최고 성능 그 이상: AI에 최적화된 FPGA와 GPU의 실제 성능 비교하기

앤드류 부트로스[1,2] , 에리코 누르비타디[1] , 루이 마[1] , 세르게이 그리복[1] , 지펭 자오[3] , 제임스 C. 호[3] , 본 베츠[2] , 마틴 랭해머[1]
[1]인텔 코퍼레이션 프로그래머블 솔루션 그룹
[2]토론토 대학교 및 벡터 연구소 [3]카네기 멜론 대학교
{andrew.boutros, eriko.nurvitadhi}@intel.com

요약-인공지능(AI)의 중요성과 컴퓨팅 수요가 증가함에 따라 도메인에 최적화된 하드웨어 플랫폼이 등장했습니다. 예를 들어, 엔비디아 GPU는 딥 러닝(DL) 연산 속도를 높이기 위해 매트릭스 연산에 특화된 텐서 코어를 도입하여 T4 GPU에서 최대 130int8 TOPS의 매우 높은 피크 처리량을 제공합니다. 최근 인텔은 12nm GPU에 필적하는 최대 143 int8 TOPS의 예상 피크 성능을 제공하는 인패브릭 AI 텐서 블록을 갖춘 최초의 AI 최적화 14nm FPGA인 Stratix 10 NX를 출시했습니다. 그러나 실제로 중요한 것은 최고 성능이 아니라 목표 워크로드에서 실제로 달성할 수 있는 성능입니다. 이는 주로 텐서 유닛의 활용도와 가속기와 데이터를 주고받는 시스템 수준의 오버헤드에 따라 달라집니다.

이 백서에서는 대규모 실시간 DL 추론 워크로드에서 인텔의 AI 최적화 FPGA인 Stratix 10 NX를 최신 액세스 가능한 AI 최적화 GPU인 Nvidia T4 및 V100과 비교한 첫 번째 성능 평가를 제시합니다. 유니티는 FPGA의 AI 텐서 블록을 활용하기 위해 브레인웨이브 NPU 오버레이 아키텍처의 재구현을 개선하고, 사용자가 루프에서 FPGA EDA 툴 없이 소프트웨어만으로 텐서 블록을 프로그래밍할 수 있는 툴체인 지원을 개발했습니다. 먼저 텐서 블록이 없는 스트라틱스 10 GX/MX 버전과 스트라틱스 10 NX NPU를 비교한 다음, T4 및 V100 GPU와의 상세한 코어 컴퓨팅 및 시스템 수준 성능 비교를 제시합니다. Stratix 10 NX의 향상된 NPU가 GPU보다 더 나은 텐서 블록 활용도를 달성하여 T4 및 V100에 비해 평균 24배 및 12배의 컴퓨팅 속도 향상을 가져온다는 것을 보여줍니다.
배치 6의 GPU. 지연 시간 제약이 완화되어 있어도
배치 크기가 32개인 경우에도 T4 및 V100 GPU에 비해 각각 5배, 2배의 평균 속도 향상을 달성합니다. 시스템 수준에서는 100Gbps 이더넷이 통합된 FPGA의 세분화된 유연성을 통해 짧은 시퀀스 및 긴 시퀀스 RNN의 경우 128Gbps PCIe를 통한 V100 GPU 로컬 액세스보다 각각 10배 및 2배 짧은 시스템 오버헤드 지연 시간으로 원격 액세스가 가능합니다.

***색인 용어-FPGA*, GPU, 딥 러닝, 신경망**

I. 소개

딥 러닝(DL)의 급속한 발전으로 로봇 공학[1], 자연어 처리[2], 복잡한 전략 게임[3], [4] 등 점점 더 많은 애플리케이션 영역에서 전례 없는 품질의 결과를 제공하고 있습니다. 이러한 발전은 또한 무수히 많은 최종 사용자 상용 애플리케이션의 문을 열었습니다. 마이크로소프트, 구글, 페이스북과 같은 주요 기술 기업들은 현재 다양한 DL 기반 지능형 서비스를 제공하고 있습니다[5]-[7]. 더 나은 품질의 결과를 제공하기 위해 개선된 DL 알고리즘은 계속해서 더 많은 컴퓨팅을 요구하고 있으며, 이는 특히 이러한 애플리케이션의 엄격한 지연 시간 제약과 결합하여 큰 도전 과제가 되고 있습니다. 이로 인해 AI에 최적화된 컴퓨팅을 위한 무수한 하드웨어 혁신과 다양한 솔루션이 등장했습니다. 예를 들어, 엔비디아는 GPU 마이크로아키텍처를 개선하여 DL 워크로드를 대상으로 하는 행렬 연산을 위한 특수 유닛인 텐서 코어를 긴밀하게 통합했습니다. 이러한 유닛은 높은 텐서 연산 처리량을 제공하므로 GPU의 피크가 크게 증가합니다.

초당 테라 연산(TOPS). 최근에는 최대 820 `int8` TOPS[10]의 더 높은 최고 성능을 약속하는 Groq의 텐서 스트리밍 프로세서[8]와 Graphcore의 인텔리전스 처리 장치[9]와 같은 다양한 AI ASIC이 발표되었습니다.

FPGA의 경우, 최대 디바이스 처리량을 개선하기 위한 몇 가지 제안은 Xilinx Ver- sal 아키텍처[11] 또는 인텔의 시스템 인 패키지 에코시스템[12], [13]의 AI 대상 칩렛과 같은 분리된 AI 최적화 컴퓨팅 컴플렉스에 FPGA 패브릭을 거칠게 통합했습니다. 최근에는 인텔이 새로운 AI 텐서 블록을 통합하고 최대 143개의 `int8` 및 블록 `fp16` TOPS[14]를 제공하는 최초의 AI 최적화 FPGA인 Stratix 10 NX를 출시하여 비슷한 세대의 GPU에 필적하는 최고 성능을 제공합니다. 이전 접근 방식과 달리 NX AI 텐서 블록은 표준 FPGA DSP 블록과 유사하게 FPGA 패브릭에 긴밀하게 통합되어 있습니다. 이러한 긴밀한 통합을 통해 프로그래머블 패브릭에 보다 풍부하고 유연한 연결이 가능하며, 표준 FPGA 개발 흐름을 사용하여 프로그래밍할 수 있습니다.

텐서 컴퓨팅을 AI에 최적화된 하드웨어에 통합하려는 경쟁이 치열해지면서, 잠재적인 가속 솔루션을 비교할 때 최고 TOPS 수치가 주요 지표로 사용되고 있습니다. 그러나 이는 텐서 유닛이 100% 활용될 때만 최고 성능을 달성할 수 있기 때문에 오해의 소지가 있을 수 있으며, 실제 애플리케이션에서는 일반적으로 그렇지 않습니다. 텐서 컴퓨팅 유닛의 활용도는 일반적으로 두 가지 주요 요인, 즉 주어진 워크로드를 사용 가능한 컴퓨팅 유닛에 매핑하는 것과 칩에서 데이터를 입출력할 때 발생하는 엔드투엔드 시스템 수준의 오버헤드에 의해 영향을 받습니다. 이 연구에서는 주요 AI 워크로드에 대한 코어 컴퓨팅 및 시스템 수준 성능에 대한 상세한 평가를 통해 AI에 최적화된 FPGA 및 GPU의 *실제 달성 가능한* 성능을 연구합니다. FPGA 평가를 위해 최근 발표된 스트라틱스 10 NX의 텐서 블록을 효율적으로 사용할 수 있도록 최신 상용 AI 소프트 프로세서인 마이크로소프트의 브레인웨이브(Brainwave) NPU[5]의 설계를 개선합니다. 그런 다음 스트라틱스 10 NX에 구현된 향상된 NPU를 텐서 블록이 없는 스트라틱스 10 GX/MX 디바이스의 [13]의 기준 NPU와 비교합니다. 마지막으로 NX의 향상된 NPU를 텐서 코어가 탑재된 최신 액세스 가능한 AI 최적화 Nvidia GPU인 T4 및 V100[1] 과 비교하고 대규모 실시간 DL 제품군에서 실제 성능을 연구합니다. 추론 워크로드에 적합합니다: MLP, RNN, GRU 및 LSTM.

이 백서의 기여는 다음과 같습니다:

- 스트라틱스 10 NX 텐서 블록을 활용하기 위해 NPU 오버레이에 대한 아키텍처, 명령어 세트 및 컴파일러가 개선되었습니다.
- 텐서 블록이 없는 표준 FPGA에서 향상된 NPU와 이전 기준 NPU의 성능 비교.
- 다양한 실시간 DL 워크로드에서 AI에 최적화된 FPGA 및 GPU의 실제 달성 가능한 성능을 평가합니다.
- 이더넷 연결 FPGA와 PCIe 연결 GPU 모두에서 시스템 수준 오버헤드 특성화.

1Nvidia는 최근 처리량이 더 높은 7nm A100 디바이스를 발표했지만 아직 일반에 출시되지 않아 벤치마킹할 수 없으며 공정 기술 측면에서도 비교할 수 있는 디바이스가 아닙니다.

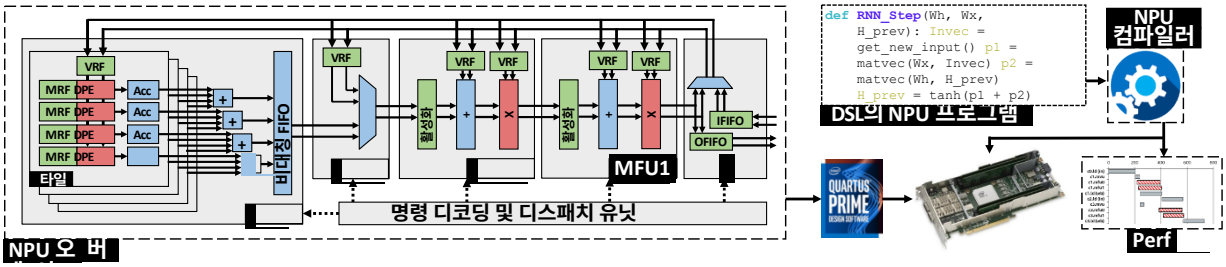

그림 1: 기본 NPU 오버레이 아키텍처의 개요 및 NPU 소프트 프로세서 프로그래밍을 위한 프런트엔드 툴 체인.

II. 배경

A. DL 워크로드: MLP 및 RNN

2017년 Google DL 데이터센터 워크로드의 90%를 차지한 다층 퍼셉트론(MLP)과 순환 신경망(RNN)을 사용한 실시간 DL 추론을 연구합니다[6]. 더 새롭고 정교한 DL 알고리즘이 등장했지만, 많은 최신 모델들이 MLP와 RNN을 서브루틴으로 사용하기 때문에 이러한 모델은 여전히 DL 워크로드의 상당 부분을 차지하고 있습니다[15], [16]. MLP[17]는 그 사이에 비선형 활성화 함수가 있는 완전히 연결된 계층을 쌓아 올린 것으로, 가장 단순한 형태의 신경망이라고 할 수 있습니다. 많은 애플리케이션에서 널리 사용되고 있으며 Facebook의 최신 추천 모델에 필수적인 구성 요소입니다[18].

