 시간 값 자체도 상대적으로 매우 작다. 따라서 DNN 모델들의 연산은 MAC 연산에서 높은 성능을 보이는 GPU에 할당한다. 하지만 복수 개의 다양한 DNN 모델들이 동시에 수행되거나, GPU를 사용하고 DNN 모델을 사용하지 않는 응용들이 시스템 내부에 혼재할 때 GPU로 인한 성능 병목 현상을 피하기 어렵다.

본 논문에서는 GPU 병목 현상을 피할 수 있는 솔루션을 제시한다. 멀티코어 기반 CPU와 GPU가 내장되어 있는 임베디드 시스템의 구조적 특성에 맞는 DNN 모델들의 스케줄링 프레임워크를 제안한다. 제안된 프레임워크는 각 DNN 모델들을 구성하는 레이어들의 특성에 따라서 알맞은 프로세싱 코어 즉, CPU 혹은 GPU에 레이어 연산을 할당한다. 또한 서로 다른 ISA(instruction set architecture)를 갖는 코어 간에 심리스(seamless)하게 연산 결과를 입출력하기 위한 방법과 데이터 전송 기법을 제시한다. 제안된 솔루션을 적용하고 다양한 실험 조건에서 검증한 결과 최대 71.1% 성능향상이 있음을 확인하였다.

Ⅱ. 연구 배경 및 문제 정의

본 장에서는 연구에서의 대상 시스템과 이 시스템이 수행하는 DNN 연산의 특징을 살펴본다. 이어서 본 연구를 통하여 해결하고자 하는 문제를 설명한다.

1. 대상 시스템

그림 2는 NVIDIA(사)의 Tegra Xavier SoC(system on chip)의 구조를 나타낸다[3]. 그림에서 알 수 있듯이

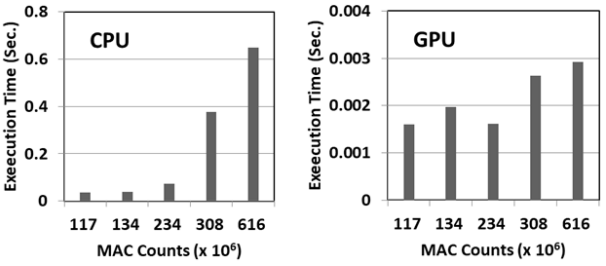

그림 1. 콘볼루션 레이어 수행시간 비교
Fig. 1. Convolution layer execution time comparison.

복수개의 ARM 코어 클러스터로 이루어진 CPU Complex와 GPU로 구성되어 있다. 각각의 ARM 클러스터는 두 개의 코어로 구성되고 클러스터 내부의 코어들은 L2 캐시를 공유하면서 각각의 L1 데이터/인스트럭션 캐시를 가지고 있다. 각 클러스터들은 L3 캐시를 공유한다. GPU는 여러 개의 SM(streaming multiprocessor)[3, 5]과 카피엔진으로 구성되며 각각의 SM은 L1 캐시를 가지고 있고 L2 캐시를 공유한다. SM은 연산을 담당하고 카피엔진은 데이터 전송을 담당한다.

그림을 통하여 알 수 있듯이 임베디드 시스템에 사용되는 GPU는 서버용 GPU와 달리 글로벌 메모리가 내장되지 않는다. 반면 CPU Complex가 사용하는 DRAM을 공유하여 사용한다. 따라서 GPU가 사용할 수 있는 메모리 공간은 CPU들에 의하여 제한될 수 있다.

각각의 ARM 코어들은 호스트로 동작하고 GPU는 호스트에서 전달받은 커널을 기반으로 디바이스로 동작한다. ARM 코어들은 보통 리눅스를 운영체제로 사용하고 여러 DNN 모델들을 응용프로그램 형태로 수행한다. 이때 연산효율성을 높이기 위하여 CUDA[5], cuDNN[6] 등의 런타임과 라이브러리를 활용한다. 따라서 CUDA 및 cuDNN을 통하여 호스트에서 커널을 전송하면 GPU는 전달받은 파라미터 및 메모리를 기반으로 SM에 분산시켜 병렬로 DNN 연산을 처리한다.

