Proceedings of 2020 IEMEK Symposium on Embedded Technology (ISET 2020)
GRABEL HOTEL, JEJU, Korea, 23-24 July, 2020.

다중 가속기 지원 딥러닝 컴파일러를 위한 프로파일링 기반 그래프 파티셔닝 시스템

(Profiling-based Graph Partitioning System for Multi-accelerator Deep Learning Compilers)

유 미 선*, 권 용 인, 이 제 민, 김 영 주, 김 태 호
한국전자통신연구원
(Misun Yu, Jemin Lee, Yongin Kwon, Young-Joo Kim, Taeho Kim)
(Electronics and Telecommunications Research Institute)

Abstract: Various types of neural network accelerators are integrated into embedded devices and used to efficiently execute neural networks. Accordingly, research to distribute deep learning operations to various accelerators embedded in the system to enable efficient DNN execution is also becoming important. In this paper, we propose the architecture of a system to find out which operation of the neural network can improve the performance of the entire neural net the most when it is executed on which accelerator. In particular, it describes the structural operation of the profiling-based system that finds how to divide the node group at the dataflow graph level of the deep neural network compiler.

Keywords : deep learning compiler, neural processing unit, dataflow graph, profiling, partitioning

I. 서 론

최근 뉴럴 프로세싱 유닛 (NPU: Neural Processing Unit)으로 지칭되는 다양한 신경망 전용 가속기들 (예: Exynos 9820, Kirin 970 등)이 임베디드 시스템 상에서 전력 소모를 최소화하면서 추론 성능을 향상시키기 위한 해결책으로 활용 범위를 넓혀가고 있다.

XLA [1], TVM [2], Glow [3]등의 최신 딥러닝 (deep lsearning) 컴파일러는 TensorFlow나 Pytorch와 같은 딥러닝 프레임워크에서 훈련된 딥 뉴럴넷 (DNN: Deep Neural Network)을 가속기에서 효율적으로 동작하는 머신 코드로 변환해 준다. 이러한 딥러닝 컴파일러는 가속기를 내장한 임베디드 시스템 기반 DNN 응용 개발의 효율성을 크게 향상시키고 있다.

그러나 이러한 최신 컴파일러들은 입력 DNN이 단일 프로세싱 유닛 (PU: CPU 또는 NPU)에서 실행되는 것을 가정으로 코드를 생성하고 있다. 즉, DNN의 각 계층이 임베디드 시스템의 어느 PU에서 실행되어야 최종 생성된 DNN 코드의 실행 성능이 가장 높을지에 대한 고려는 하지 않고 있다. 그러나 PU가 여러개이거나 가속기를 여러 개 포함하고 있는 임베디드 시스템의 경우에는 어떤 연산을 어느 PU에서 실행하는지가 전체 DNN 성능에 큰 영향을 미칠 수 있는 중요한 문제이므로 딥러닝 컴파일러에서 반드시 고려해야 한다.

DNN의 각 계층을 어느 PU에서 실행시킬 때 전체 DNN의 성능이 높아질 수 있는지 찾는 연구 [4]가 최근에 제안되었으나, 이 연구는 각 계층이 순서대로, 그리고 독립적으로 PU에서 실행되는 것을 가정하고 있다. 즉, 연산 결합이나 타일링 등 코드 생성을 위해 딥러닝 컴파일러에서 수행하는 최적화로 인한 성능 변화는 고려하지 않고 있다.

본 논문에서는 제안하는 시스템은 딥러닝 컴파일러와 프로파일링 기법을 활용하여 하나의 DNN

*Corresponding Author (msyu@etri.re.kr)
유미선, 권용인, 이제민, 김영주, 김태호: 한국전자통신연구원(ETRI)
※이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2018-0-00769,인공지능 시스템을 위한 뉴로모픽 컴퓨팅 SW 플랫폼 기술 개발).

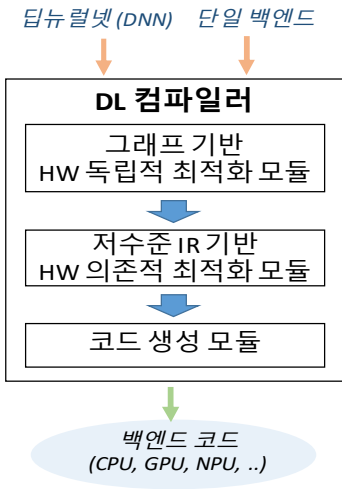

그림 1. 기존 딥러닝 컴파일러 구조

을 여러 PU 상에서 효율적으로 동작하는 코드로 변환한다.