반면에 RNN은 음성 샘플이나 문장과 같은 시퀀스 입력을 처리하는 모델입니다. 주로 다수의 행렬-벡터 곱셈과 *게이트*를 형성하는 벡터 요소별 연산으로 구성됩니다. 이 작업에서는 바닐라 RNN, 게이트 순환 유닛(GRU), 장단기 메모리(LSTM)의 세 가지 변형 RNN을 사용하여 각각 시간 단계당 2, 6, 8개의 벡터-행렬 곱셈을 수행합니다[19]-[21]. 다음 공식은 LSTM의 계산을 설명합니다.

$$i_t = \sigma(x W_{ti} + h U_{t-1i} + b)_i \qquad f_t = \sigma(x W_{tf} + h U_{t-1f} + b_f)$$

$$ot = \sigma(xtWo + ht\text{-}1Uo + bo) \qquad gt = 탄h(xtWg + ht\text{-}1Ug + bg)$$

$$CT = FT \circ CT\text{-}1 + IT \circ GT \qquad HT = OT \circ 탄(CT)$$

여기서 xt, it, ft, ot, gt, ct, ht는 각각 시간 단계 t*에서* 입력, 입력 게이트, 잊음 게이트, 출력 게이트, 셀 입력, 셀 상태 및 숨겨진 상태 벡터입니다. 연산자 ◦와 +는 벡터 요소의 현명한 곱셈과 덧셈을 나타내며, σ와 tanh는 시그모이드 및 쌍곡선 탄젠트 활성화를 나타냅니다. W와 U는 입력 행렬과 숨겨진 행렬이며, b항은 바이어스 벡터입니다. RNN은 다음과 같습니다.

일반적으로 음성 분석 및 자연어 처리에 사용되며, 가장 최근의 MLPerf v0.7 추론 벤치마크 제품군[22]의 일부입니다. 이 백서에서는 크기와 단계 수에 따라 RNN을 지정합니다. LSTM-1024-16 워크로드는 8개의 1024×1024 행렬과 16개의 타임 스텝을 가진 LSTM을 설명합니다. 우리가 사용하는 RNN은 DeepBench [23]와 엔비디아의 퍼시스턴트 RNN [24]에서 가져온 것입니다.

B. 베이스라인 NPU 아키텍처 및 툴체인

Brainwave NPU는 지연 시간이 짧은 배치 1 DL 추론[5]을 목표로 Microsoft에서 설계한 FPGA 기반 AI 오버레이입니다. 이 오버레이는 모든 모델 가중치를 하나 또는 여러 개의 네트워크에 연결된 FPGA에 지속적으로 유지하여 처리 지연 시간을 증가시킬 수 있는 외부 오프칩 메모리 액세스를 제거합니다. 기준 NPU는 [5] 및 [25]에 공개된 설명을 기반으로 Microsoft의 Brainwave NPU를 재구현한 것입니다. 그러나 다른 하드웨어 플랫폼과 보다 직접적인 비교를 위해 Microsoft의 커스텀 부동 소수점 형식(예: `fp11` 및 `fp8`) 대신 표준 `int8` 정밀도를 구현했습니다. 재구현의 성능은 동일한 NPU 구성을 사용할 때 Microsoft Brainwave의 성능과 거의 일치합니다. 그림 1은 5개의 주요 파이프라인 단계로 구성된 NPU 아키텍처의 개요를 보여줍니다. 매트릭스-벡터 곱셈 유닛(MVU)은 T 타일과 타일 간 가산기 감소 트리로 구성됩니다. 각 타일에는 입력 벡터를 저장하는 VRF(벡터 레지스터 파일)와 각 타일에 L *개의* 곱셈 레인이 있는 DPE(도트 곱셈 엔진)가 포함되어 있습니다.

DPE는 영구 모델 가중치를 저장하는 매트릭스 레지스터 파일(MRF) 및 로컬 누산기와 긴밀하게 결합되어 있습니다. MVU의 출력 대역폭은 MVU 블록을 종료하기 전에 비대칭 FIFO를 사용하여 D에서 L 벡터 요소로 축소됩니다. 외부 VRF(eVRF) 블록을 사용하면 매트릭스 벡터 연산이 없는 명령에 대해 MVU를 건너뛸 수 있습니다. eVRF 뒤에는 벡터 요소별 활성화, 덧셈 및 곱셈을 위한 두 개의 다기능 유닛(MFU)이 있습니다. 마지막으로 로더(LD) 블록은 입출력 FIFO를 통해 외부와 통신하고 파이프라인 결과를 아키텍처의 모든 VRF에 다시 쓸 수 있습니다. 이 아키텍처는 파이프라인의 각 단계에 하나씩 총 5개의 매크로 연산(mOP)으로 구성된 VLIW 명령어를 사용합니다. mOP는 일련의 마이크로 연산(μOP)으로 디코딩되어 여러 파이프라인 단계로 전달됩니다. 그림 1에 예시로 표시된 NPU에는 타일 4개, DPE 4개, 레인이 2개 있습니다. 이러한 구성을 4T-4D-2L이라고 하며 이 백서의 나머지 부분에서는 이 표기법을 사용하여 NPU 구성을 설명합니다.

또한 애플리케이션 개발자가 그림 1과 같이 도메인별 언어(DSL)로 NPU 프로그램을 작성할 수 있는 완전한 NPU 소프트웨어 툴체인을 구현합니다. NPU 프로그램은 NPU VLIW 명령어로 컴파일되며, 이는 C++ 사이클 정확도 성능 시뮬레이터에서 시뮬레이션하거나 배포된 NPU 인스턴스를 프로그래밍하는 데 사용할 수 있습니다. 이 접근 방식을 사용하면 애플리케이션 개발자는 FPGA 설계 전문 지식이 없거나 FPGA CAD 툴의 긴 컴파일 시간으로 인해 어려움을 겪지 않고도 다양한 NPU 프로그램을 빠르게 실험할 수 있습니다. 26]에 기반한 약간의 수정을 통해, 우리의 NPU 아키텍처와 툴체인은 컴퓨터 비전 애플리케이션을 위한 또 다른 중요한 DL 워크로드 클래스인 컨볼루션 신경망을 지원할 수 있습니다. 그러나 이는 향후 작업을 위해 남겨두고 이 섹션에서 논의한 워크로드에 주로 초점을 맞춥니다. NPU 오버레이 및 툴체인에 대한 자세한 내용은 [5] 및 [13]을 참조하시기 바랍니다.

III. 스트라틱스 10 NX: AI에 최적화된 FPGA

A. DL용 FPGA 아키텍처 동향

DL의 계산 수요 증가와 보편화로 인해 더 많은 계산 유닛, 특히 추론을 위한 저정밀 인티저 곱셈기(MAC)를 추가해야 합니다. 27]-[29]의 저자는 소프트 저정밀 곱셈기의 밀도를 높일 수 있는 몇 가지 논리 블록 수정을 제안합니다. 다른 연구에서는 비용이 많이 드는 라우팅 포트 추가를 피하기 위해 동일한 블록 인터페이스를 유지하면서 저정밀 MAC 밀도를 높이기 위해 DSP 블록에서 승수의 분해능을 향상시키는 데 중점을 둡니다. 30]에서는 인텔과 유사한 DSP 블록을 수정하여 다이 크기를 거의 늘리지 않고 블록당 더 많은 `int9` 및 `int4` 연산을 지원하도록 했습니다. 같은 맥락에서 차세대 Intel Agilex 제품군 [31]은 블록당 4개의 `int9` 곱셈이 가능한 DSP 모드를 추가하고, PIR-DSP 블록 [32]은 낮은 정밀도와 상호 연결 및 데이터 재사용 최적화를 통합하여 Xilinx와 유사한 DSP 블록을 향상시킵니다. 일부 선행 연구 [33], [34]에서도 특수 DL 하드 블록을 FPGA 패브릭에 통합하는 아이디어를 탐색했지만, 실제 DL 가속 아키텍처에 대한 실험 결과는 보여주지 못했습니다. 최근 발표된 새로운 AI 텐서 블록이 포함된 Stratix 10 NX FPGA는 레거시 DSP 블록을 제거했습니다.

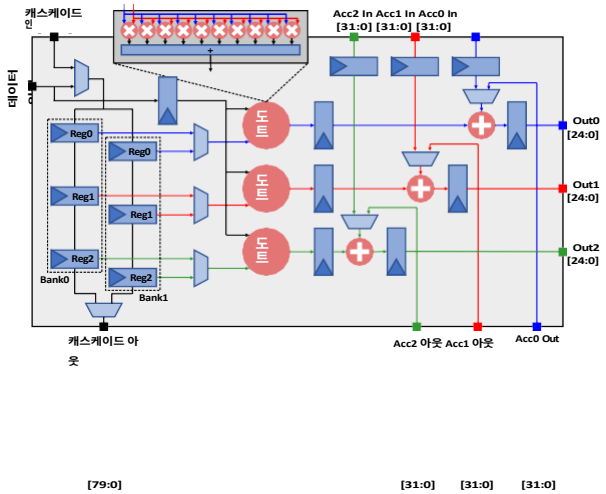

그림 2: Stratix 10 NX AI 텐서 블록의 `int8` 텐서 모드의 단순화된 블록 다이어그램.

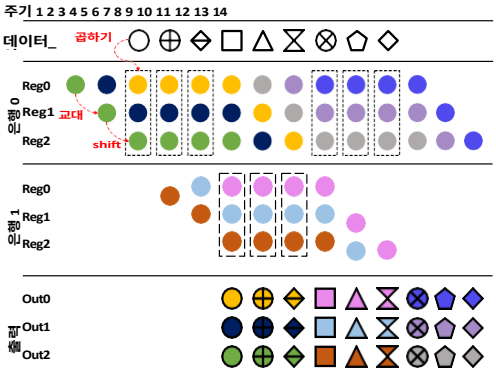

그림 3: Stratix 10 NX AI 텐서 블록의 텐서 모드 작동. 도형은 3개의 도트 프로덕트 유닛으로 전송되는 다양한 입력 데이터 벡터를 나타내고, 색상은 위 블록에서 입력 캐스케이드를 통해 핑퐁 레지스터 뱅크로 이동되는 벡터를 나타냅니다. 점선 상자는 각 사이클에서 도트 프로덕트에 공급되어 해당 모양과 색상의 출력을 생성하는 레지스터 뱅크를 나타냅니다.

효율적인 필터링 구조를 구현하는 데 사용되는 모드를 거의 동일한 다이 크기를 유지하면서 AI 워크로드에 맞게 조정된 모드와 정밀도로 대체합니다[35]. 라우팅 포트 제한을 완화하기 위해 새로운 텐서 블록은 데이터 재사용 레지스터 뱅크를 도입하여 이전 작업보다 훨씬 더 많은 수의 `int8` 승수를 장착할 수 있는 동시에 표준 Stratix 10 DSP 블록과 동일한 수의 입력/출력 포트를 유지합니다. 또 다른 작업 라인에서는 FPGA 패브릭을 별도의 AI 컴퓨팅 컴플렉스와 나란히 통합하는 것을 제안하는데, 여기에는 자일링스의 Versal 아키텍처[11]의 AI 엔진과 인텔의 시스템 인 패키지 에코시스템[12], [13]을 사용하는 특수 AI 칩렛 등이 있습니다. 반면 Stratix 10 NX는 인패브릭 텐서 블록을 통합하여 소프트 패브릭에 보다 유연하게 연결할 수 있고 기존 FPGA 개발 플로우를 사용합니다.

