Torch-MLIR을 활용한 LIF 기반 SNN MNIST 모델의 프론트엔드 파이프라인 설계

최항준, 조정훈*
경북대학교, *경북대학교
champ747@knu.ac.kr, *jcho@knu.ac.kr

Design of a Frontend Pipeline for LIF-Based SNN MNIST Model Using Torch-MLIR

Hangjune Choi, Jeonghun Cho*
Kyungpook National Univ., *Kyungpook National Univ.

요 약

본 논문에서는 PyTorch 기반의 스파이킹 신경망(SNN) 모델을 하드웨어 독립적인 중간표현인 MLIR(Multi-Level Intermediate Representation)으로 변환하고 최적화하는 frontend 컴파일러 프레임워크를 제안한다. 기존의 딥러닝 컴파일러는 SNN의 state와 spike 동작을 효율적으로 표현하지 못하는 한계가 있다. 이를 해결하기 위해 본 연구에서는 LIF(Leaky Integrate-and-Fire) 뉴런의 동작을 추상화한 SNN dialect를 정의하고, PyTorch의 동적 그래프에서 뉴런의 패턴을 추출하는 알고리즘을 구현하였다. 또한, 추출된 SNN 연산을 lowering하는 파이프라인을 통해 Linalg 백엔드와의 호환성을 확보하였다.

Ⅰ. 서 론

최근 인공지능(AI) 기술이 비약적으로 발전함에 따라, 방대한 연산량과 전력 소모가 시스템 성능의 주요 병목으로 대두되고 있다. 이에 대한 대안으로 인간 뇌의 신경망 동작을 모사한 스파이킹 신경망(Spiking Neural Network, SNN)이 제3세대 신경망으로 주목받고 있다[1]. SNN은 연속적인 실수값 대신 이산적인 스파이크(Spike) 신호를 통해 정보를 전달하며, 이벤트 기반(Event-driven) 처리를 통해 높은 에너지 효율성을 제공할 잠재력을 가진다.

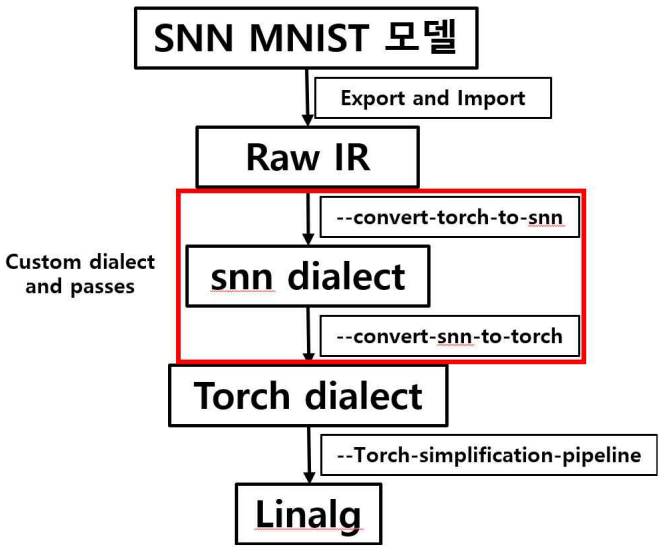
그러나 현재 SNN의 실용화를 가로막는 가장 큰 장벽은 최적화된 하드웨어 및 소프트웨어 인프라의 부재이다. 기존의 GPU나 NPU는 밀집 행렬 연산(Dense Matrix Operation)에 특화되어 있어, SNN 특유의 희소성(Sparsity)과 비동기적 특성을 효율적으로 처리하지 못한다. 또한, SNN 구동을 위한 전용 뉴로모픽 하드웨어(Neuromorphic Hardware) 개발은 아직 초기 단계에 머물러 있어, 하드웨어 아키텍처 설계와 검증에 난항을 겪고 있다. 이러한 상황에서 다양한 하드웨어 백엔드에 유연하게 대응하면서도 SNN 모델을 효율적으로 변환할 수 있는 범용적이고 최적화된 프론트엔드(Frontend) 컴파일러의 구성이 필수적으로 요구된다.

현재 SNN 연구는 주로 snntorch[2]나 SpikingJelly[3]와 같은 PyTorch 기반 프레임워크상에서 이루어지고 있다. 이러한 프레임워크는 연구 편의성은 높으나, 파이썬 인터프리터 의존성과 하드웨어 종속적인 구현 방식으로 인해 SNN 모델을 실제 임베디드 장치나 전용 가속기에 배포하는 데 한계가 있다. 기존의 딥러닝 컴파일러(TVM, XLA 등) 역시 SNN의 시간적 동작(Temporal Dynamics)을 단순한 산술 연산의 나열로만 인식하여, 뉴런 레벨의 고수준 최적화를 수행하지 못한다.

본 논문에서는 이러한 문제를 해결하고 SNN 하드웨어 개발을 위한 소프트웨어적 토대를 마련하기 위해, MLIR(Multi-Level Intermediate Representation) 기반의 SNN 프론트엔드 컴파일러 기반을 마련한다. 기존 PyTorch 모델을 수정 없이 입력받아 SNN 동작을 추상화한 전용 중간 표현(Dialect)으로 변환하고, 이를 다시 하드웨어 독립적인 선형대수 연산(Linalg)으로 최적화하여 생성한다. 이를 통해 SNN 모델의 논리적 구조를 유지하면서도 다양한 하드웨어 백엔드로 확장 가능한 범용적인 프론트엔드 환경을 구축하고자 한다.

