이종 프로세싱 유닛 상에서의 분기 병렬 실행을 통한 뉴럴넷 계산 성능 향상

(Performance Improvement of Neural-net Computation using Branch-Parallel Execution on Heterogeneous Processing Units)

유 미 선*, 권 용 인, 이 제 민, 박 제 만, 김 태 호
한국전자통신연구원
(Misun Yu, Yongin Kwon, Jemin Lee, Jeman Park, Taeho Kim)
(Electronics and Telecommunications Research Institute)

Abstract : As deep learning is used as a core technology in various fields, the types of neural nets are diversified and the architecture is also becoming more complex. Recently, neural nets include various types of operations that can be performed in parallel. Deep learning compilers are being used to transform these complex neural nets into executable code for a target hardware platform. Deep learning compilers proposed so far generate codes that sequentially execute the operations of a neural net. However, when the target platform contains multiple processing units that can be executed in parallel, it is possible to generate more high-performance code if the deep learning compilers can effectively utilize the concurrency of multiple processing uints. In this paper, we explain how to generate code that executes in parallel by distributing the parallel branches included in the neural net to CPU and NPU, and present the experimental results for SqueezeNet. According to the experimental results using the proposed branch-parallel technique integrated into an open-source deep-learning compiler named NEST-C, the code generated using the branch-parallel execution technique showed 11.12% higher performance than the code that executes operations sequentially.

Keywords : deep learning compiler, heterogeneous processing, parallel processing

I. 서 론

딥러닝은 이미지 분류, 객체 인식, 음성 인식과 같은 다양한 분야에서 핵심 기술로 활용되고 있다. 이에 따라 CNN (Convolutional Neural Network) [1], RNN (Recurrent Neural Network) [2], LSTM (Long Short-term Memory) [3] 등과 같은 다양한 뉴럴넷들이 제안되고 있다. 이러한 뉴럴넷들은 딥러닝 연산에 필요한 연산들과 이들 간의 데이터 흐름을 포함하는 그래프로 표현 가능하다.

*Corresponding Author (msyu@etri.re.kr)
유미선, 권용인, 이제민, 박제만, 김태호: 한국전자통신연구원

※ 이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2018-0-00769,인공지능 시스템을 위한 뉴로모픽 컴퓨팅 SW 플랫폼 기술 개발)

이러한 다양한 뉴럴넷들은 XLA [4], TVM [5], Glow [6] 등과 같은 딥러닝 컴파일러를 이용하여 타겟 임베디드 플랫폼에서 동작 가능한 실행코드로 변환 가능하다. 그러나 기존의 딥러닝 컴파일러들은 뉴럴넷이 포함하고 있는 연산들을 순차적으로 실행하는 코드만을 생성가능하다. 이러한 코드는 뉴럴넷 내의 연산 실행 순서가 확실히 정해져 있고, 타겟 플랫폼도 단일 프로세싱 유닛 (PU: Processing Unit) 만을 포함하고 있는 경우에는 적합할 수 있다. 그러나 최근 뉴럴넷들은 병렬적으로 실행이 가능한 연산 그룹인 병렬 분기들을 다수 포함하고 있으며, 타겟 임베디드 시스템 (예: Exynos 9820, Kirin 970) 또한 딥러닝 연산을 효율적으로 실행하기 위해 CPU (Centralized Processing Unit) 뿐 아니라 GPU (Graphic Processing Unit)나 NPU (Neural Processing Unit) 같은 딥러닝 가속기를 추가적으로

포함하고 있다. 그러므로 딥러닝 컴파일러가 리소스 효율적으로 동작하는 고성능 코드를 생성하기 위해서는 여러 PU를 활용하여 뉴럴넷의 병렬 분기를 동시에 실행하는 코드를 생성 가능해야 한다.

본 논문에서는 오픈소스 딥러닝 컴파일 프레임워크를 사용하여 (1) 병렬로 실행 가능한 뉴럴넷의 분기를 찾아내는 방법을 제안하고, (2) 제안한 방법을 사용했을 때의 추론 속도 향상률을 실험을 통해 보여준다.

Ⅱ. 연구 동기

ResNet, GoogleNet, Inception, SqueezeNet 등 널리 사용되는 최근 뉴럴넷들은 서로 데이터 의존성이 없는 연산 집합들을 포함하고 있다. 이들 연산 집합은 그래프 내에서 데이터를 공유하지 않는 서로 다른 분기의 형태로 존재한다.

그림 1은 SqueezeNet 내에 존재하는 병렬 분기의 예를 보여준다.