B. 스트라틱스 10 NX AI 텐서 블록

Stratix 10 NX 텐서 블록은 AI 워크로드를 위해 특별히 설계된 구성 가능한 기능 블록입니다. 기존의 스트라틱스 10 가변 정밀도 DSP 블록[36]을 대체하며 정수(`int8/int4`), 단정밀도 부동 소수점(`fp32`), 블록 부동 소수점, 브레인 부동 소수점 (`bfloat24/bfloat16`) 등 다양한 숫자 형식으로 스칼라, 벡터 및 텐서 연산 모드를 지원합니다. 추가 라우팅 포트 없이 텐서 블록당 매우 높은 승수 수를 구현하기 위해 Stratix 10 NX는 데이터 브로드캐스트와 직렬로 로드된 데이터 재사용 레지스터를 혼합하여 사용합니다. 이 블록을 최대한 활용하려면 애플리케이션의 연산을 블록에 매핑하는 방법을 신중하게 고려하여 매 사이클마다 모든 승수를 바쁘게 사용할 수 있는 충분한 피연산자를 확보해야 합니다. 다음으로, 이 논문에서 사용하는 `int8` 텐서 모드에 특히 초점을 맞춥니다. 스트라틱스 10 NX 장치에 대한 자세한 내용은 [35]를 참고하시기 바랍니다.

인텔 기술 개요[14]의 스트라틱스 10 NX AI 텐서 블록 다이어그램을 단순화한 버전으로 그림 2를 다시 만들어 `int8` 텐서 모드와 그 사용 방법을 강조했습니다. 텐서 블록에는 3개의 도트 프로덕트 유닛이 있으며, 각 유닛에는 10개의 8×8 곱셈기와 3개의 선택적 누산기가 있습니다. 두 개의 핑퐁 데이터 재사용 레지스터 뱅크(프리로드 버퍼)는 피연산자를 저장하는 데 사용되며, 블록의 데이터 입력 포트(프로그래밍 가능한 라우팅에 대한 10개의 8비트 연결을 통해)를 통해 또는 위의 텐서 블록에서 전용 체인을 통해 채워질 수 있습니다. 일반적으로 각 체인의 시작 부분에 있는 하나의 텐서 블록은
입력 바이패스를 사용하여 아래 텐서 블록의 핑퐁 레지스터 체인에 입력을 로드합니다. 체인의 나머지 텐서 블록의 경우, 각 도트곱 단위는 다음에서 피연산자 한 세트(30개)를 받습니다.

핑퐁 레지스터 뱅크와 다른 피연산자 세트(10개)가 데이터 입력 포트에서 직접 세 개의 도트 곱 유닛 모두에 브로드캐스트됩니다. 첫 번째 레지스터 뱅크가 피연산자를 도트 곱 유닛에 공급하는 동안 두 번째 뱅크는 위의 텐서 블록에서 로드(3사이클 이상)할 수 있습니다. 누산기 사이의 전용 하드 체인을 사용하여 여러 텐서 블록을 열에 캐스케이드하여 더 긴 도트 곱셈 단위를 효율적으로 구성할 수도 있습니다. 그림 3은 이 모드에서 AI 텐서 블록의 작동을 보여주는데, 컬러 원은 핑퐁 레지스터 뱅크로 이동되는 입력 벡터이고 컬러가 없는 도형은 3개의 도트 곱 단위로 브로드캐스트되는 텐서 블록의 데이터 포트에 입력되는 벡터입니다. 매 사이클마다 도트 곱셈 유닛에 공급되는 레지스터 뱅크는 점선으로 표시됩니다. 3 클럭 사이클이 지나면 해당 피연산자의 모양과 색상을 사용하여 그려진 출력이 생성됩니다.

IV. 스트라틱스 10 NX를 위한 향상된 NPU

A. *NPU 아키텍처 및 툴체인 개선 사항*

1) 행렬-벡터 곱셈 매핑: 그림 4a는 행렬-벡터 연산을 베이스라인 NPU에 타일 2개와 DPE가 있는 MVU에 매핑하는 것을 보여줍니다. 매트릭스는 수평으로 *T열 블록으로* 분할되어 각 타일이 하나의 열 블록을 담당합니다. 타일의 각 DPE는 여러 주기에 걸쳐 입력 벡터 블록과 행렬 행 블록 사이에서 도트 곱셈을 수행합니다. 그런 다음 매트릭스의 다음 *행 블록으로* 진행하기 전에 다른 타일에서 해당 DPE의 출력을 줄여 최종 결과를 생성합니다. Stratix 10 NX의 새로운 텐서 블록을 활용하려면 NPU의 MVU를 재구성해야 합니다. 그림 4b와 같이 3개의 서로 다른 입력 벡터 블록(3개의 도트 곱셈 유닛에 공급)을 레지스터 뱅크 중 하나(B0)에 로드하고, 다른 레지스터 뱅크(B1)에 입력 벡터의 다음 3개 블록을 로드하는 지연 시간을 숨길 수 있도록 충분한 매트릭스 행에 걸쳐 재사용합니다. B0과 B1은 그림 2의 뱅크0과 뱅크1에 해당한다는 것을 기억하세요. 이는 아키텍처가 3개의 입력 배치에서 작동하고 있음을 의미합니다. 그러나 이로 인해 여러 부분 결과의 누적이 인터리빙됩니다(대시 윤곽선으로 표시된 주황색, 노란색 및 보라색으로 표시됨). 를 사용하면 t = 1, 2, 3이 되며, 이 섹션의 뒷부분에서 설명하는 대로 누산기를 다시 설계해야 합니다.

다른 매핑은 매트릭스 블록을 레지스터 뱅크로 이동하고 대신 벡터 블록을 브로드캐스트하는 것입니다. 하지만 더 많은 텐서 블록을 캐스케이드하면 체인화된 레지스터 뱅크에 로드하는 지연 시간이 길어지므로 숨기려면 더 큰 배치 크기가 필요합니다. 예를 들어 40레인 DPE의 경우, 12개의 레지스터 체인(블록당 3개의 레지스터)에 대한 로딩 지연 시간을 숨기려면 4개의 텐서 블록을 캐스케이드하고 최소 12개의 입력 벡터를 배치 크기로 가져야 합니다. 따라서 첫 번째 매핑(그림 4b)을 사용하여 배치 크기를 낮게 유지하고 일반적으로 많은 수인 행렬 행을 브로드캐스트 입력으로 사용하여 로딩 지연 시간을 숨깁니다.

2) 누산기 설계: 그림 4b와 같이 여러 부분 결과의 누적을 인터리빙한 결과, MVU 누산기는 더 이상 단순한 레지스터와 가산기가 될 수 없습니다. 대신, 그림 4c와 같이 단일 가산기에 공급되는 인터리빙된 여러 부분 결과를 저장하기 위한 작은 스크래치 패드 역할을 하는 BRAM 기반 누산기를 구현합니다. 누산기는 다음과 같습니다.

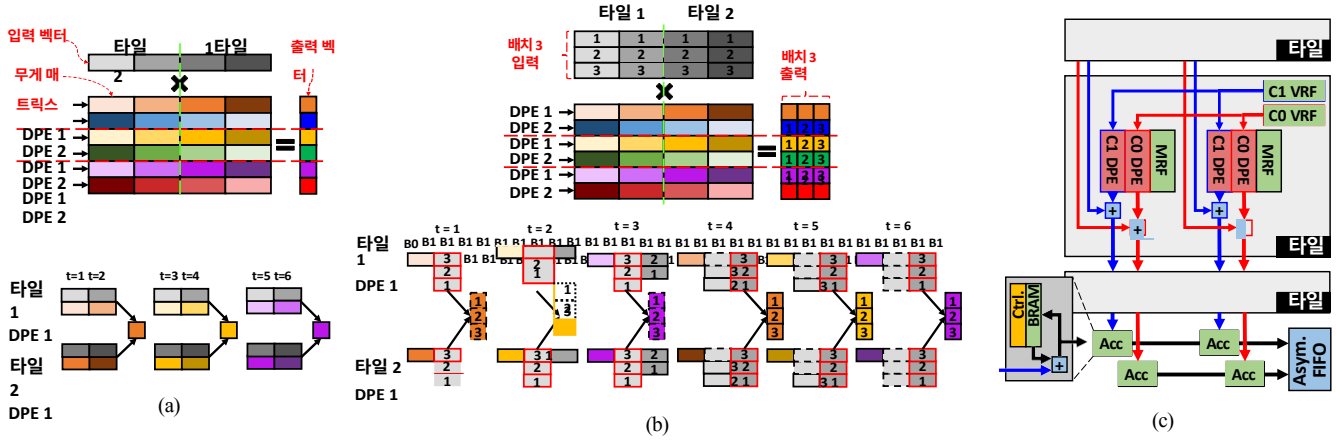

그림 4: 스트라틱스 10 NX의 (a) 베이스라인 NPU와 (b) 향상된 NPU의 행렬-벡터 곱셈 매핑. (c) BRAM 기반 누산기, 데이지 체인 상호 연결 및 동일한 MRF를 공유하는 2개의 코어를 갖춘 향상된 NPU의 MVU 아키텍처.

은 `int32` 정밀도로 수행되므로 부분 결과를 저장하는 데 몇 개의 BRAM만 필요합니다. 또한 타일 간 감소(그림 1 참조) 후 어큐뮬레이터를 이동하여 감소한 다음 누적합니다. 이렇게 하면 기본 NPU [13]에서처럼 각 타일마다 다른 누산기 세트를 사용하는 대신 모든 타일에 걸쳐 누산기 비용을 상각합니다.

3) 데이지 체인 타일: MVU 타일의 출력 대역폭은 3개의 입력 벡터 배치에서 작동하는 데 따른 부작용으로 기존 NPU에 비해 세 배로 증가합니다. 따라서 타일 간 감소를 수행하기 위해 각 타일에서 중앙 가산 트리로 넓은 버스를 라우팅하면 상당한 라우팅 혼잡이 발생할 수 있습니다. 예를 들어, 각각 3개의 32비트 출력을 생성하는 40개의 DPE가 있는 MVU 타일은 각 타일에서 중앙 가산기 감소 트리로 이동하는 3,840비트 와이드 버스를 생성합니다. 이로 인한 라우팅 혼잡을 완화하기 위해 각 타일이 이전 타일의 결과를 가져와 로컬 바이너리를 수행하는 데이지 체인 아키텍처를 갖도록 MVU를 재설계합니다.
를 줄이고 그 결과를 다음 타일로 전달합니다(그림 4c 참조). 이 아키텍처는 연속된 두 타일 사이에 더 짧고 국지적인 라우팅을 사용하며, 몇 사이클 더 높은 지연 시간을 감수하더라도 더 효율적이고 라우팅 친화적인 것으로 밝혀졌습니다[37].