2. DNN 연산의 특성

DNN 기반의 인공신경망들은 서로 비슷한 연산 방식을 사용한다. 여러 레이어를 연속적으로 쌓은 후 마지막 레이어에서 분류작업을 수행하여 추론(inference) 결과를 출력한다. 표 1은 대표적인 CNN(convolutional neural network)기반의 DNN 모델 4종류를 비교하고 있다. 이는 각각 AlexNet[7], DenseNet201[8], ResNet152[9], Vggnet16[10]이다. 표에서 알 수 있듯이 각 모델은 콘볼루션과 풀링을 여러 레이어에 걸쳐 반복하면서 수행하

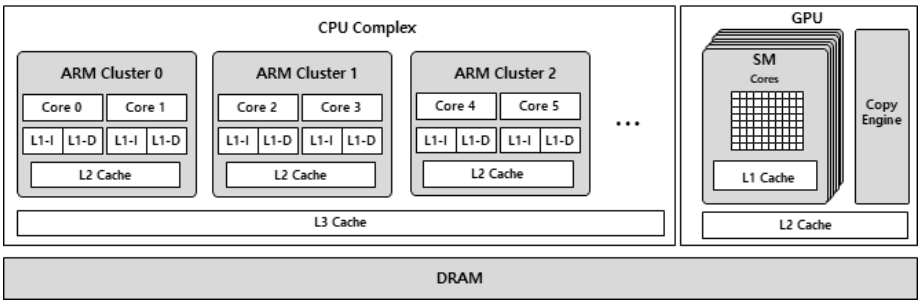

그림 2. NVIDIA(사)의 Tegra Xavier SoC
Fig. 2. NVIDIA Tegra Xavier SoC.

고 각 DNN 특성에 맞게 특별한 레이어가 추가된다. 표의 맨 우측 두 개의 항목인 DenseNet201의 dense connectivity[8], ResNet152의 skip connection[9]이 이를 나타낸다. 표에서 각 항목의 숫자는 해당 레이어 수행 횟수를 나타내고 콘볼루션 항목에는 전체 DNN 연산에서 콘볼루션 레이어들이 차지하는 컴퓨팅 량의 비율을 포함시켜 나타내었다. 컴퓨팅 량은 NVIDIA(사) AGX Xavier 보드[3]에서 각 DNN을 하나의 CPU에 고정시키고 주파수 역시 최대값으로 고정시킨 후 전체 DNN 연산 시간에서 차지하는 콘볼루션 연산 비율을 측정한 결과이다. 모든 DNN에서 90% 이상의 시간을 콘볼루션 레이어가 차지함을 알 수 있다. 콘볼루션 연산의 대부분은 MAC 동작이기 때문에 분산된 형태의 병렬 컴퓨팅을 적용 시 매우 높은 성능을 나타낸다. 따라서 GPU의 SIMD (single instruction multiple data) 처리 혹은 최근 빠르게 발전되어 가는 하드웨어 가속 기반의 NPU(neural processing unit)의 텐서[11~15] 처리를 통하여, 표 1에 나타나 있는 CPU 사용 시 90%이상을 차지했던 수행시간을 획기적으로 줄일 수 있다.

표 1. 다양한 DNN 모델들의 레이어 구성 예
Table1. Layer architectures of various DNN models.

	Conv. /Ratio	Full Con.	Max Pool	Avg. Pool	Soft Max	Dense Conn.	Skip Con.
Alex	5 / 90.8%	3	3	.	1	.	.
Dense	201 / 94.3%	.	4	1	1	98	.
Res	152 / 95.9%	.	1	1	1	.	50
VGG	13 / 98.3%	3	5	.	1	.	.

