임베디드 디바이스를 위한 딥뉴럴넷 C/C++ 코드 자동 생성 프레임워크

유미선 ○, 이제민, 권용인, 김영주, 김태호

{msyu, leejaymin, younin.kwon, kr.yjkim, taehokim}@etri.re.kr

An Automatic C/C++ code Generation Framework for Deep Neural Network Deployment on Embedded Devices

Misun Yu ○, Jemin Lee, Yongin Kwon, Young-Joo Kim, Taeho Kim

Electronics and Telecommunications Research Institute

요 약

딥 뉴럴넷은 이미지나 객체 인식, 음성인식 등 다양한 인공지능 응용에서 성공적으로 사용되고 있다. 딥 뉴럴넷 성공적 사용의 큰 이유 중 하나는 TensorFlow, Pytorch, MxNet과 같은 딥러닝 프레임워크를 이용하여 손쉽게 인공지능 응용 프로그램 작성이 가능해졌기 때문이라고 할 수 있다. 그런데 리소스가 충분하지 않아 딥러닝 프레임워크를 설치하기 어려운 디바이스상에서의 딥러닝 응용 개발은 여전히 어려운 것이 현실이다. 본 논문에서는 이러한 리소스에 제약이 큰 디바이스 상에서 실행 가능한 C/C++ 코드를 자동으로 생성해 주는 프레임워크인 NEST-CGen를 제안하고, NEST-CGen 프레임워크의 구조와 동작에 대해 자세히 설명한다. 또한 NEST-CGen이 생성한 C/C++코드와 최신 딥러닝 컴파일인 GLOW와 TVM이 생성한 코드를 성능 및 활용도 측면에서 비교한 결과도 제시한다.

1. 서 론

CNN (Convolutional Neural Network), RNN (Recurrent Neural Network)으로 대표되는 다양한 종류의 딥 뉴럴넷 (이하 뉴럴넷) 들이 이미지 구별, 객체 인식, 음성 인식 등의 인공지능 분야에서 성공적으로 활용되고 있다. 뉴럴넷은 TensorFlow, Pytorch, MxNet과 같은 딥러닝 프레임워크를 이용하여 훈련 과정을 통해 생성되며, 인공지능 응용에서 이용할 수 있도록 해당 특정 프레임워크에서 정의한 형식이나 ONNX (Open Neural Network Exchange) [1]와 같은 표준 포맷으로 저장된다. 저장된 뉴럴넷은 타겟 디바이스상에서 입출력 처리를 담당하는 인공지능 응용에 의해 로딩되어 수행되게 된다.

만약, 타겟 디바이스의 리소스 (메모리, 가속기 등)가 딥러닝 프레임워크를 설치하여 실행하기 충분한 경우에는 프레임워크에서 제공하는 모델 로딩 및 연산 라이브러리를 이용하여 간단히 뉴럴넷을 실행하는 것이 가능하다. 그러나 소형 임베디스 디바이스와 같이 리소스가 충분하지 않은 경우에는 사람이 직접 혹은 도구를 사용하여 뉴럴넷을 디바이스의 머신코드나 디바이스가 제공하는 백엔드 컴파일러 (예: GCC)의 입력 형태로 변환한 후 컴파일하여 사용해야 한다.

최근에 제안된 GLOW [2], TVM [3], XLA [4]와 같은 딥러닝 컴파일러들은 뉴럴넷을 타켓 디바이스의 메모리 아키텍처를 고려하여 해당 타켓에 최적화 된 머신 코드를 생성한다. 그러나 타켓 디바이스의 종류가 x86-64와 ARM64 등으로 제한적이어서 다양한 임베디스 시스템에서 사용하기 힘들다.

뉴럴넷을 다양한 타겟 디바이스에서 실행할 수 있도록 C코드로 변환해 주는 연구 [5][6]들도 이루어지고 있으나 Darknet 형식의 뉴럴넷 만을 입력 형식으로 지원하여 일반적으로 사용하기 어렵다.