2장에서는 배경 지식 및 연구 동기에 대해 설명한다. 3장에는 딥러닝 컴파일러와 통합된 프로파일링 기반 그래프 파티셔닝 시스템의 구성 요소와 동작에 대해 설명하고 4장에서 결론을 맺는다.

II. 배경 지식 및 연구 동기

1. 딥러닝 컴파일러

딥러닝 컴파일러는 딥러닝 프레임워크에서 훈련된 DNN을 입력으로 받아 타겟 하드웨어 상에서 효율적으로 동작하는 코드를 생성한다. 학계 및 산업계에서 사용되고 있는 잘 알려진 딥러닝 컴파일러로는 TVM, GLOW, XLA 등이 있다. 그림 1은 이들 딥러닝 컴파일러의 공통적인 구조를 보여준다.

그림 1에서 볼 수 있듯이, 딥러닝 컴파일러들은 입력으로 받는 DNN의 형식 (예: TensorFlow, ONNX, Caffe2 등)에는 차이가 있지만, 공통적으로 입력 DNN을 데이터 흐름 그래프 (DFG: Dataflow Graph)로 변환한 후 해당 그래프를 대상으로 하드웨어 아키텍처에 독립적인 최적화 (예: operator fusion, quantization, constant folding 등)를 수행한다.

하드웨어 독립적 최적화가 끝나면 그래프는 저수준 IR로 변환 된 후 타겟 하드웨어 의존적인 최적화가 수행된다. 하드웨어 의존적 최적화 단계에서는 가용한 가속기의 내부 메모리 크기, 수행 가능한 연산의 종류를 참고하여 해당 가속기에서 효율적으로 실행 가능한 코드를 생성하기 위한 추가적인 최적화 (예: tiling, memory latency hiding 등)를 수행한다.

딥러닝 컴파일러는 가능한 모든 최적화가 끝난 후에 하드웨어에서 실행 가능한 백엔드 코드 (또는 머신 코드)를 생성한다. 현재까지 제안된 딥러닝 컴파일러들은 뉴럴넷을 실행할 백엔드의 종류를 하나만 지정 가능하고, DFG의 모든 연산은 지정된 하나의 백엔드 (CPU 또는 단일 가속기)용 머신코드로 변환한다. 만일 특정 연산이 가속기 사용의 이득을 얻을 수 있는 경우, 해당 연산만 CPU에서 실행하도록 할 수 있다면 전체 DNN의 실행 성능은 모든 연산을 가속기에서 실행하는 것보다 더 높아질 것이다. 더 나아가 다양한 가속기 내장한 시스템의 경우, 각각의 가속기에서 더 효율적으로 실행 가능한 연산의 그룹 (그래프의 파티션)을 자동으로 찾아내어 그에 맞게 코드를 생성할 수 있다며 가속기들의 성능을 최대한 활용할 수 있고 좀 더 효율적인 DNN 실행이 가능할 수 있을 것이다.

본 논문에서는 여러개의 가속기를 포함하는 타겟 디바이스상에서 효율적으로 동작하는 DNN 코드를 생성하는 것을 목적으로 한다. 목적을 달성하기 위해 딥러닝 컴파일러가 DNN으로부터 생성한 DFG를 파티셔닝 하고, 각 파티션을 가장 효율적으로 실행할 수 있는 가속기를 찾아 코드를 생성해 주는 프로파일링 기반시스템을 제시한다.

2. 그래프 파티셔닝

다중 가속기를 내장한 임베디드 시스템에서 효율적으로 DNN을 실행하기 위한 연구는 대부분 한 계층을 병렬처리 가능한 단위로 나누어 동시에 실행하여 처리 속도를 높이는 방법에 초점을 맞추고 있다 [5-8].