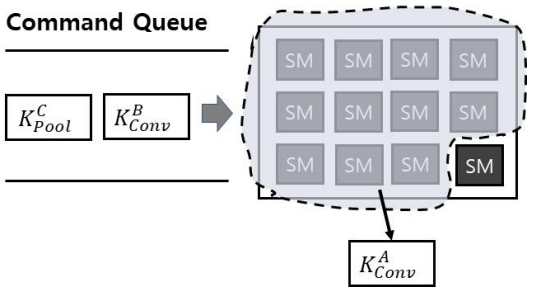

그림 3. GPU의 커맨드 큐를 통한 커널 처리
Fig. 3. GPU kernel processing through command queue.

3. 문제 정의

그림 3은 호스트에서 커널형태를 취하여 GPU에게 A, B, C로 명명된 DNN의 콘볼루션 레이어와 풀링 레이어를 커맨드 큐를 통하여 전송하고 있는 상황을 보여주고 있다. 현재 DNN A의 콘볼루션 레이어가 회색으로 칠해진 만큼의 SM들을 점유하며 수행되고 있다. 이 연산이 끝나면 DNN B의 콘볼루션이 수행되고 DNN C의 풀링 레이어가 순서에 맞춰 실행된다. DNN C의 콘볼루션 레이어가 DNN C의 풀링 레이어 수행이 끝나고 이의 결과를 입력으로 받는다면, DNN A, B의 콘볼루션이 모두 종료되고 나서 DNN C의 풀링 레이어가 완료되어야 DNN C의 콘볼루션 레이어가 수행된다. 이때 GPU보다 상대적으로 연산시간은 길지만 콘볼루션보다 MAC 연산 부담이 작은 DNN C의 풀링 레이어를 CPU에게 맡긴다면 DNN A, B의 콘볼루션 연산과 병행적으로 수행될 수 있고, DNN B의 콘볼루션 다음에 바로 DNN C의 콘볼루션 연산이 수행되어 전체 A, B, C의 연산시간을 단축시킬 수 있다.

기존 연구에서도 DNN 연산을 CPU와 GPU에 분산하여 할당하는 방법을 제시하였다. μlayer[16]는 DNN을 파티션하고 파티션된 워크로드를 스마트폰 내부의 CPU와 GPU에 효율적으로 할당한다. 또한 CPU와 GPU에 각각 적합한 양자화(quantization) 기법을 적용

하여 성능을 향상시켰다. µlayer는 단일 DNN 모델 내부의 연산량이 집중된 레이어 한 개를 파티션한 후 CPU와 GPU에 할당하는 방식을 취한다. 본 논문에서는 이와 다르게 복수 개의 DNN 모델들을 각 레이어 단위로 분리하여 CPU와 GPU에 할당하는 문제를 다룬다.

위에서 설명한대로 각각의 레이어를 서로 다른 타입의 코어들에 할당할 수 있다고 하더라도 서로 다른 타입의 코어들 간 메모리 계층구조가 다름으로 인한 데이터 복사 오버헤드는 존재한다. 따라서 그림 2에 나타낸 것처럼 DRAM을 공유하는 구조에 맞는 데이터 전송 기법을 적용하여 이를 최소화해야 한다.

Ⅲ. 다중 DNN 처리를 위한 CPU-GPU 협력 스케줄링 프레임워크

본 장에서는 CPU와 GPU가 협력해서 다중 DNN 모델들을 연산할 수 있는 스케줄링 프레임워크에 대하여 설명한다. 전체적인 시스템의 구성과 사용된 세부 기술들에 대하여 논한다.

1. 시스템 구성

제안된 스케줄링 프레임워크는 하나의 프로세스와 이를 구성하는 복수 개의 쓰레드로 이루어진다. 이러한 구조는 동일한 가상주소 공간 내에서 쓰레드 형태로 구현된 기능 블록 간 데이터 공유를 수월하게 하고 일원화된 동기화 방법을 취할 수 있는 장점을 제공한다. 그림 4는 제안된 솔루션의 전체적인 구조를 보여준다. 메인 쓰레드는 응용프로그램들이 사용하는 DNN 모델들을 나타내고 각 모델들은 별도의 쓰레드로 구현된다. 스케줄링의 핵심 기능 블록들은 레이어의 제어기(controller), 분배기(distributor), 워커(worker)로 구성된다. 이 기능 블록들을 통하여 서로 다른 ISA를 갖는 CPU와 GPU가 협력하여 DNN 모델들의 연산을 수행할 수 있다.