4) 멀티코어 NPU: 강화된 NPU 구성을 선택하기 위해 주기 정확도가 높은 성능 시뮬레이터를 사용하여 설계 공간을 빠르게 탐색합니다. MVU 출력 대역폭(3 z *D* 요소)과 나머지 파이프라인 블록(*L* 요소) 간의 불일치로 인해 D가 *L보다* 훨씬 클 경우 성능 병목 현상이 발생할 수 있음을 발견했습니다(여기서 D와 L은 각각 DPE와 레인의 수). 따라서 모든 파이프라인 블록의 대역폭을 늘리기 위해 상당한 양의 로직 및 라우팅 리소스를 사용하지 않고도 대역폭 불일치를 최소화하기 위해 *D = L을* 유지하기로 결정합니다. 또한 다음을 발견합니다.
L = 40은 입력 바이패스로 사용되는 텐서 블록의 수(매)L/10¶ 체인 텐서 블록당 1)와 핑퐁 레지스터 체인을 로드하는 데 필요한 사이클 수(3z)L/10¶ 사이클) 사이에서 적절한 절충점을 제공합니다. 따라서 NPU 아키텍처를 확장하기 위해 타일 수 *T만* 자유 파라미터로 남게 됩니다. 그러나 한 차원에서만 너무 많은 병렬 처리를 도입하면 특히 소규모 워크로드의 경우 하드웨어 활용도가 낮아질 수 있습니다. 그 결과, 저희는 다음과 같은 개념을 도입했습니다.
코어를 NPU 아키텍처에 추가했습니다. 모든 코어가 퍼시스턴트 모델 가중치를 보유하는 동일한 MRF를 공유하고 단일 명령어 다중 스레드(SIMT) 방식으로 동일한 명령어 스트림을 실행한다는 점을 제외하면 NPU 코어는 그림 1의 5개 블록이 모두 포함된 완전한 NPU 파이프라인입니다. 설계 공간 검색을 기반으로 유니티는 하드웨어 활용도, 처리 지연 시간, FPGA 리소스 활용도 간에 좋은 균형을 제공하는 코어 2개, 타일 7개, DPE 40개, 레인 40개(2C-7T-40D-40L)의 NPU 아키텍처를 구현했습니다. 이 구성에서 각 코어는 배치 크기 3을 병렬로 처리합니다,

표 I: Stratix 10 NX, MX 및 GX 디바이스에서 가장 큰 NPU 오버레이의 구현 결과(TB: 텐서 블록).

	S10	NXS10 MX	2100S10 GX 2800
NPU Config.	2C-7T-40D-40L	1C-4T-80D-40L	1C-4T-120D-40L
승수 # 멀티플라이어	67,200	12,800	19,200
ALM	256,125 (37%)	399,506 (57%)	567,982 (61%)
BRAM	6,400 (93%)	6,428 (94%)	9,018 (77%)
결핵/DSP	3,600 (91%)	3,360 (85%)	4,880 (85%)
주파수(MHz)	300	290	275
피크 int8 TOPS*	40.3	7.4	10.6
TOPS/Mil.LE	19.45	3.58	3.84

 NPU 피크 TOPS는 NX, MX 및 GX 버전에서 각각 2240, 3200 및 4800인 MVU의 TB/DSP만을 사용하여 계산됩니다. 다른 TB/DSP는 MFU에서 사용되거나 NX에서 입력 바이패스로 사용됩니다.

그 결과 효과적인 배치 크기는 6입니다. 향상된 NPU의 새로운 MVU 아키텍처는 그림 4c에 나와 있습니다.

5) ISA 및 툴체인: 향상된 NPU에는 배치 3 연산을 위한 새로운 지침도 도입되었습니다. 이제 모든 파이프라인 블록의 mOP 정의에는 기본 NPU의 1개 대신 3개의 피연산자 벡터 주소를 위한 3개의 필드가 있습니다. 또한 핑퐁 레지스터 뱅크에 대한 레지스터 활성화와 도트 프로덕트 유닛에 공급할 입력 세트를 선택하는 믹스 선택 신호와 같은 텐서 블록의 로우레벨 제어 신호를 추가하도록 MVU μOP 정의가 변경되었습니다. C++ 성능 시뮬레이터도 신속하면서도 정확한 성능 추정을 위해 새로운 아키텍처 변경 사항을 반영하도록 수정되었습니다. 마지막으로 일괄 연산을 위해 DSL 및 NPU 컴파일러에 지원을 추가합니다. 이 접근 방식을 사용하면 애플리케이션 개발자가 텐서 블록에 대한 연산 매핑 및 저수준 제어 시퀀싱에 대해 걱정할 필요 없이 소프트웨어로만 FPGA의 텐서 블록을 프로그래밍할 수 있습니다.

B. 구현 결과

표 I은 텐서 블록이 없는 MX 2100 및 GX 2800 디바이스 모두에서 [13]의 기준 NPU와 비교하여 Stratix 10 NX 디바이스에서 향상된 NPU의 구현 결과를 나타냅니다. 세 디바이스 모두에 가장 빠른 패브릭 속도 등급을 사용했으며, 인텔에서 제공하는 Stratix 10 NX에 대한 디바이스 지원과 함께 인텔 쿼터스 20.1의 결과를 보고했습니다. NX의 향상된 NPU는 약간 더 높은 주파수에서 실행되며 다음을 달성합니다.
(비슷한 크기의) MX 및 (더 큰 크기의) GX 장치의 NPU에 비해 각각 5.25배 및 3.5배 더 높은 승수 수입니다. NX의 NPU가 사용하는 3600개의 텐서 블록 중 2240개(장치 텐서 블록의 57%)만이 MVU의 텐서 모드에서 사용되며, 이 블록만 계산하여 40.3 피크 TOPS를 계산하므로 이는 GPU 및 기본 NPU 사례와 가장 잘 일치하는 수치입니다. 나머지 텐서 블록은 MFU의 `int32` 요소별 곱셈(800블록)을 구현하는 데 사용되거나 텐서 블록 체인의 데이터 재사용 레지스터에 공급하기 위한 입력 바이패스로 사용됩니다.

표 II: 다양한 워크로드에 대한 Stratix 10 NX에서 향상된 NPU의 배치 6 성능 결과(MLP-5: 5계층 MLP).

워크로드	크기 (h)	단계 (t)	지연 시간(ms)	Eff. int8 TOPS	HW Util.
MLP-5	512	-	0.004	4.3	10.7%
	1024	-	0.005	12.3	30.4%
RNN	512	256	0.23	7.0	17.4%
	1024	256	0.33	19.1	47.2%
	1152	256	0.39	20.4	50.7%
	1536	256	0.49	29.1	72.2%
	1792	256	0.61	32.4	80.3%
GRU	512	256	0.59	8.1	20.1%
	1024	256	0.95	20.4	50.7%
	1152	256	1.1	22.6	55.9%
LSTM	512	256	0.51	12.7	31.6%
	1024	256	0.89	29.0	71.9%

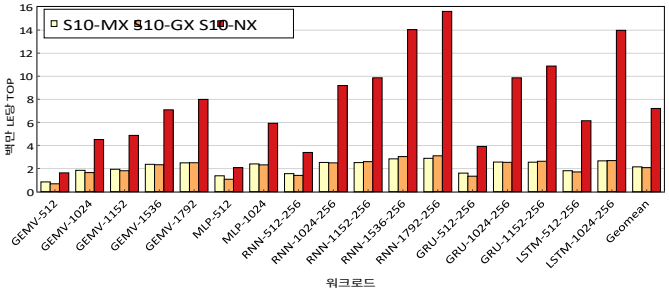

그림 5: GEMV 및 DL 워크로드에 대한 Stratix 10 MX, GX 및 NX 디바이스에서 NPU의 백만 LE당 유효 `int8` TOPS.

입력 캐스케이드(560블록)를 통해 처리합니다. 이는 저배치 지속적 AI 사용 사례에 맞게 조정된 NPU의 인공물입니다. 다른 애플리케이션은 클럭 섹터당 하나의 입력 바이패스 텐서 블록만 사용하여 더 긴 체인을 사용할 수 있습니다. 또한 애플리케이션이 BRAM 활용에 구속되지 않는 경우 벡터 요소별 곱셈을 소프트 로직에서 효율적으로 구현할 수 있으며[38], 텐서 블록은 더 고밀도 컴퓨팅 연산을 위해 확보할 수 있습니다. MX 및 GX NPU 버전에서는 다음과 같은 고밀도 패킹을 구현합니다.

결과 오염 복구 및 도트 곱셈 감소를 위해 소프트 로직이 있는 2개의 18비트 하드 멀티플라이어를 사용하여 DSP 블록당 공유 입력을 가진 4개의 `int8` 멀티플라이어를 사용합니다[39]. 네이티브 `int8`을 지원하는 텐서 블록은 DPE를 구현할 때 소프트 로직을 사용할 필요가 없습니다. 이는 다른 마이크로 아키텍처 최적화와 함께 소프트 로직 사용률을 MX 및 GX 구현에 비해 각각 36%와 55% 감소시킵니다.

표 II는 실시간 DL 워크로드에서 Stratix 10 NX 장치에서 퓨어 스토리지의 향상된 NPU의 배치 6 레이턴시, 유효 성능 및 하드웨어 사용률 결과를 보여줍니다. 유니티는 하드웨어 타임스탬프를 사용하여 입력 FIFO에서 입력을 소비하고, 모든 NPU 명령어를 실행하고, 출력을 출력 FIFO에 기록하는 사이클 수를 세는 방식으로 Stratix 10 NX 개발 키트에서 NPU 성능을 실험적으로 측정했습니다. 또한 이러한 측정값이 RTL 시뮬레이션에서 예상되는 사이클 수와 정확히 일치하는지 확인했습니다. 그 결과 문제 크기가 증가함에 따라 NPU의 유효 성능이 증가하여 최대 **32.4 TOPS**를 달성하고 NPU의 피크 처리량을 **80.3%** 활용한다는 것을 알 수 있었습니다. 또한 배치 크기를 6으로 사용했음에도 불구하고 여전히 매우 낮은 지연 시간을 달성했습니다. 시퀀스 길이가 256으로 매우 긴 가장 큰 GRU 워크로드는 **1.1ms밖에** 걸리지 않습니다. 이

표 III: 연구 중인 세 가지 AI 최적화 디바이스의 사양 요약: V100, T4, Stratix 10 NX.

	Nvidia V100†	Nvidia T4†	인텔 S10 NX‡
피크 FP32 TOPS	15.7	8.1	3.96
피크 FP16 탑	(125)	(65)	143*
피크 INT8 탑	62.8	(130)	143
온칩 메모리(MB)	16	10	16
프로세스 기술.	TSMC 12nm	TSMC 12nm	인텔 14nm
다이 크기(mm^2)	815	545	< 500 [43]

괄호 안의 퍼센트는 다음과 같습니다. 텐서 코어 FPGA 최고 성능(600MHz 기준)

블록 부동 소수점 사용 GPU용 레지스터 파일, FPGA용 M20K 사용

를 맥락에 맞게 설명하자면, 평균 문장 길이는 블록 디바이스와 같은 소자 수로 계산하여 영어는 15~20단어[40]로, 초당 11,600~15,500개의 문장에 대해 추론을 실행할 수 있습니다. 이는 추론에 배치 1 대신 배치 6을 사용하면서도 실시간 애플리케이션 요구 사항을 충족하는 동시에 더 높은 처리량과 하드웨어 활용도를 달성할 수 있음을 보여줍니다.

그림 5는 다양한 워크로드 세트에서 연구 중인 세 가지 NPU 버전의 성능을 보여줍니다. 결과를 정규화합니다.