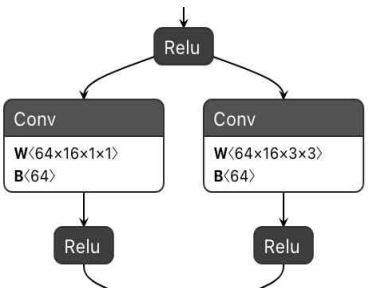

그림 1. SqueezeNet의 병렬 분기
Fig. 1. Parallel branches of SqueezeNet

그림 1에서 첫 번째 Relu 연산 이후에 이어지는 서로 다른 두 개의 분기는 서로간에 데이터 의존성이 존재하지 않아 병렬적으로 실행가능한 병렬 분기이다. 이러한 병렬 분기들은 순차적으로 실행하는 것보다 병렬적으로 실행하면 더 나은 성능을 얻을 수 있다. 특히, 타겟 플랫폼이 여러 개의 PU를 포함하고 있는 경우 각각의 분기를 서로 다른 PU에서 동시에 실행한다면 전체 뉴럴넷의 실행 속도를 향상시키는 것이 가능하다.

다음 장에서는 뉴럴넷으로부터 병렬 분기를 추출하는 방법 및 추출된 병렬 분기를 PU에 할당하는 방법에 대해 설명한다.

Ⅱ. 접근 방법

본 연구에서는 딥러닝 컴파일러가 뉴럴넷으로부터 생성한 계산 그래프 G를 입력으로 받아 병렬 분기를 추출한다. 계산 그래프는 노드 V가 연산이며 에지 E는 연산 간의 데이터 흐름을 나타낸다. 예를 들어, 그림 1에서 보이는 사각형이 노드이며, 노드 간을 연결하고 있는 화살표는 데이터 전송 방향을 나타내는 에지이다.

1. 병렬 분기 추출

입력 그래프 G(V,E)를 여러개의 분기 리스트로 분리한 후, 데이터 의존성이 없는 병렬 분기들을 추출한다. 분기와 병렬 분기의 정의는 아래와 같다.

- 분기: 순차적으로 연결된 (sequential) 노드만을 포함하고 있으며, 서로 다른 분기들 간에는 연결되는 경로가 없다 (disjoint).
- 병렬 분기 그룹: 동일한 입력 노드와 출력 노드를 가진 분기들의 집합

본 연구에서는 위 정의를 이용하여 병렬 분기 그룹들을 찾아낸다.

2. 병렬 실행

병렬 실행은 타겟 플랫폼 내에 존재하는 프로세싱 유닛들 상에서 병렬 분기 내의 분기들이 동시에 실행되는 것을 의미한다. 어떤 분기가 어떤 PU에서 실행될지는 프로파일링을 통해 결정된다. 만일 한 PU에서 병렬 분기 그룹 내의 분기들을 순차적으로 실행하는 것이 더 빠른 경우에는 병렬로 실행하지 않는다.

한 그룹 내의 병렬 분기의 수가 타겟의 PU 수보다 많은 경우에는 다음과 같은 절차에 따라 분기가 실행 될 PU를 결정한다.

1) 분기들을 PU들 상에서 실행하여 각각의 실행 시간을 구한다.
2) <PU 개수 -1> 개의 분기는 하나의 분기와 단일 PU를 매칭하고, 나머지 분기 들은 나머지 PU에 매칭할 수 있는 모든 <PU-분기> 조합을 구한다.
3) 다양한 조합에 대한 실행 시간을 계산한 후, 병렬 분기 그룹의 실행 시간을 가장 적게 소모하는 최적의 <PU-분기> 매칭을 결정한다.

Ⅲ. 실험 및 성능 분석

1. 실험 환경

1.1 병렬 분기 추출기 및 코드 생성기 구현

병렬 분기 추출기는 오픈소스 딥러닝 컴파일러인 NEST-C[1]의 그래프 파티셔너 [7]에 추가하여 구현하였다. 분기 추출기의 결과로 나온 병렬 분기 그룹들은 NEST-C의 백엔드 코드 생성기를 이용하여 타겟 하드웨어 플랫폼에서 실행 가능한 코드로 변환된다.

병렬 분기 추출기가 추가된 NEST-C는 입력 모델로부터 CPU와 EVTA상에서 분기들을 병렬 수행하는 실행 코드를 생성한다.

성능 비교를 위해 모든 분기를 순차적으로 수행하는 실행코드도 함께 생성할 수 있도록 구현하였다.

1.2 타겟 하드웨어 플랫폼

타겟 하드웨어 플랫폼은 ARM Cortex-A53 CPU와 EVTA[2] NPU가 동작하는 FPGA가 장착된 Xilinx ZCU102 보드를 사용하였다. 병렬 분기들은 ZCU102 보드의 CPU와 NPU 상에서 병렬로 실행된다.