레이어 제어기는 메인 쓰레드로부터 전체 시스템에서 수행되는 응용프로그램이 사용하는 임의의 한 DNN 모델의 레이어 정보를 입력으로 받는다. 그 후 *Synch. Checker*를 통하여 해당 DNN 모델의 i 번째 레이어가 완료됨을 확인한 후 i+1 번째 레이어가 수행할 수 있는지 체크한다. 이를 만족하면 레이어 타입(예: 콘볼루션, 풀링 등)을 추출하고 i+1 번째 레이어의 타입에 맞는 입력 데이터를 채운다. 이때 입력 데이터는 i 번째 레이어의 출력이 된다.

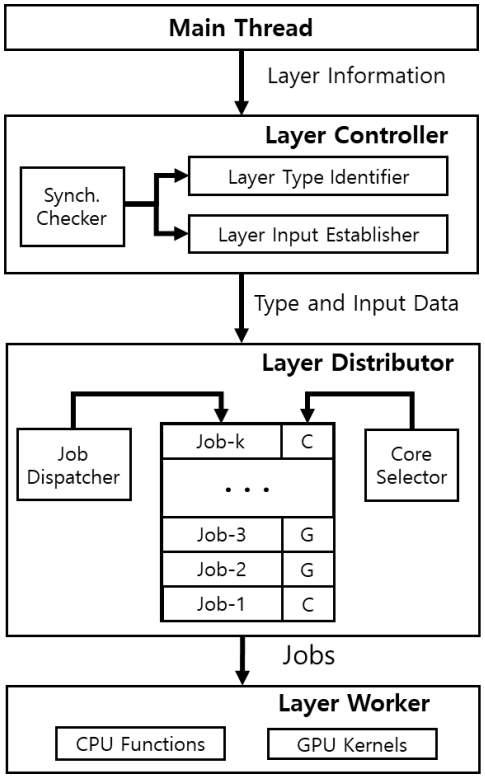

그림 4. CPU-GPU 협력 스케줄링 프레임워크
Fig. 4. CPU-GPU collaborative scheduling framework.

레이어 분배기는 먼저 전달된 레이어 타입을 확인하고 코어 선택기는 레이어 타입에 맞는 코어를 할당한다. 앞 장에서 설명한 것처럼 CNN기반 DNN 모델들의 연산의 대부분을 콘볼루션 레이어가 차지하고, MAC 연산 성격상 병렬화 처리 시 높은 성능향상을 얻을 수 있기 때문에 콘볼루션 타입을 갖는 레이어는 GPU에 할당하고 그 외 타입들은 CPU에 할당한다. 이 시점에 i 번째 레이어를 수행한 코어가 CPU이고 i+1 번째 레이어는 GPU에서 수행할 경우(혹은 그 반대의 경우도 포함), 데이터 복사에 따른 오버헤드가 발생한다[17]. 앞서 설명했듯이 본 연구에서 대상으로 하는 시스템은 그림 2에 나온 것처럼 메인 메모리를 공유하는 시스템이다. 따라서 제로카피 기법[17]을 사용하여 이를 극복한다. 이 기법에 대해서는 다음 절에서 더 자세히 기술한다. i+1 번째 레이어를 수행할 코어가 결정되면 해당 레이어를 잡(job)형태로 포맷팅 한다. 이는 레이어 워커가 해당 잡을 의뢰받아 수행 시 미리 정해진 포맷으로 입력받을 수 있게 한다. 대상 시스템에서는 cuDNN, cuBLAS, CUDA를 사용하므로 GPU가 코어로 선택되면 GPU에게 전달되는 커널에 맞는 라이브러리용 파라미터, 그리드 및 블록 크기가 설정된다. CPU가 선택되면 이와 반대로 CPU 상에서 수행될 함수에 맞는 포맷으로 형성된

다. 이 과정이 끝나게 되면 레이어 분배기 내부의 잡 전송기(dispatcher)를 통하여 잡큐(job queue)의 맨 끝에 등록되게 된다.