최근 제안된 MOSIC [4]은 DNN이 효율적으로 실행될 수 있도록 이종 디바이스 상에 DNN의 각 계층을 할당하기 위한 방법이다. 이 방법은 CPU와 가속기 상에서의 연산 수행 비용과 데이터 전송 비용 등을 고려하여 가장 효율적인 슬라이스 (또는 그래프 파티션)을 찾는다. 그러나 이 방법은 DNN의 각 계층의 연산이 독립적으로 실행된다고 가정하고 있으므로 컴파일러를 사용했을 때 적용 가능한 최적화 기법들로 인한 각 가속기 상에서의 성능 변화는 고려하지 않는다.

본 논문에서는 여러 가속기에 배치될 코드를 컴

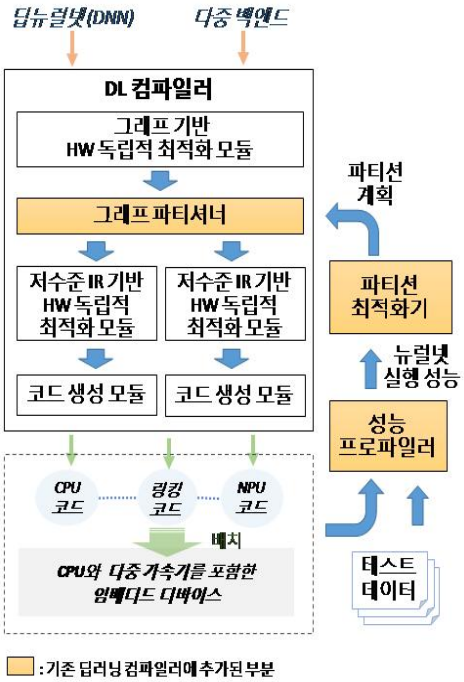

그림 2 딥러닝 컴파일러와 통합된 프로파일링 기반 그래프 파티셔닝 시스템

파일러를 이용하여 생성하고, 이 코드를 실제 실행하여 정확한 실행 성능을 측정함으로서 최적의 DFG 파티션을 찾는 방법을 제시한다.

Ⅲ. 프로파일링 기반 그래프 파티셔닝 시스템

그림 2는 딥러닝 컴파일러와 통합된 프로파일링 기반 그래프 파티셔닝 시스템의 구성 요소와 데이터 흐름을 보여준다. 노란색으로 표시된 요소들은 기존 딥러닝 컴파일러를 확장하여 추가한 부분이다.

그림 2에서 볼 수 있듯이 프로파일링 기반 그래프 파티셔닝 시스템은 딥 뉴럴넷과 가용한 백엔드의 종류를 입력으로 받아 최종적으로 타겟 임베디드 시스템에서 동작하는 머신코드를 출력으로 내보낸다. 여기서 백엔드의 종류는 타겟 임베디드 시스템이 포함하고 있는 PU의 종류를 의미한다.

그래프 파티셔너는 DFG로 변환된 DNN을 파티션 최적화기가 생성한 파티션 계획에 따라 파티셔닝하는 역할을 하며, 성능 프로파일러는 딥러닝 컴파일러가 생성한 코드를 타겟 임베디드 시스템에서 실행하는 역할을, 파티션 최적화기는 성능 프로파일러로부터 받은 최종 코드의 실행 성능을 통해 이전 그래프 파티션보다 더 나은 성능을 얻을 수 있는 새로운 그래프 파티션을 찾는 역할을 수행한다.

각 구성 요소의 자세한 설명은 아래와 같다.

1. 파티션 최적화기

파티션 최적화기는 성능 프로파일러가 현재까지 생성한 파티션 실행 정보를 바탕으로 추가적인 파티션 정보를 생성한다. 추가적인 파티션 정보를 찾는다는 것은 이전 파티션보다 전체 DNN 실행 시간을 최소화 할 수 있는 [파티션]-[PU ID] 쌍의 집합을 찾는 것이다.

새로운 파티션들의 후보를 생성하고 해당 후보들의 성능을 측정해 보기 위해서는 새롭게 생성한 파티션 후보 리스트를 딥러닝 컴파일러로 보내 코드를 생성하고 성능 프로파일러를 통해 성능 정보를 받아야한다. 여기서 딥러닝 컴파일러로 보내지는 파티션 후보들의 리스트를 파티션 계획이라고 한다. 파티션 계획은 다음과 같은 정보를 포함한다.