1.3 벤치마크 및 데이터셋

뉴럴넷은 총 8개의 병렬 분기 그룹을 포함하고 있는 SqueezeNet [8]을 사용하여 실험을 진행하였다. SqeezeNet은 AlexNet [9] 보다 파라미터 수가 50배 적지만 동일한 정확도를 가진 모델이다. 크기가 0.5 메가바이즈 이하로 리소스 제약이 심한 임베디드 시스템에서도 딥러닝을 적용하는 것을 가능하게 해 주는 모델이다.

실험에 사용한 입력 데이터는 이미지 넷의 50,000개 이미지 중 1,000개를 랜덤하게 선택하여 사용하였다. 선택된 데이터를 입력으로 하여 분기 병렬 실행 코드와 순차 실행 코드를 실행한 후 평균 추론 시간을 구했다.

1. 실험 결과 및 분석

SqueezeNet을 대상으로 병렬 분기를 추출한 결과 총 8개의 병렬 분기 그룹이 추출되었으며, 각 그룹은 2개의 분기를 포함하고 있었다. 그리고 프로파일링 결과 8개의 병렬 분기 중 4개를 CPU와 NPU에서 병렬 실행할 때 최소의 추론 시간을 소모하는 것으로 계산이 되었다. 이러한 결과를 바탕으로 4개의 병렬 분기 그룹 내의 분기들이 병렬 수행되는 실행 코드가 생성되었다.

표 1은 SqueezeNet을 모든 분기들을 순차적으로 실행한 코드와, 4개의 병렬 분기 그룹내의 분기들을 병렬로 실행하는 코드의 추론 시간을 보여준다.

표 1. 분기 순차 실행과 병렬 실행 시 전체 추론 시간
Table 1. Total inference time for branch sequential and parallel execution.

	추론 시간 (millisec)		# 병렬 실행 분기 그룹
	순차 실행	병렬 실행	
SqueezeNet	202.72	180.18	4

표 1에서 볼 수 있듯이 SqueezeNet 내에 존재하는 병렬 분기를 타겟 플랫폼 내에 존재하는 여러 PU들에 나누어 실행하는 것만으로도 약 11.12%의 성능 향상 효과를 얻을 수 있었다.

분기 병렬 실행 코드는 순차 실행 코드와 동일한 연산에 동일한 PU를 사용하기 때문에 인식 정확도의 하락을 발생시키지 않는다. 즉, 타겟 시스템 내의 가용한 리소스를 좀 더 효율적으로 사용함으로써 인식 정확도의 하락 없이 추론 시간을 감소시켰다.

Ⅳ. 결 론

본 연구에서는 뉴럴넷 분기 병렬 수행 기법을 제안하고, 병렬 수행 기법을 딥러닝 컴파일러에 적용하여 뉴럴넷 실행성능 향상도를 확인해 보는 연구를 수행하였다. SqueezeNet을 대상으로 병렬 수행 가능한 분기들을 타겟 플랫폼에 장착된 CPU와 NPU에 나누어 동시에 실행하는 실험을 수행해 보았으며, 실험 결과 순차 실행 보다 속도 측면에서 11.12%의 성능 향상을 보였다. 분기 병렬 수행 기법은 여러 개의 가속기가 설치된 임베디드 보드에서도 적용 가능한 기법이며, 향후 연구에서는 이러한 다중 가속기 환경에서의 성능 향상 효과를 확인해 볼 계획이다.

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi, Understanding of a

1) https://github.com/etri/nest-compiler
2) https://github.com/etri/nest-compiler/blob/main/docs/nestc/evta.md

convolutional neural network, ICET, pp. 1-6, 2017.

[2] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, Learning representations by back-propagating errors, Nature 323, pp. 533-536, 1986.

[3] Sepp Hochreiter and Jürgen Schmidhuber, Long short-term memory, Neural computation 9, no. 8, pp. 1735-1780, 1997.

[4] Chris Leary and Todd Wang, XLA: TensorFlow, Compiled, TensorFlow Dev Summit, 2017.

[5] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al., TVM: An automated end-to-end optimizing compiler for deep learning, OSDI, pp. 578-594, 2018.

[6] Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, et al., Glow: graph lowering compiler techniques for neural networks, arXiv preprint arXiv:1805.00907, 2018

[7] 다중 가속기 지원 딥러닝 컴파일러를 위한 프로파일링 기반 그래프 파티셔닝 시스템, ISET, 2020.

[8] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer, Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size, arXiv preprint arXiv:1602.07360, 2016.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 1097-1105, 2012