레이어 워커는 실제 코어에서 수행되는 연산 자체이며 잡큐에 등록된 잡들 중 항상 맨 앞의 잡을 실행한다. 이때 해당 잡에 기록된 코어 타입을 확인하고 해당 코어 타입에 맞는 입력 데이터를 미리 정해진 포맷에서 추출한다. CPU의 경우 해당 레이어에 매핑된 함수에 추출된 입력 데이터를 전송하고 GPU의 경우 커널 형태로 GPU의 커맨드 큐에 전송된다.

2. 서로 다른 코어 간 데이터 전송 오버헤드 최소화

표 1을 통해서 알 수 있듯이 DNN 모델들은 대규모 MAC 연산을 수행하는 콘볼루션 레이어와 상대적으로 이보다 연산 규모가 작은 풀링 및 기타 레이어들로 이루어진다. 또한 콘볼루션 레이어가 연속해서 결합된 형태는 찾아보기 힘들고 대부분 콘볼루션 레이어가 끝나면 그 출력을 입력으로 받아서 다른 레이어들이 동작한다. 따라서 GPU에 할당된 콘볼루션 레이어의 출력을 CPU에 할당된 풀링 레이어가 입력으로 받고 CPU의 레이어 연산 출력은 다시 GPU가 콘볼루션 입력으로 받게 된다. 이러한 반복이 계속되면 전통적인 CUDA 프로그래밍 모델에서 호스트(CPU)와 디바이스(GPU) 사이의 데이터 복사가 큰 오버헤드로 나타나게 된다.

그림 2에서 알 수 있듯이 본 연구에서 대상으로 하는 시스템에서는 DRAM을 CPU와 GPU가 공유하므로 호스트와 디바이스 간 CUDA에서 제공하는 제로카피[17]를 활용하였다. 이 기법을 사용하면 호스트의 데이터를 디바이스 쪽 메모리에 복사하지 않고 단지 호스트 데이터의 메모리상 위치만을 공유하면서 복사에 따른 오버헤드를 줄일 수 있다. 특히 메인 메모리(DRAM)를 CPU와 GPU가 공유하는 임베디드 시스템에서는 큰 효과를 기대할 수 있다.

만일 GPU에 전용 메모리인 글로벌 버퍼가 존재할 때, 복사 횟수가 적고 한꺼번에 많은 데이터를 호스트용 메모리에서 글로벌 버퍼로 복사한 후 연산을 수행하는 형태라면 제로카피 기법은 효과가 없다. 하지만 글로벌 버퍼가 없고 DNN 모델처럼 여러 레이어가 존재하면서 GPU와 CPU가 콘볼루션 및 그 외 레이어들의 연산들을 분리해서 담당한다면 많은 횟수의 호스트/디바이스 간 데이터 복사가 필요하고 제로카피는 이를 효과적으로 제거할 수 있다.

Ⅳ. 실 험

본 장에서는 제안된 스케줄링 프레임워크의 효용성을 증명하기 위해 수행한 실험들의 결과를 제시하고 이의 분석 내용을 기술한다.

1. 실험 환경

앞서 Ⅱ장에서 설명한 대상 시스템의 특성을 갖춘 실제 상용 제품인 NVIDIA(사)의 Jetson AGX Xavier 보드[3]를 실험에 사용하였다. 표 2는 구체적인 보드의 하드웨어 사양을, 표 3은 운영체제 및 라이브러리로 사용된 시스템 소프트웨어와 워크로드를 설명하고 있다. 본 연구에서는 4개의 DNN 모델을 사용하여 이들이 제안된 스케줄링 프레임워크를 적용 시 성능향상이 있음을 보인다. 사용된 모델들은 각각 AlexNet[7] DenseNet201[8], ResNet152[9], Vggnet16[10]이다.