Ⅱ. 본론

본 장에서는 SNN 모델을 효율적으로 컴파일하기 위해 설계된 전체 프레임워크의 구조를 설명한다. 제안하는 시스템은 [그림1]과 같이 크게 snn dialect 설계, Torch dialect와 snn dialect사이의 변환 pass, 그리고 표준 연산으로의 lowering 단계로 구성된다.



[그림1] snntorch frontend pipeline

본 연구에서는 제안하는 컴파일러 프레임워크의 동작 검증 및 최적화 효과를 평가하기 위해 타겟 코드를 MNIST 데이터셋으로 학습된 3층 구조(784-1000-10)의 Fully Connected SNN모델을 선정하였다. 해당 모델은 LIF(Leaky Integrate-and-Fire) 뉴런을 기반으로 하며, 각 계층 간의 시냅스 연결이 조밀(dense)하게 구성되어 있어 컴파일러의 메모리 관리 및 연산 분해 능력을 검증하기에 적합하다. 특히, CNN과 달리 모든 입력 뉴런이 출력 뉴런과 연결되므로 리프팅 단계에서 패턴 매칭의 복잡도가 높고, 로워링 단계에서 대규모 행렬 연산 변환 시 발생할 수 있는 병목 현상을 분석하기에 용이하다. 또한, MNIST는 가장 널리 사용되는 벤치마크로서, 기존 연구들과의 비교 용이성을 확보하고 프레임워크의 범용성을 입증하기 위한 기준 모델로 선정되었다.

MLIR의 기존 Dialect들은 ANN(Artificial Neural Network)에 최적화되어 있어, 시간적 동작을 포함하는 SNN의 특성을 표현하기에 부족하다. 본 연구에서는 이를 보완하기 위해 SNN 전용의 snn Dialect를 정의하였다. 핵심 연산인 snn.lif는 LIF 뉴런의 복잡한 동작을 단일 노드로 추상화하며, 입력 전류(Input Current), 이전 막전위(V), 감쇠율(Beta), 임계값(Threshold)을 인자로 받아, 최종적으로 출력 스파이크(Spike), 갱신된 막전위(V)가 출력값으로 나온다. 이러한 고수준 추상화는 그래프의 복잡도를 낮추고, 컴파일러가 해당 연산이 뉴런임을 인지하게 하여 향후 SNN 특화 최적화를 적용할 수 있는 기반을 제공한다.

--convert-torch-to-snn은 여러 산술 연산 그래프에서 'LIF뉴런'의 의미론적 구조를 찾아내어 snn.lif 연산으로 치환하는 과정이 구현되어 있다. 이 과정에서 lookThrough 함수를 구현하여 데이터값에는 변화를 주지 않는 보조 연산들을 재귀적으로 건너뛰고 원본 텐서를 찾아냄으로써, 그래프의 구조가 변형되더라도 뉴런의 입력인 막전위와 입력 전류를 식별할 수 있도록한다. 또한 snntorch 라이브러리의 특징적인 구현 방식인 Sub-Zero패턴($V_{mem} - V_{th} > 0$)을 처리하기 위해 발화여부를 결정하는 aten.gt, 임계값 비교를 위한 aten.sub, 이전 값의 적분 및 감쇠연산을 위한 aten.add, aten.mul 연산을 추적하여 전체 뉴런 구조가 일치할 때만 단일 snn.lif 노드로 치환된다.

--convert-snn-to-torch는 추상화된 snn.lif연산을 표준 Torch 연산들의 조합으로 분해한다. LowerLIFPattern클래스를 통해 단순히 이전 연산으로 되돌리는 것이 아닌, Linalg와의 호환성을 확보하는 방향으로 설계하였다. 구체적으로, 감쇠율을 상수로 변환하여, 이전 막전위와 곱하고 입력 전류를 더하는 적분 과정을 거친 후, 임계값 비교를 통해 bool타입의 스파이크 텐서를 생성한다. 이때, 출력된 스파이크 텐서를 리셋 수식($V = V_{mem} - S_{out} * V_{th}$)에 대입하기 위해 다시 float형으로 바꿔주는 작업을 진행하였다. aten.type_as연산을 삽입하여 bool 타입의 스파이크를 입력 텐서와 동일한 float타입(1.0, 0.0)으로 변환한 후 연산하도록 하였다. 결과적으로 Linalg까지 lowering과정을 확인할 수 있었다.

Ⅲ. 결론

본 논문에서는 하드웨어 독립적인 SNN 가속을 위한 MLIR 기반의 프론트엔드 컴파일러 프레임워크를 제안하고 구현하였다. 제안하는 시스템은 PyTorch로 작성된 SNN 모델을 입력받아, SNN 특화 중간 표현인 snn Dialect로 추상화하고, 이를 다시 최적화된 표준 연산으로 변환하는 Frontend 파이프라인을 제공한다.

향후 연구를 통해 snntorch로 구성된 다양한 모델 및 SpikingJelly로 구성된 다양한 모델들을 타겟으로 범용적인 snn-frontend를 구성할 예정이다.

ACKNOWLEDGMENT

이 논문은 2025년도 정부(산업통상자원부)의 재원으로 한국산업기술진흥원의 지원(RS-2024-00415938, 2024년 산업혁신인재성장지원사업) 및 2025년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.RS-2025-02216517, 재구성형 인공지능 프로세서 SW 프레임워크 기술개발)을 받아 경북대학교에서 수행된 연구 결과입니다.

참 고 문 헌

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," Neural Networks, vol. 10, no. 9, pp. 1659-1671, 1997.

[2] J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning," Proceedings of the IEEE, vol. 111, no. 9, pp. 1016-1054, Sept. 2023.

[3] W. Fang et al., "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence," Science Advances, vol. 9, no. 40, 2023.