본 논문에서는 뉴럴넷 표준 포맷 형식인 ONNX를 입력으로 받아 대표적인 크로스 컴파일러인 GCC (GNU Compiler Collection)와 LLVM (Low Level Virtual Machine)에서 컴파일 가능한 C/C++ 코드를 생성하는 프레임워크인 NEST-CGen에 대해 설명한다.

2장에서는 NEST-CGen의 구성과 동작에 대해 설명한다. 3장에서는 NEST-CGen의 성능을 최신 딥러닝 컴파일러의 성능과 비교한 결과와 NEST-CGen의 활용도에 대해 설명하고 4장에서 결론을 맺는다.

2. NEST-CGen

그림 1은 NEST-CGen의 입출력 및 구성요소를 보여준다. NEST-CGen은 딥러닝 프레임워크에서 훈련되어 저장된 ONNX (.onnx) 혹은 Caffe2 (.pb, .pbtxt) 형식의 파일을 입력으로 받아 GCC 혹은 LLVM 컴파일러로 컴파일 가능한 C/C++ 코드를 자동 생성한다. 생성된 코드는 뉴럴넷을 배치할 디바이스가 제공하는 백엔드 컴파일러로 컴파일한 후 타겟 디바이스에 업로드 하여 사용 가능하다.

2.1 코드 생성기

코드 생성기는 파일 형태로 저장되어 있는 딥 뉴럴넷을 파싱하여 DFG (DataFlow Graph)를 구성한다. DFG는 뉴럴넷이 수행해야 할 연산, 각 연산의 입출력 크기 정보 및 연산에 필요한 파라미터 값들을 포함하고 있다.

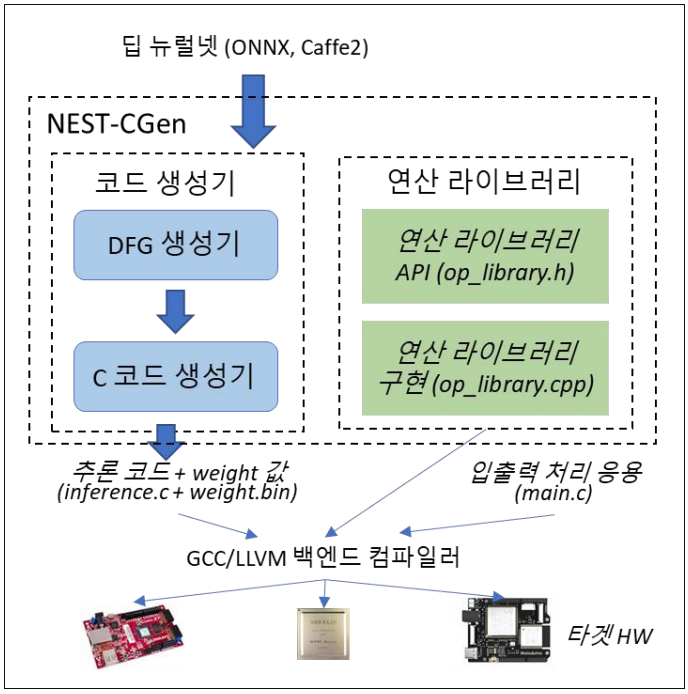

그림 1 NEST-CGen의 입출력 및 구성요소

DFG 생성이 완료되면 해당 NEST-CGen은 DFG를 바탕으로 입력 데이터를 받아 추론 결과를 리턴하는 추론 함수를 포함하는 C 코드 파일인 'inference.c'와 추론에 사용되는 훈련된 weight 값들을 저장한 바이너리 파일인 'weight.bin'을 생성한다. 'inference.c'는 1) 변수 선언부, 2) 변수 메모리 할당 및 초기화 부, 3) 연산 수행부, 4) 메모리 해지부로 구성되며 각 부분이 담당하는 역할은 아래와 같다.