- 입력 DNN ID
- 파티션 계획 ID
- 파티션 개수
- 각 파티션 별
  - 파티션 ID
  - 파티션에 포함되는 노드 ID 리스트
  - 파티션을 실행할 PU ID

최초에 파티션 최적화기는 DFG의 각 노드를 독립된 파티션으로 만들고 가용한 모든 PU용 코드를 생성하도록 파티션 계획을 설정한다.

이후부터는 그래프의 파티션들을 실행 순서대로 따라가면서 이전 파티션과 병합한 파티션 계획을 생성하여 딥러닝 컴파일러로 보내고, 성능 프로파일러로부터 실행 성능을 받아보면서 최적의 파티션을 찾는다.

2. 그래프 파티셔너

그래프 파티셔너는 파티션 최적화기가 넘겨준 파티션 계획을 바탕으로 DFG를 여러개의 파티션으로 분리한 후, 각 파티션을 파티션 계획에 명시된 PU ID 리스트와 연결된 저수준 IR 기반 HW 독립적 최적화 모듈로 보낸다.

3. 성능 프로파일러

딥러닝 컴파일러가 파티션 계획에 따라 생성한 코드를 테스트 데이터를 이용하여 실행하고, 실행 성능 (실행 시간)을 측정한 결과를 파티션 최적화기로 보낸다. 성능 프로파일러가 생성한 결과는 다음과 같다.

- 입력 DNN ID
- 파티션 계획 ID
- 각 파티션 별
  - 파티션 ID
  - 파티션을 실행한 PU별
    - ✓ PU ID
    - ✓ 연산 수행 시간
    - ✓ 입출력 데이터 송수신 시간
    - ✓ 연산량 (연산의 clock cycle 수)
    - ✓ DRAM 사용량

프로파일링 기반 그래프 파티셔닝 시스템은 제한된 시간동안 또는 더 이상 통합 가능한 파티션이 없을때까지 파티션 최적화기, 그래프 파티셔너, 성능 프로파일러를 반복적으로 실행하면서 최고의 DNN 실행 성능을 내는 파티션 계획을 찾는다.

Ⅳ. 결 론

본 논문에서는 추론 성능을 최대화 하기 위해 하나의 DNN을 여러 개의 프로세싱 유닛에 나누어 배치하는 프로파일링 기반 그래프 파티셔닝 시스템을 제시하였다. 제시된 시스템은 딥러닝 컴파일러와 통합되어 동작하며, 입력으로 받은 DNN이 최적의 성능을 낼 수 있도록 DFG를 파티셔닝하여 각각의 프로세싱 유닛에 할당하는 역할을 한다. 본 논문에서 제시한 시스템은 다중 가속기가 내장된 최신칩을 위한 컴파일러 개발에 유용하게 사용될 수 있을 것으로 기대된다. 향후에는 최적의 DFG 파티션 검색 시간을 단축할 수 있는 효율인 검색 방법에 대해 연구할 계획이다.

참 고 문 헌

[1] https://www.tensorflow.org/xla

[2] Chen, Tianqi, et al. "TVM: An automated end-to-end optimizing compiler for deep learning," OSDI. 2018.

[3] Rotem, Nadav, et al. "Glow: Graph lowering compiler techniques for neural networks," arXiv preprint arXiv:1805.00907, 2018.

[4] Han, Myeonggyun, et al., "MOSAIC: Heterogeneity-, Communication-, and Constraint-Aware Model Slicing and Execution for Accurate and Efficient Inference," PACT, 2019

[5] Zou, Kaiwei, et al. "Learn-to-Scale: Parallelizing Deep Learning Inference on Chip Multiprocessor Architecture," DATE, 2019

[6] Wang, Minjie, Chien-chin Huang, and Jinyang Li. "Supporting Very Large Models using Automatic Dataflow Graph Partitioning," EuroSys, 2019

[7] Song, Linghao, et al. "Hypar: Towards hybrid parallelism for deep learning accelerator array," HPCA, 2019

[8] Song, Linghao, et al. "AccPar: Tensor Partitioning for Heterogeneous Deep Learning Accelerators," HPCA, 2020