실제 환경에서처럼 여러 응용프로그램들이 GPU를 공유하는 상황을 만들기 위하여 두 가지 워크로드를 사용하였고 이들은 각각 Rodinia[18] 벤치마크 프로그램과 인위적 합성 응용(synthetic workload)이다. 인위적 합성 응용은 CUDA를 사용하여 단순한 덧셈연산을 무한히 수행하며, 사용되는 GPU 내부 블록과 그리드 크기를 조절하면서 0%, 40%, 60%만큼 GPU 사용율을 변화시킬 수 있게 하였다.

표 2. Jetson AGX Xavier 하드웨어 구성
Table2. Jetson AGX Xavier hardware specifications.

GPU	512-Core Volta GPU/64 Tensor Cores 11 TFLOPS (FP16), 22 TOPS (INT8)
CPU	8-Core ARM v8.2 64-Bit CPU, 8MB L2 + 4MB L3
Memory	32GB 256-bit LPDDR4x, 137GB/s, 32GB eMMC 5.1 (Storage)

표 3. 시스템 소프트웨어 및 워크로드
Table3. System software and workloads.

System SW	Linux kernel version 4.9.140 CUDA 10.0, cuDNN v7
Workload	Rodinia: streamcluster, b+tree, gaussian, myocyte, kmeans, lavaMD, nn, nw, pathfinder
	Synthetic workload
	AlexNet, DenseNet201, ResNet152, Vggnet16

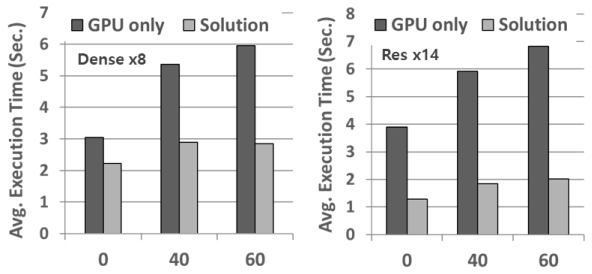

그림 5. GPU 사용율에 따른 복수 개의 동일 DNN 모델들의 수행 시간 결과

Fig. 5. Execution time results of multiple identical DNN models according to the GPU utilization.

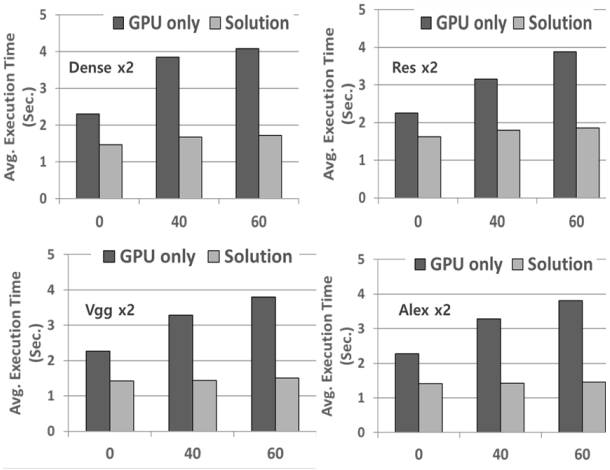

그림 6. GPU 사용율에 따른 복수 개의 서로 다른 DNN 모델들의 수행 시간 결과

Fig. 6. Execution time results of multiple different DNN models according to the GPU utilization.

2. 실험 결과 및 분석

수행된 실험들의 결과를 나타낸 그림에서 각각의 DNN 모델들은 *Dens, Res, Vgg, Alex* 이렇게 표현하고 *GPU only*와 *Solution*은 각각 CPU를 사용하지 않고 GPU만 사용하는 경우와 제안된 프레임워크를 적용했을 때를 나타낸다. 실험 결과에서 GPU를 사용하지 않고 CPU만으로 모든 연산을 수행하는 경우는 그 성능이 GPU를 사용하는 경우보다 현저히 낮아서 비교 대상에서 제외하였다.