1) 변수 선언부: 뉴럴넷 연산 수행에 필요한 weight 값, 각 연산의 입출력 값을 저장하는데 필요한 변수 선언
2) 변수 메모리 할당 및 초기화 부: 변수 선언부에서 선언한 변수들에 동적으로 메모리를 할당하고, weight 값 저장에 사용되는 변수들은 'weight.bin' 파일로부터 해당값을 읽어와 초기화 한다.
3) 추론 연산 수행부: 뉴럴넷에서 수행해야 할 연산들에 대응하는 '연산 라이브러리' 내의 함수들을 호출하고 최종 결과를 리턴한다.
4) 메모리 해지부: 변수들에 할당된 메모리를 해지한다.

2.2 연산 라이브러리

연산 라이브러리는 'inference.c'의 추론 연산 수행부에서 호출되는 1) 연산의 API (op_library.h)와 2) 구현 코드 (op_library.cpp)를 포함하고 있다. 그림 2는 연산 라이브러리 API와 구현코드의 예를 보여준다.

연산 라이브러리

op_library.h

```
//GCC 사용 시 정의
#define USE_GCC

//OpenBLAS 사용 시 정의
#define USE_OpenBLAS

void convolution(..)
void matmul(..)
void avgpool(..)
void maxpool(..)
void add()
..
```

op_library.cpp

```
void convolution(..) {
#ifdef USE_GCC /* GCC */
..
#else /* LLVM */
..
#endif
..
/*for matrix multiplication*/
#ifdef OpenBLAS
  /* with OpenBLAS */
..
#else
  /* without OpenBLAS */
..
```

그림 2 연산 라이브러리의 API 및 구현코드 예

1) 연산 API 코드: ONNX 연산자에 대응하는 연산을 수행하는 함수들의 API를 정의한다. API는 연산 함수들이 변수의 타입 (float, int16, int8, 등)에 의존적이지 않게 하기 위하여 C++ 템플릿 함수로 정의되었다. 그리고, 디바이스가 제공하는 컴파일러의 종류 (GCC 또는 LLVM)를 사용자가 설정하도록 하여 각 컴파일러의 문법에 차이 발생하는 부분을 자동으로 처리할 수 있도록 한다. 또한, 딥 뉴럴넷에서 가장 많은 시간을 소모하는 것으로 알려진 행렬 곱 연산의 성능을 최대한 향상시킬 수 있도록 하기 위하여 백엔드 디바이스 상에서 OpenBLAS 라이브러리의 사용 여부를 설정할 수 있도록 하고 있다.

2) 연산 구현 코드: ONNX 연산자에 대응하는 연산을 수행하는 함수들을 구현한다. 연산 API 코드에서 설정된 백엔드 컴파일러의 종류와 OpenBLAS 사용 여부에 따라 다른 구현을 제공한다.

3. 성능 비교 및 분석

3.1 실험 방법

ONNX 형식으로 저장된 Resnet-50과 VGG-19 뉴럴넷을 NEST-CGen, GLOW 딥러닝 컴파일러, TVM 딥러닝 컴파일러 각각을 이용하여 생성한 코드의 실행 성능을 비교한다.

NEST-CGen의 경우, OpenBLAS 라이브러리를 사용하는 코드와 NEST-CGen의 연산 라이브러리에 포함된 GEMM 연산을 사용하는 두 가지 버전의 코드를 모두 생성 가능하므로 두 버전의 성능을 모두 제시한다.

3.2 실험 환경

- 운영체제: macOS Catalina 10.15.3
- 메모리: 32GB
- CPU: 2.4 GHz 8코어 Intel Core i9
- 컴파일러: Xcode 11.4.1/Apple clang version 11.0.0

3.3 성능 비교 및 분석

표 1은 ResNet-50과 VGG-19을 입력으로 하여 NEST-Gen, GLOW, TVM이 생성한 코드를 x86-64 CPU상에서 실행하였을 때 걸리는 추론 시간을 보여준다. 추론 시간은 동일한 이미지를 대상으로 추론 함수를 50번 실행한 후 이미지를 추론하는데 걸리는 평균 시간을 의미한다.