디바이스 크기의 차이. 주어진 워크로드에 대해 장치 간 하드웨어 사용률이 동일하다고 가정할 때, 향상된 NPU로 달성할 수 있는 최대 성능 향상은 표 I의 피크 TOPS/LE에서 계산할 수 있습니다: MX 및 GX 디바이스에서 각각 5.4z 및 5.1z와 베이스라인 NPU 대비. 그림 5는 향상된 NPU가 기준 NPU에 매우 근접한 이득을 달성했음을 보여줍니다.

더 큰 워크로드에서 가능한 최대 이득(예: LSTM-1024에서 5.2z 및 5.1z, RNN-1792에서 5.4z 및 5.1z). 모든 연구 워크로드에서 MX 및 GX의 기본 NPU 대비 기하평균 속도 향상은 ~3.5z로, 이는 소규모 워크로드에서는 향상된 NPU의 활용도가 낮기 때문입니다.

V. 코어 컴퓨팅 벤치마킹

A. 실험 설정

이 섹션에서는 Stratix 10 NX AI 최적화 FPGA에 탑재된 향상된 NPU의 성능을 Nvidia의 최신 액세스 가능한 AI 최적화 GPU인 T4 및 V100 과 비교합니다. T4와 V100에는 각각 320개와 640개의 텐서 코어(AI 워크로드에 특화된 행렬 곱셈 엔진)가 있습니다[41], [42]. 표 III에는 Stratix 10 NX와 비교한 이들 GPU의 주요 사양이 요약되어 있습니다. 세 장치는 유사한 세대 프로세스 기술을 사용합니다. V100은 가장 큰 엔비디아 12nm GPU이며 T4보다 거의 50% 더 큰 반면, Stratix 10 NX는 두 GPU보다 작습니다[43].

먼저, GPU가 가장 선호하는 워크로드인 사각형 행렬-행렬 곱셈(GEMM)을 사용하여 GPU 마이크로 벤치마킹을 수행하여 이전 연구[44]에서와 유사한 GEMM 성능을 재현함으로써 설정이 최적의지 확인합니다. 최상의 GPU 성능을 측정하기 위해 각 장치에 대해 최신 라이브러리와 오차가 없는 정밀도를 사용하고 텐서 코어를 활성화한 상태와 활성화하지 않은 상태의 성능을 측정합니다. `fp32` 및 `fp16` 실험에서는 각각 V100 및 T4용 CUDA 10.0 및 10.2의 cuBLAS 라이브러리를 사용했습니다. `int8`의 경우, CUDA 10.2의 cuBLASLt 라이브러리를 사용하는데, 이는 cuBLAS보다 높은 `int8` 성능을 달성합니다[44]. DL 워크로드에는 엔비디아의 공식(고도로 최적화된) cuDNN 커널을 사용합니다. DL 워크로드에는 cuDNN 7.6.2와
7.6.5를 지원합니다. cuDNN 라이브러리는 `int8` 컴퓨팅 커널을 지원하지 않지만, 가능한 경우 모든 모델 가중치를 온칩 메모리에 유지하는 영구 모드를 지원하여 NPU 퍼시스턴트 방식과 유사하게 더 높은 성능을 제공합니다. 모든 워크로드, 문제 크기 및 시퀀스 길이에 대해 다음과 같은 측면에서 가능한 모든 구성을 철저하게 실행합니다.

정밀도{FP32, `FP16`, INT8}, 연산 스타일{퍼시스턴트, 비퍼시스턴트}, 텐서 코어 설정{활성화, 비활성화}을 두 GPU에 모두 적용합니다. 그런 다음 Stratix 10 NX의 NPU와 비교하기 위해 가장 우수한 성능을 선택합니다. 이 섹션의 모든 결과는 다음을 기록하는 `cudaEventRecord()` API를 사용하여 측정됩니다.

타임스탬프를 지정된 지점에서 GPU 장치에 기록합니다. 따라서 초기화, 커널 실행 또는 호스트-GPU 데이터 전송 오버헤드를 제외한 코어 연산만 고려하며, 이는 FPGA의 NPU 실행 주기와 정확히 일치합니다. 섹션 VI에서는 엔드투엔드 시스템 수준의 성능 평가를 수행하기 위해 GPU와 FPGA에 대한 이러한 모든 오버헤드를 포함합니다.

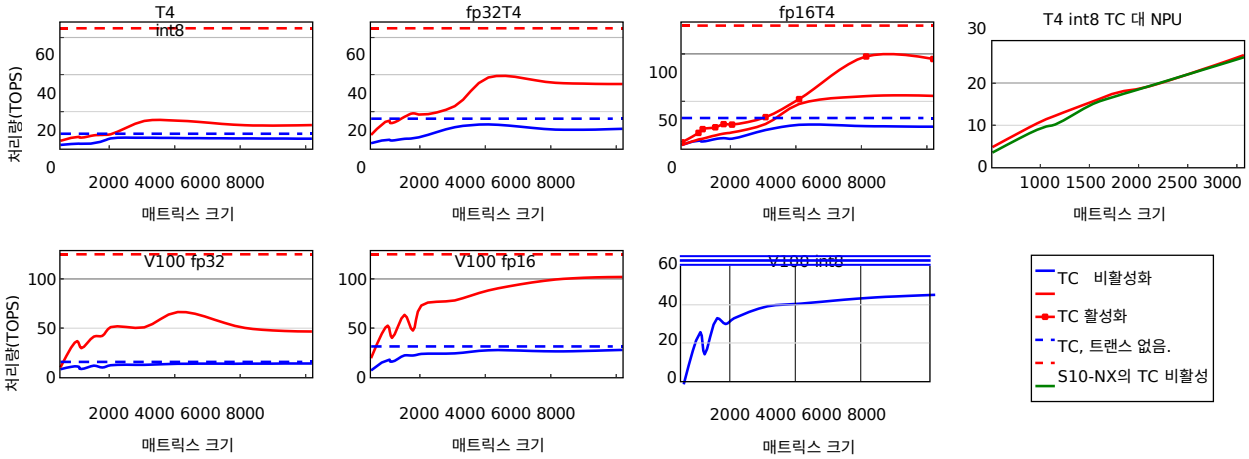

그림 6: 다양한 정밀도 및 텐서 코어(TC) 설정에서 T4 및 V100 GPU의 달성 가능 및 최고 GEMM 성능. V100 TC는 `int8`을 지원하지 않습니다. 오른쪽 상단 플롯은 NPU에서 영구적으로 유지할 수 있는 매트릭스 크기에 대한 T4와 NPU GEMM 성능을 비교한 것입니다.

B. 경쟁사 파악하기: GEMM 마이크로 벤치마킹

그림 6은 T4 및 V100 GPU에서 `fp32`, `fp16` 및 `int8` 정밀도에 대한 GEMM 마이크로 벤치마킹 결과를 보여줍니다. 결과에 따르면 텐서 코어는 텐서 코어가 비활성화된 경우(파란색 선)와 비교하여 GEMM(빨간색 선)에서 GPU의 성능을 크게 향상시킬 수 있음을 보여줍니다. 그러나 일반적인 추세는 텐서 코어가 GEMM용으로 설계되었음에도 불구하고 2048 이하의 매트릭스 크기에서 최고 성능(빨간색 점선)에 비해 여전히 활용도가 현저히 낮다는 것입니다. 실제 DL 워크로드에서 흔히 볼 수 없는 매우 큰 행렬 크기를 제외하고는 높은 활용도를 달성하지 못하며, 이는 [44]의 이전 연구 결과와 유사합니다. T4와 V100의 텐서 코어는 모두 `fp32`를 지원하지 않습니다.

대신 텐서 코어에서 곱셈 연산을 실행하기 전에 `fp32` 데이터를 `fp16`으로 변환합니다[45]. 이러한 데이터 변환 오버헤드는 순수 `fp16` GEMM에 비해 텐서 코어 성능을 저하시킵니다. 또 다른 흥미로운 관찰은 T4 텐서 코어가 `int8` 모드에서 작동할 때 입력 행렬을 표준 행/열 메이저 형식에서 특수 텐서 코어 전용 레이아웃으로 변환해야 한다는 것입니다[46]. 그 결과, 텐서 코어(마커가 없는 빨간색 선)에서 달성되는 `int8` 성능은 매우 큰 8192 x 8192 행렬을 처리할 때에도 최고 성능의 45% 미만입니다. 이 변환의 오버헤드를 더 잘 이해하기 위해, 저희는

텐서 코어 특수 레이아웃(마커가 있는 빨간색 선)에 입력 행렬을 공급하는 추가 실험. 행렬 레이아웃 변환 오버헤드가 없더라도 4096 z4096 이하의 크기에서는 텐서 코어 사용률이 40% 미만이며, 최대값은 다음과 같습니다.

6144 x 6144 매트릭스에서 72%의 사용률을 보였습니다.

하나의 행렬은 온칩에 영구적으로 유지하고 다른 행렬은 행 벡터 시퀀스로 스트리밍하는 방식으로 NPU에서 GEMM을 구현합니다. 그림 6의 오른쪽 상단 플롯은 Stratix 10 NX의 NPU 성능과 `int8` 텐서 코어가 있는 T4 GPU를 비교한 것입니다. 공정한 비교를 위해 두 입력 행렬 중 하나에 대한 매트릭스 레이아웃 변환을 비활성화했으며, 이는 NPU 측의 퍼시스턴트 행렬에 해당합니다. 그러나 두 번째 입력 및 출력 행렬에 대해서는 레이아웃 변환을 유지하는데, 이는 NPU가 이러한 행렬을 표준 형식으로 소비하고 생성하기 때문입니다. NPU는 행렬-벡터 연산을 위해 설계되었지만 512에서 3072(가장 큰 행렬을 처리할 수 있는 크기) 범위의 GEMM 워크로드에서 여전히 T4와 유사한 성능을 발휘합니다.

온칩 BRAM에 지속적으로 적합). 우리는 FPGA의 재구성을 통해 NX 텐서 블록에 잘 매핑되는 수축기 행렬 곱셈 유닛을 추가하여 GEMM용 NPU 오버레이를 커스터마이징할 수 있습니다. 그러나 여기서 연구하는 대상 워크로드에서는 대규모 GEMM을 사용하지 않기 때문에 이 작업의 범위에서 벗어납니다.