가. 동일 DNN 모델을 사용한 다중 DNN 환경

그림 5는 인위적 합성응용을 활용하여 GPU 사용율을 0%, 40%, 60%로 설정한 후 DensNet201 8개, ResNet152 14개를 동시에 수행할 때의 성능을 나타낸다. 그림에서의 x축은 DNN 모델들이 수행될 때 인위적 합성 응용을 사용하여 설정한 GPU 사용율을 나타내고,

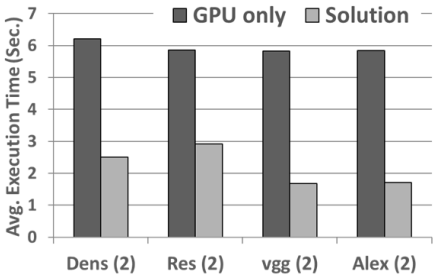

그림 7. 복수 개의 서로 다른 DNN 모델을 Rodinia와 함께 수행한 결과

Fig. 7. Execution time results of running multiple different DNN models with Rodinia.

y축은 복수 개의 DNN 모델들이 모두 종료할 때까지 걸린 수행 시간을 나타낸다. 실제 DNN 모델들이 연산을 시작하면 설정된 x축의 사용율보다 더 높은 값으로 변하게 된다.

먼저 GPU 사용율이 0%에서 60%로 증가될 때 수행시간이 늘어나는 정도를 살펴보았다. 제안된 프레임워크를 적용 시 DensNet201은 1.29배, ResNet152의 경우는 1.55배 증가된 반면, GPU만을 사용하는 경우 각각 1.95배, 1.75배로 그 증가된 폭이 상대적으로 컸다.

다음으로 수행 시간 자체를 비교하였다. 그림 5에서 알 수 있듯이 GPU 사용율 60% 설정 환경에서 절대적인 연산 시간을 비교할 때 DensNet201, ResNet152의 경우 제안된 솔루션을 적용하면 GPU만을 사용할 때보다 각각 52.1%, 70.5% 더 짧은 수행시간을 기록하였다.

나. 서로 다른 DNN 모델을 사용한 다중 DNN 환경

다음으로 4종류의 DNN 모델들을 두 개씩 동시에 수행할 때의 효과를 실험하였다. 그림 6은 그 결과를 나타내며 그림에서의 x, y축은 앞 절에서 사용한 것과 동일하다. 우선 절대적인 수행시간을 비교해보면 GPU 사용율 60% 설정 환경에서, 제안된 기법을 적용할 때 각각 DenseNet201, ResNet152, Vggnet16, AlexNet 순으로 GPU만을 사용할 때보다 57.8%, 52.3%, 60.5%, 61.7% 감소하였다.

GPU 사용율이 0%에서 60%로 증가될 때 수행 시간 변화량을 측정하였다. GPU만을 사용할 경우 최대 약 77%(DenseNet201의 경우) 늘어났지만 제안된 기법을 적용 시 최대 17%만 증가하였다. 앞서 보인 동일 DNN 모델을 사용한 다중 DNN 환경과 마찬가지로 GPU 사용율이 높은 환경에서의 성능 차이가 더 컸다. 이는 제안된 기법이 GPU에게만 DNN 모델의 연산을 의지하지 않고 CPU를 동시에 사용할 수 있기 때문이라고 할 수 있다.

다. 벤치마크 프로그램과 다중 DNN 환경

다음으로 인위적 합성 응용보다 조금 더 실제 환경에 가까운 실험을 수행하기 위하여 Rodinia를 4종류의 DNN 모델들과 동시에 수행시킨 후 DNN 모델들이 겪는 성능 저하를 분석하였다. 그림 7은 이의 결과를 나타내며 x축은 각각의 DNN 모델들과 그 개수를, y축은 각 모델의 총 수행 시간 평균을 나타낸다. Rodinia 벤치마크에서 사용된 프로그램들은 표 3에 나타내었으며 이들은 일정량의 GPU 연산과 CPU 연산을 사용한다. 본 실험에서는 이들을 무한 루프를 형성하여 각각 서로 다른 프로세스로 수행되게 하였다. 그림 7에서 알 수 있듯이 GPU만을 사용할 때보다 평균 수행시간이 50.1% ~71.1% 짧음을 알 수 있다.