표 1. NEST-CGen, Glow, TVM이 생성한 코드의 x86-64 CPU 상에서의 실행 속도

평균 추론 시간 (ms)			
Resnet-50 / VGG-19			
CGen /w OpenBLAS	CGen w/o OpenBLAS	GLOW	TVM
820.76 /7,573.03	573.13 /2,708.23	114.39 /471.03	48.28 /242.13

표 1에서 볼 수 있듯이 NEST-CGen은 OpenBLAS를 사용하는 코드의 경우에도 GLOW 보다는 약 5배, TVM 보다는 약 10배 가량 느렸다. 이러한 결과는 NEST-CGen이 뉴럴넷에 표시된 연산자에 대응하는 'NEST-CGen 연산자 라이브러리'의 함수를 호출하는 형태로 코드를 생성하였기 때문에 GLOW나 TVM 컴파일러가 수행하는 operator fusion, data-layout transformation과 같은 속도 향상을 위한 그래프 수준의 최적화 [3]가 적용되어 있지 않기 때문이다.

또한, TVM이 수행하는 최적화된 머신코드 생성을 위한 loop unrolling이나 지역성 (locality)높이기 위한 인스트럭션 스케쥴링도 적용되어 있지 않다. 그러므로 이러한 최적화 기법들을 적용하고 있는 GLOW나 TVM이 생성한 CPU용 머신 코드보다 느릴 수밖에 없다.

이러한 성능상의 단점은 사용자가 '연산자 라이브러리'의 각 연산을 타겟 디바이스의 아키텍처를 고려하여 추가적인 최적화를 함으로써 보완 가능하다.

그리고 NEST-CGen은 GLOW나 TVM과 같이 사람이 수정하기 어려운 형태의 LLVM IR형태의 저수준 코드가 아닌 고수준의 C/C++코드를 생성하기 때문에 백엔드 컴파일러가 타켓 디바이스의 리소스 (멀티코어, 가속기 등)를 얼마만큼 활용하여 C/C++ 코드를 최적화 하느냐에 따라 큰 성능차를 보일 수 있다.

추가적으로, 정확도를 측정하기 위해서 NEST-CGen with OpenBLAS, CGen without OpenBlas, GLOW, TVM의 추론 결과를 비교하였으며 비교 결과 모든 결과값이 동일하였다.

4. 결론

본 논문에서는 딥 뉴럴넷 표준 형식인 ONNX로 표현된 뉴럴넷 모델을 입력으로 받아 GCC/LLVM 컴파일러로 컴파일 가능한 C/C++ 코드를 자동으로 생성해 주는 컴파일러인 NEST-CGen의 구조 및 동작에 대해 설명했다. 비록 NEST-CGen이 생성한 C/C++ 코드가 GLOW나 TVM과 같은 최신 딥러닝 컴파일러가 지원하는 특정 디바이스 상에서의 실행 속도는 느렸지만, NEST-CGen이 생성한 C/C++ 코드는 GCC/LLVM을 백엔드 컴파일러로 사용하는 모든 디바이스에서 배치 가능하다는 장점이 있다. 이러한 장점을 바탕으로 NEST-CGen은 딥러닝 프레임워크를 설치하기 힘들고 GLOW나 TVM과 같은 딥러닝 컴파일러가 지원하지 않는 타겟 디바이스상에서 뉴럴넷을 동작 시키고자 할 때 유용하게 활용될 수 있을 것이다.

Acknowledgement

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)

참 고 문 헌

[1] ONNX, https://github.com/onnx/onnx/, 2020.

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

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

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

[5] Urbann, Oliver, et al. "A C Code Generator for Fast Inference and Simple Deployment of Convolutional Neural Networks on Resource Constrained Systems." arXiv preprint arXiv:2001.05572, 2020.

[6] Kang, Duseok, et al. "C-GOOD: C-code generation framework for optimized on-device deep learning." ICCAD. 2018.