C. AI 워크로드에 대한 성능 비교

그림 7은 섹션 V-A에 자세히 설명된 구성의 모든 조합에서 GEMV 및 실시간 DL 워크로드에 대해 Stratix 10 NX의 향상된 NPU 성능을 최상의 T4 및 V100 성능과 비교한 것입니다. 3 및 6 사이즈의 소규모 배치에서 NPU 성능이 항상 두 GPU보다 훨씬 높습니다. NPU는 배치 6(원래 설계된 배치 3에서 각각 코어 2개)에서 가장 우수한 성능을 발휘하며, T4 및 V100에 비해 각각 평균 **24**.2z 및 **11**.7z 더 높은 성능을 제공합니다. 두 개의 코어 중 하나가 완전히 유휴 상태이기 때문에 배치 3에서는 배치 6에 비해 NPU의 성능이 떨어집니다. 그러나 여전히 T4 및 V100에 비해 각각 평균 **22**.3z 및 **9**.3z의 속도 향상을 달성합니다. 배치 크기가 6보다 큰 경우 배치 크기를 6으로 나눌 수 없는 경우 NPU를 제대로 활용하지 못할 수 있습니다. 예를 들어 배치 크기가 8, 32 및 256인 경우 NPU는 배치 6 성능의 최대 67%, 89% 및 99%를 달성할 수 있는 반면 배치 크기가 12, 36 및 258인 경우(그림 7의 점선으로 표시됨) 모두 100% 효율을 달성할 수 있습니다. 32개의 입력이 있는 중간 크기의 배치에서도 NPU는 여전히 T4보다 성능이 우수하며, V100보다 우수하거나 비슷한 성능을 제공합니다. 256개 크기의 대규모 배치에서도 NPU는 T4보다 58% 더 높은 성능을 제공합니다.
T4보다 더 강력하고 더 큰 V100보다 30% 적은 성능에 불과했습니다. 이러한 결과는 AI에 최적화된 FPGA가 로우 배치 실시간 추론에서 GPU보다 훨씬 뛰어난 성능을 달성할 수 있을 뿐만 아니라 지연 시간 제약이 완화된 하이 배치 추론에서도 경쟁할 수 있음을 보여줍니다. 그림 7의 오른쪽 하단 그래프는 다양한 배치 크기에서 T4 성능과 비교하여 연구된 모든 워크로드의 기하평균 속도 향상을 요약한 것입니다.

그림 7의 오른쪽 상단 플롯은 서로 다른 배치 크기에서 두 GPU와 비교한 NPU의 지오메인 활용도를 보여줍니다. NPU는 배치 6에서 37.1%의 지오메인 활용도를 달성한 반면, T4와 V100은 각각 1.5%와 3%에 그쳤습니다. GPU 텐서 코어는 서로 직접 연결되지 않으며[47], 로컬 인코어 레지스터 파일에서만 입력을 받을 수 있습니다. 따라서 각 GPU 텐서 코어는 부분 결과(예: RNN의 출력 벡터 하위 집합)를 글로벌 메모리로 전송하고 다른 텐서 코어와 동기화하여 이러한 부분 결과를 결합해야 합니다. 그런 다음 GPU는 글로벌 메모리에서 결합된 벡터를 읽어 활성화 함수와 같은 추가 작업을 수행합니다. 배치 크기가 클수록 이러한 동기화 지연 시간을 상각할 수 있지만, 배치 256에서도 GPU 사용률은 T4와 V100에서 각각 13.3%와 17.8%에 불과합니다. 반면에 FPGA는 텐서 블록 간의 전용 인터커넥트를 통해 이를 줄일 수 있습니다. 또한 FPGA의 프로그래머블 라우팅은 직접 공간 통신을 위해 MVU 타일과 벡터 요소별 엔진을 캐스케이딩할 수 있으므로 GPU의 경우처럼 메모리를 통한 통신이 필요하지 않습니다.

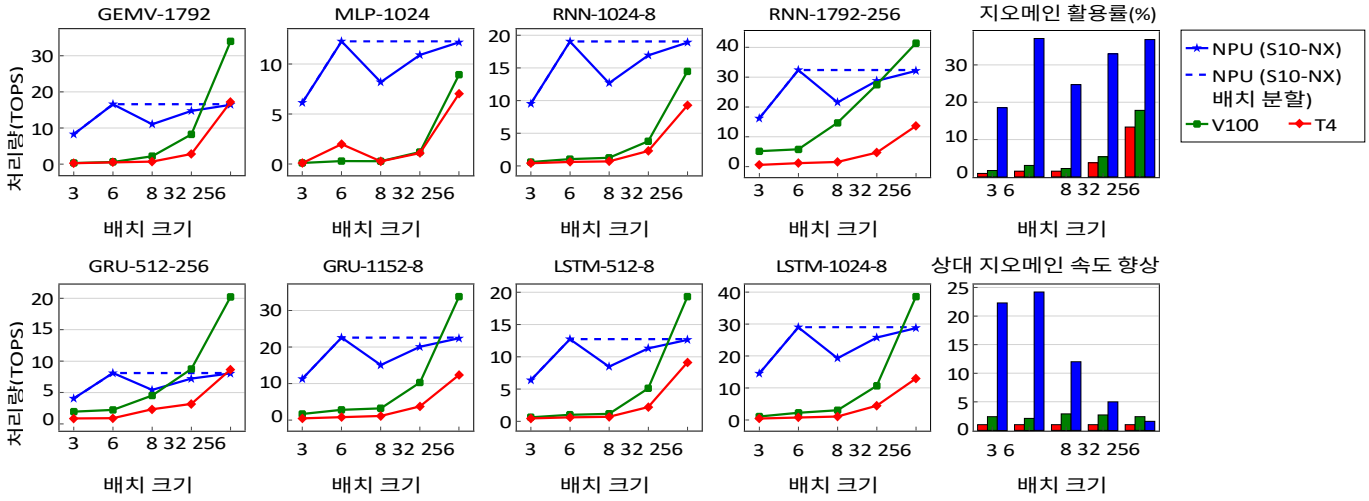

그림 7: 다양한 배치 크기에서 DL 벤치마크 제품군의 일부에 대한 Stratix 10 NX FPGA의 NPU 대비 V100 및 T4의 성능: 3, 6, 8, 32, 256. 점선은 6으로 나눌 수 있는 더 큰 배치 크기에 대한 NPU 성능을 나타냅니다. 모든 워크로드에서 배치 6 및 256에 대해 NPU가 T4보다 24.2z 및 1.6z의 속도 향상을 달성합니다(오른쪽 하단 플롯).

표 IV: GPU 및 FPGA 시스템 사양 요약.

	CPU	HostOS	호스트-장치 링크	BW (Gbps)
Nvidia T4 (AWS)	인텔 제온 플래티넘 8259CL @2.5GHz	Virt. Linux	PCIe Gen3x16	128
엔비디아 V100 (DGX)	인텔 제온 플래티넘 8168 @2.7GHz	Linux	PCIe Gen3x16	128
인텔 S10 MX/NX	인텔 제온 골드 6128 @3.6GHz	Linux	100G 이더넷†	100

□ 클라이언트 CPU와 해당 네트워크 인터페이스 카드(NIC) 사이
클라이언트 NIC와 FPGA 간 †

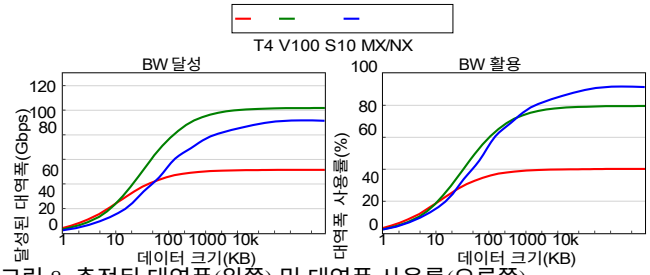

그림 8: 측정된 대역폭(왼쪽) 및 대역폭 사용률(오른쪽)

VI. 시스템 수준 벤치마킹

A. AI 가속화를 위한 GPU 및 FPGA 시스템

일반적인 GPU 기반 추론 시스템은 PCIe 인터페이스를 통해 GPU 카드에 연결된 호스트 CPU가 있는 서버로 구성됩니다. 원격 클라이언트에서 전송된 추론 요청은 먼저 서버의 NIC로 이동한 다음 호스트 CPU로 이동하고 마지막으로 GPU 카드로 이동합니다. FPGA 시스템의 경우, Microsoft의 Brainwave[25]와 유사하게 원격 클라이언트로부터 직접 입력을 받기 위해 FPGA에 긴밀하게 통합된 이더넷 인터페이스를 활용합니다. 실제 엔드투엔드 시스템 성능에 대한 구체적인 인사이트를 얻기 위해 앞서 언급한 프로토타입 GPU 및 FPGA 시스템을 표 IV에 자세히 설명된 대로 평가했습니다. 우리는 TACC [49]에서 V100 GPU에 물리적으로 액세스하고, AWS 가상 머신 인스턴스 [50]에서 T4 GPU에 액세스합니다.

B. 엔드투엔드 AI 추론 애플리케이션의 단계

표 V에서 볼 수 있듯이 엔드투엔드 AI 추론 워크로드는 단순히 가속기 카드의 코어 컴퓨팅을 넘어서는 것입니다. 일반적으로 일회성 초기화, 애플리케이션 입력 데이터 준비 및 가속기로 전송, 가속기 실행, 결과 전송으로 구성됩니다. FPGA 기반 시스템의 경우 원격 클라이언트 CPU는 최적화된 소프트웨어 라이브러리를 사용하여 NIC에 액세스해야 합니다. 본 연구에서는 데이터 플레인 개발 키트(DPDK)[48]를 사용합니다. 다른 한편으로, 우리는 페이징 오버헤드를 피하기 위해 `cudaMallocHost()`를 사용하여 GPU 장치로 전송할 호스트 메모리 공간을 고정하는 등 호스트 CPU와 GPU 카드 간의 상호 작용을 위해 엔비디아가 권장하는 방법을 사용합니다. 1단계는 추론 요청 횟수에 상관없이 한 번만 수행하면 되고, 2단계는 GPU와 FPGA 시스템 모두에서 공통적으로 수행됩니다. 따라서 본 연구에서는 3~5단계에만 초점을 맞춥니다. 공정한 비교를 위해 FPGA 시스템의 NPU 가중치는 온칩에 지속적으로 저장되므로 가중치를 위한 `cudaMemcpy()`는 평가에 포함하지 않았습니다.

이더넷 기반 FPGA 및 PCIe 기반 GPU 시스템에서.

C. 데이터 이동 효율성 특성 분석

먼저 이더넷을 통해 FPGA와 PCIe를 통해 GPU 간 데이터 이동의 효율성을 연구합니다. 인텔의 100G 이더넷 하드 IP[51]를 사용하여 루프백 로직이 FPGA에 구현되어 수신된 패킷을 즉시 CPU로 다시 보냅니다. 이 섹션의 실험에서는 NX 보드와 정확히 동일한 100G 이더넷 인터페이스를 갖춘 네트워크에 연결된 Stratix 10 MX 보드를 사용하여 FPGA 시스템 오버헤드를 평가합니다. 모든 시스템에서 호스트-장치 및 장치-호스트 데이터 전송 성능을 개별적으로 측정하고 그 평균을 구합니다. 그림 8에서 볼 수 있듯이 세 시스템에서 측정된 대역폭은 데이터 크기에 따라 처음에 증가하다가 포화 상태에 이릅니다. V100 GPU 시스템은 PCIe가 가장 높은 피크 대역폭을 제공하기 때문에 모든 데이터 크기에서 가장 높은 대역폭을 달성합니다. T4 시스템은 AWS 가상화 오버헤드(예: 가상 머신의 물리적 메모리에 고정된 메모리 공간에서 GPU 입력 데이터를 전송할 때)로 인해 더 낮은 대역폭을 실현하는 것으로 관찰되었습니다. FPGA는 최대 90%의 활용률로 100G 이더넷 인터페이스를 최대한 활용하는 반면, V100은 최대 128Gbps PCIe 대역폭의 80%까지만 활용할 수 있습니다.