V. 결 론

스마트폰, 로보틱스, 자율주행 자동차 등으로 대표되는 임베디드 시스템에서 더욱 고도화된 서비스를 제공하기 위하여 여러 DNN 모델들을 활용하고 사용되는 종류 또한 다양화되고 있다. 제한된 시스템 자원 환경하에서 DNN 연산을 독점해서 수행하는 GPU와 같은 DNN 가속 장치는 성능 병목 현상의 원인이 되고 있다. 본 연구에서는 콘볼루션 레이어처럼 규모가 큰 연산은 GPU에 할당하여 최대한 병렬가속하게 하고 나머지 레이어들은 CPU에 분산시켜 성능 병목 현상을 해결하는 솔루션을 제안하였다. 다양한 워크로드를 대표적인 DNN 모델들과 동시에 수행시키면서 실험한 결과 솔루션 적용 후 GPU만을 사용할 때보다 최대 71.1% 성능 향상을 나타내었다. 향 후 DNN 모델들의 CPU 및 GPU에서의 레이어별 수행시간 모델을 개발하고 이를 적용하여 더 향상된 레이어 기반 CPU-GPU 협력 스케줄링 알고리즘으로 연구를 확대할 계획이다.

REFERENCES

[1] J. Dyrstad and J. Mathiassen, “Grasping virtual fish: A step towards robotic deep learning from demonstration in virtual reality,” in Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO), Macau, China, 2017.

[2] P. Fekrl, M. Zadeh, and J. Dargahi, “On the Safety of Autonomous Driving: A Dynamic Deep Object Detection Approach”, SAE Technical Paper 2019-01-1044, 2019

[3] https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit

[4] R. Kaleem, R. Barik, T. Shpeisman, B. T. Lewis, C. Hu, and K. Pingali, “Adaptive Heterogeneous Scheduling for Integrated GPUs”, In the 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 151 −162. ACM, 2014.

[5] https://developer.nvidia.com/cuda-toolkit

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

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proceedings of the 26th Conference on Neural Information Processing Systems, Lake Tahoe, pp. 1097 - 1105, 2012.

[8] X. Yu, N. Zeng, S. Liu and Y. Zhang, “Utilization of DenseNet201 for diagnosis of breast abnormality”, in Machine Vision and Applications. vol. 30, Oct. 2019.

[9] L. Nguyen, D. Lin, Z. Lin and J. Cao, “Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation”, in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 2018.

[10] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proceedings of the International Conference on Learning Representations, San Diego, CA, 2015.

[11] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, “DaDianNao: A machine-learning supercomputer,” in Proceeding of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, pp. 609 - 622, 2014.

[12] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in Proceeding of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, Salt Lake, Utah, pp. 269 - 284, 2014.

[13] V. Sze, Y. Chen, T. Yang, and J. S. Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” in Proceedings of the IEEE, Vol. 105, no. 12, pp. 2295 - 2329, Jan. 2017.

[14] S. Higginbotham, “Google takes unconventional route with homegrown machine learning chips,” Next Platform, Technical Report, May 19, 2016.

[15] Y. Chen, T. Krishna, J. S. Emer, V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” IEEE Journal of Solid-State Circuits, Vol. 52, no. 1, pp. 127-138, Nov. 2017.
[16] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, “μLayer: Low Latency On-Device Inference Using Cooperative Single-Layer Acceleration and Processor-Friendly Quantization,” In Proceedings of the 14th EuroSys Conference, New York, NY, USA, Article 45, 1 - 15, 2019.
[17] https://www.fastcompression.com/blog/jetson-zero-copy.htm
[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In IEEE International Symposium on Workload Characterization (IISWC), pages 44 - 54. IEEE, 2009.

저 자 소 개

김 명 선(정회원)
2000년~2002년 LG전자 액세스네트워크 연구소 주임연구원
2002년~2011년 삼성전자 DMC 연구소 책임연구원
2016년 서울대학교 전기컴퓨터공학부 박사 졸업.
2016년~2019년 삼성전자 SR연구소 수석연구원
2019년~현재 한성대학교 IT융합공학부 조교수

<주관심분야 : NPU(인공지능 가속기), Linux kernel, HW/SW Co-desgin 등>