D. AI 워크로드에 대한 엔드투엔드 성능 비교

그림 9는 배치 6 및 시퀀스 길이가 8(짧음) 및 256(길음)일 때 RNN 워크로드의 시스템 수준 실행 시간을 보여줍니다. 시스템 오버헤드를 포함한 후 FPGA 시스템은 짧은 입력 시퀀스를 가진 RNN에서 16~19z 및 15~25z의 속도 향상을 달성합니다.
T4에 비해 긴 시퀀스의 경우 11-16z, 5-6z입니다.
및 V100 시스템입니다. 이러한 속도 향상은 암달의 법칙으로 인해 섹션 V에서 보고된 코어 컴퓨팅 속도 향상보다 낮으며, FPGA 시스템은 GPU 시스템에 비해 커널 실행 시간과 시스템 오버헤드에 대해 동일한 속도 향상을 달성하지 못합니다. 더 많은 출력 데이터가 있는 긴 시퀀스의 경우, NPU는 스트리밍과 겹치기 때문에 시스템 오버헤드가 ~2z 더 낮습니다.

표 V: FPGA 및 GPU 시스템 모두에서 AI 추론 워크로드의 엔드투엔드 단계.

#	스테이지	FPGA 시스템	GPU 시스템
1	**일회성 초기화**	원격 CPU에 메모리를 할당합니다(예: DPDK [48]을 사용하여 NIC 패킷 버퍼 할당).	`cudaMallocHost()` 및 `cudaMalloc()`을 사용하여 로컬 CPU 호스트 및 GPU 디바이스에 메모리를 할당합니다.
2	**입력 준비**	입력 데이터로 원격 CPU 호스트 메모리를 초기화합니다(예: DPDK TX 버퍼).	입력 데이터로 로컬 CPU 호스트 메모리를 초기화합니다.
3	**가속기에 입력 보내기**	이더넷 패킷을 구성하고 NIC를 통해 전송하는 데는 DPDK API가 사용됩니다. RNN 워크로드의 경우, 숨겨진 벡터 초기화는 NPU에서 수행됩니다.	`cudaMemcpy()`를 사용하여 호스트 메모리에서 GPU 메모리로 데이터를 이동합니다. RNN 워크로드의 경우 호스트는 GPU 커널을 호출하여 숨겨진 벡터를 초기화합니다.
4	**가속기 실행**	FPGA는 이더넷에서 입력을 수신하고 NPU 실행을 트리거합니다. NPU 실행은 논블로킹이며, 준비가 되면 부분적인 결과(RNN의 한 타임스텝)를 다시 전송할 수 있습니다.	GPU 애플리케이션 라이브러리를 호출합니다(예: cuDNN의 RNN). 실행이 차단되고 있으며 GPU 실행이 완료된 후에만 최종 결과(예: RNN의 모든 시간 단계)를 전송할 수 있습니다.
5	**결과 다시 보내기**	원격 CPU 호스트는 NIC에서 결과를 수신합니다. DPDK를 사용하여 NIC 버퍼에 액세스합니다.	다음을 사용하여 GPU 메모리에서 호스트에 결과를 복사합니다. `cudaMemcpy()`.

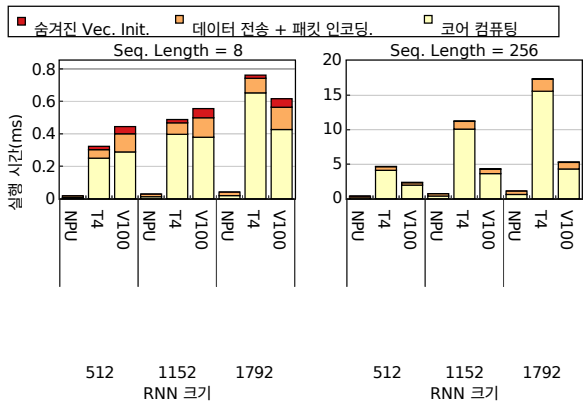

그림 9: 짧은(왼쪽) 시퀀스와 긴(오른쪽) 시퀀스에 대한 RNN 워크로드의 시스템 수준 실행 시간.

를 사용하면 단계별 결과를 다음 단계의 계산과 함께 출력할 수 있습니다. 이와 대조적으로, GPU의 사전 정의된 프로그래밍 모델은 CPU 호스트가 액세스하기 전에 모든 시간 단계의 결과를 계산해야 합니다. 또한 GPU는 커널 호출을 통해 상태를 초기화해야 하지만[52], FPGA의 유연한 프로그래밍 모델에서는 이러한 오버헤드가 발생하지 않습니다. 짧은 시퀀스의 경우 데이터 전송 및 커널 실행 시간이 줄어들면서 이러한 오버헤드가 두드러지게 나타납니다. 따라서 이 경우 FPGA 시스템 오버헤드는 훨씬 더 작습니다(~5z 및
T4 및 V100보다 각각 ~10z 더 우수합니다.

VII. 전력 분석

GPU의 경우, GPU 다이 온도[53]뿐만 아니라 전체 GPU 카드(팩 에이징된 GPU 칩과 메모리 및 팬과 같은 기타 구성 요소 포함)의 전력을 보고하는 Nvidia SMI 도구로 전력을 측정합니다. 당사가 사용하는 GPU에는 강력한 냉각 솔루션(예: AWS의 프로덕션 등급 냉각 및 TACC의 침수 냉각[54])이 적용되어 있으므로, Nvidia SMI 툴은 V100 및 T4의 코어 GPU 온도를 각각 35° C 및 40° C로 보고합니다. 우리가 연구한 모든 워크로드에서 GPU의 활용도가 현저히 낮기 때문에 측정된 GPU 전력은 T4의 경우 27-45W, V100의 경우 35-72W 범위입니다. 그러나 8192 z8192 GEMM과 같이 GPU에서 높은 활용도를 달성하는 워크로드를 실행할 경우 전력 소비는 다음과 같습니다.
T4와 V100의 경우 각각 최대 70W와 190W까지 올라갑니다.

또한 고해상도 파워 미터를 사용하여 Stratix 10 NX 개발 키트

6 추론은 연구 워크로드에서 T4 및 V100 GPU에 비해 각각 12-16z 및 8-12z 더 높은 평균 에너지 효율(즉, TOPS/Watt)을 달성합니다.

VIII. 관련 작업

섹션 II와 III에서는 각각 DL을 위한 기본 NPU 오버레이와 다양한 FPGA 아키텍처 최적화에 대한 관련 작업에 대해 설명했습니다. 이 백서에서는 다양한 변형 RNN 모델에서 새로운 Stratix 10 NX FPGA의 성능을 평가했습니다. 이전의 여러 연구들은 RNN 가속화에 중점을 두었습니다.
FPGA [5], [13], [55]-[59]에서. 향상된 NPU를 활용한
NX 텐서 블록은 FPGA 기반 RNN의 모든 이전 작업을 능가합니다.
의 전력을 측정합니다. 우리가 사용하는 개발 키트에는 공랭식 방열판과 온보드 온도 센서가 있습니다. 모든 실험은 특별한 냉각 솔루션 없이 실온의 주변 온도와 실험실 책상 위에 놓인 FPGA 보드에서 수행합니다. GEMV 및 DL 워크로드를 무한 루프에서 실행하도록 NPU를 구성하고 각 워크로드를 몇 분간 연속 실행한 후 전력 및 온도 측정값을 기록합니다. 측정된 전력은 실행 중인 워크로드의 NPU 사용률에 따라 35-39° C의 측정된 보드 온도에서 54-70W 범위입니다(표 II에 표시된 대로). 이러한 결과는 Stratix 10 NX NPU가 일괄 처리를 실행하는 경우 다음과 같이 나타납니다.

가속도, 심지어 `int8` 정밀도보다 낮은 정밀도를 사용하는 워크로드에서도 마찬가지입니다. 일부 선행 연구에서도 AI 워크로드에서 FPGA와 GPU의 성능을 비교한 바 있습니다[60]-[62]. 그러나 이번 연구는 텐서 코어를 갖춘 최신 액세스 가능한 AI 최적화 GPU와 비교하여 텐서 블록이 통합된 AI 최적화 FPGA인 Stratix 10 NX의 성능을 평가한 최초의 연구입니다. 비슷한 최고 TOPS에서 FPGA의 유연한 아키텍처가 GPU보다 훨씬 더 높은 텐서 유닛의 활용도를 제공할 수 있음을 보여줍니다. 또한, 이전 연구에서는 호스트 대 FPGA PCIe 통신 오버헤드 [15], [63] 및 GPU 시스템 오버헤드 [64]에 대해서도 연구했습니다. 본 연구에서는 특히 실시간 DL 워크로드에 대해 PCIe 연결 GPU와 비교하여 100G 이더넷 연결 FPGA 시스템 오버헤드를 연구합니다.

IX. 결론

이 연구에서는 텐서 블록이 탑재된 인텔의 AI 최적화 Stratix 10 NX FPGA의 성능을 최신 액세스 가능한 AI 최적화 엔비디아 GPU인 T4 및 V100과 비교한 첫 번째 평가를 발표했습니다. 유니티는 NX의 텐서 블록을 가장 잘 활용하도록 계산을 재구성하기 위해 이전 Brainwave NPU 아키텍처, ISA 및 툴체인에 대한 개선 사항을 제안했습니다. Stratix 10 NX의 향상된 NPU는 주요 AI 워크로드에서 평균적으로 Stratix 10 MX/GX의 기본 NPU(텐서 블록 제외)보다 LE당 3.5배 높은 성능을 달성하며, 최대 80%의 NPU 활용도를 제공합니다. 반면, 간단한 정사각형 GEMM 벤치마크에서도 GPU 텐서 코어의 활용도가 현저히 낮을 수 있음을 보여줍니다. Stratix 10 NX의 향상된 NPU는 더 작은 NX 다이 크기에도 불구하고 배치 6에서 T4 및 V100 GPU에 비해 평균적으로 24배 및 12배 더 높은 코어 컴퓨팅 성능을 제공합니다. 배치 크기가 256인 대규모 배치에서도 NPU는 T4보다 58% 더 높은 성능을 달성하고, 더 큰 V100보다 30% 더 낮은 성능을 제공합니다. 마지막으로, 일반적으로 간과되는 엔드투엔드 시스템 수준의 오버헤드를 FPGA 기반 및 GPU 기반 AI 추론 시스템 모두에서 평가합니다. 짧고 긴 입력 시퀀스가 있는 RNN 워크로드에서 FPGA의 통합 100G 이더넷이 V100 GPU의 128Gbps PCIe 인터페이스에 비해 오버헤드가 10초 및 2초 더 적다는 것을 보여줍니다. 모든 시스템 수준 오버헤드를 포함하여 Stratix 10 NX의 NPU는 연구 대상 GPU에 비해 평균적으로 약 1.5배의 속도 향상을 제공합니다.

참조

[1] I. Akkaya *외*, "로봇 손으로 루빅스 큐브 풀기", *arXiv 사전 인쇄물 arXiv:1910.07113*, 2019.

[2] A. Radford *외*, "언어 모델은 비지도 멀티태스크 학습자", *OpenAI 블로그*, 1권, 8호, 9페이지, 2019.

[3] D. 실버 *외*, "인간 지식 없이 바둑 게임 마스터하기", *Nature*, 550, 7676호, 354-359쪽, 2017.

[4] C. Berner *외*, "대규모 심층 강화 학습을 사용한 DOTA 2," *arXiv preprint arXiv:1912.06680*, 2019.

[5] J. Fowers *외*, "실시간 AI를 위한 구성 가능한 클라우드 스케일 DNN 프로세서", *국제 컴퓨터 아키텍처 심포지엄(ISCA)*, 2018, 1-14쪽.

[6] N. Jouppi *외*, "데이터센터 내 텐서 처리 장치의 성능 분석", *국제 컴퓨터 아키텍처 심포지엄(ISCA)*, 2017, 1-12쪽.

[7] K. 헤이즐우드 *외*, "Applied Machine Learning at Facebook: 데이터센터 인프라 관점", *고성능 컴퓨터 아키텍처 국제 심포지엄(HPCA)*, 2018, 620-629쪽.

[8] Dale Southard, "컴퓨팅 집약적인 워크로드에 탁월한 성능을 제공하는 텐서 스트리밍 아키텍처," 2019.

[9] Z. Jia *외*, "Mi-크로 벤치마킹을 통한 그래프코어 IPU 아키텍처 해부", *arXiv 사전 인쇄물 arXiv:1912.03413*, 2019.

[10] D. Abts *외*, "Think Fast: 딥 러닝 워크로드 가속화를 위한 텐서 스트리밍 프로세서(TSP)", *국제 컴퓨터 아키텍처 심포지엄(ISCA)*, 2020, 145-158쪽.

[11] B. Gaide *외*, "Xilinx 적응형 컴퓨팅 가속화 플랫폼: Versal 아키텍처", *국제 필드 프로그래머블 게이트 어레이(FPGA) 심포지엄*, 2019, 84-93쪽.

[12] E. Nurvitadhi *외*, "인텔® 스트라틱스® 10 FPGA를 위한 인-패키지 도메인별 ASIC: TensorTile ASIC을 사용한 딥 러닝 가속화 사례 연구", *국제 필드 프로그래머블 로직 및 애플리케이션 컨퍼런스(FPL)*, 2018, 106-1064쪽.

[13] --, "함께 일할 수 있는데 왜 경쟁해야 하는가: 퍼시스턴트 RNN을 위한 FPGA-ASIC 통합", *국제 현장 프로그래밍 가능 맞춤형 컴퓨팅 머신 심포지엄(FCCM)*, 2019, 199-207쪽.

[14] Intel Corp., "Intel Stratix 10 NX FPGA: 고대역폭, 저지연 AI 가속을 위한 AI 최적화 FPGA(SS-1121-2.0)", 2020.

[15] E. 누르비타디 *외*, "다중 FPGA를 갖춘 CPU 서버에서 확장 가능한 저지연 지속적 신경망 기계 번역", *필드 프로그래머블 기술(FPT) 국제 컨퍼런스*, 2019, 307-310페이지.

[16] M. Jain *외*, "향상된 빔 검색으로 지연 시간 제어 ASR을 위한 RNN-T," *arXiv 사전 인쇄본 arXiv:1911.01629*, 2019.

[17] T. Hastie *외*, *통계적 학습의 요소: 데이터 마이닝, 추론 및 예측*. Springer 과학 및 비즈니스 미디어, 2009.

[18] M. Naumov *외*, "개인화 및 추천 시스템을 위한 딥 러닝 추천 모델", *arXiv preprint arXiv:1906.00091*, 2019.

[19] I. 굿펠로우 *외*, *딥러닝*. MIT press, 2016.

[20] K. Cho *외*, "통계적 기계 번역을 위해 RNN 인코더-디코더를 사용한 구문 표현 학습", *arXiv preprint arXiv:1406.1078*, 2014.

[21] S. Hochreiter와 J. Schmidhuber, "장단기 기억", *신경 계산*, 9권, 8호, 1735-1780쪽, 1997.

[22] V. J. Reddi *외*, "MLPerf 추론 벤치마크", *국제 컴퓨터 아키텍처 심포지엄(ISCA)*, 2020, 446-459쪽.

[23] *딥벤치*, (2020 년 8 월 5 일 액세스). [온라인]. Available: https://github.com/baidu-research/DeepBench

[24] F. Zhu *외*, "Sparse Persistent RNNs: Squeezing Large Recurrent Networks On-chip," *arXiv preprint arXiv:1804.10223*, 2018.

[25] E. Chung *외*, "프로젝트 브레인웨이브를 통한 데이터센터 규모의 실시간 DNN 서비스," *IEEE Micro*, 38권 2호, 8-20쪽, 2018.

[26] J. Fowers *외*, "프로젝트 브레인웨이브의 클라우드 스케일 실시간 AI 프로세서 내부", *IEEE Micro*, vol. 3, pp. 20-28, 2019.

[27] A. 부트로스 *외,* "수학이 어려울 필요는 없다: FPGA에서 저정밀 곱셈-누적을 향상시키는 논리 블록 아키텍처", *필드 프로그래머블 게이트 어레이(FPGA) 국제 심포지엄*, 2019, 94-103쪽.

[28] M. Eldafrawy *외*, "효율적인 딥 러닝 추론을 위한 FPGA 로직 블록 아키텍처," *ACM Transactions on Reconfigurable Tech- nology and Systems (TRETS)*, 13권, no. 3, pp. 1-34, 2020.

[29] S. Rasoulinezhad *외*, "LUXOR: 효율적인 압축기 트리 구현을 위한 FPGA 로직 셀 아키텍처", *국제 필드 프로그래머블 게이트 어레이 (FPGA) 심포지엄*, 2020, 161- 171페이지.

[30] A. 부트로스 *외*, "다양성 수용: *FPGA의* 저정밀 딥 러닝을 위한 향상된 DSP 블록", *국제 필드 프로그래머블 논리 및 애플리케이션 컨퍼런스(FPL)*, 2018, 35-357쪽.

[31] 인텔 주식회사, "인텔 애질렉스 가변 정밀 DSP 블록 사용 설명서 (UG-20213)", 2020.

[32] S. Rasoulinezhad *외,* "PIR-DSP: 다중 정밀도 심층 신경망을 위한 FPGA DSP 블록 아키텍처," *국제 현장 프로그래밍 가능 맞춤형 컴퓨팅 머신 심포지엄(FCCM)*, 2019,
pp. 35-44.

[33] Achronix Semiconductor Corp., "Speedster7t 머신 러닝 프로세싱 사용자 가이드(UG088)."

[34] A. Arora *외*, "Hamamu: 하드 매트릭스 배율기 블록을 추가하여 ML 애플리케이션을 위한 FPGA 전문화", *애플리케이션별 시스템, 아키텍처 및 프로세서에 대한 국제 컨퍼런스(ASAP)*, 2020, 53-60페이지.

[35] L. 그웬납, *스트라틱스 10 NX에 AI 블록 추가*, 2020 (2020 년 8 월 5 일 액세스). [온라인]. 사용 가능: https://www.linleygroup.com/ 뉴스레터/뉴스레터 상세정보.php?num=6183&year=2020&tag=3

[36] 인텔 주식회사, "인텔 스트라틱스 10 가변 정밀도 DSP 블록 사용 설명서(UG-S10-DSP)", 2020.

[37] A. Boutros *외*, "측정하지 않는 것을 개선할 수 없다: 컨볼루션 신경망 인퍼런스를 위한 FPGA와 ASIC의 효율성 격차", *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 3, pp. 1-23, 2018.

[38] M. Langhammer *외*, "프랙탈 합성: 초청 튜토리얼", *국 제 필드 프로그래머블 게이트 어레이(FPGA) 심포지엄*, 2019,
pp. 202-211.

[39] M. Langhammer 외, "INT18 곱셈기에서 INT8 곱셈기 추출하기", *국제 현장 프로그래밍 가능 논리 및 응용 컨퍼런스(FPL)*, 2019, 114-120쪽.

[40] A. 무어, "긴 문장: 인터넷 시대의 과학에 대한 불신," *BioEssays*, 33 권 12호, 193-193쪽, 2011.

[41] Nvidia Corp., "Nvidia T4 텐서 코어 GPU(장치 데이터시트)", 2019.

[42] --, "Nvidia Tesla V100 GPU 아키텍처(WP-08608-001)", 2017.

[43] *인텔과의 비공개 커뮤니케이션*.

[44] Z. Jia *외*, "마이크로벤치 마킹을 통한 엔비디아 튜링 T4 GPU 해부", *arXiv 사전 인쇄물 arXiv:1903.07486*, 2019.

[45] *엔비디아 딥 러닝 SDK 설명서*, (2020년 8월 5일 액세스). [온라인]. 사용 가능: https://docs.nvidia.com/deeplearning/sdk/cudnn-archived/cudnn 765/cudnn-api/index.html#cudnnSetRNNMatrixMathType

[46] *엔비디아 CUDA 툴킷문서*, (ac- cessed 8월 5, 2020). [온라인]. 사용 가능: https://docs.nvidia.com/cuda/archive/10.2/cublas/index.html#cublasLt-example-tensorop

[47] Nvidia Corp., "Nvida Turing GPU 아키텍처 백서(WP- 09183-001 v01)".

[48] *데이터 플레인 개발 키트(DPDK)*, (2020년 8월 5일 액세스). [온라인]. 사용 가능: https://www.dpdk.org/

[49] T. U. 텍사스 오스틴, *텍사스 고급 컴퓨팅 센터*. [온라인]. 이용 가능: https://www.tacc.utexas.edu/

[50] 아마존, *아마존 EC2 G4 인스턴스*. [온라인]. 사용 가능: https://aws.amazon.com/ec2/instance-types/g4/

[51] Intel Corp., "Intel Stratix 10 H-타일 이더넷 IP 코어용 하드 IP 사용 설명서(UG-2021)", 2018.

[52] *CUDA C++ 프로그래밍 가이드*, (2020년 8월 5일 액세스). [온라인]. 사용 가능: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[53] *NVIDIA 시스템 관리 인터페이스*, (2020년 8월 5일 액세스). [온라인]. 사용 가능: https://developer.nvidia.com/nvidia-system- 관리 인터페이스

[54] *TACC의 매버릭2 냉각 시스템*. [온라인]. Available: https://portal.tacc.utexas.edu/documents/10157/1181317/Maverick2+cooling+system/fbef2f24-4252-4d5f-857e-73c138ff6a0e?t=1592320888902

[55] Y. Guan *외*, "장단기 메모리 순환 신경망을 위한 FPGA 기반 가속기", *아시아 및 남태평양 디자인 자동화 컨퍼런스(ASP-DAC)*. IEEE, 2017, 629-634쪽.

[56] S. Han *외*, "ESE: 효율적인 음성 인식 엔진: FPGA에서 스파스 LSTM을 사용한 음성 인식 엔진", *국제 현장 프로그래밍 가능 게이트 어레이 (FPGA) 심포지엄*, 2017, 75-84쪽.

[57] V. Rybalkin *외.*, "FINN-L: FPGA의 가변 정밀도 LSTM 네트워크를 위한 라이브러리 확장 및 설계 트레이드 오프 분석", *국제 필드 프로그래머블 로직 및 애플리케이션 컨퍼런스(FPL)*. IEEE, 2018, 89-897쪽.

[58] Z. Que *외.*, "재구성 가능한 순환 신경망 최적화", *현장 프로그래밍 가능 맞춤형 컴퓨팅 머신 국제 심포지엄(FCCM)*. IEEE, 2020, 10-18쪽.

[59] V. Rybalkin *외*, "1D 및 MD- LSTM 네트워크를 위한 효율적인 하드웨어 아키텍처", *신호 처리 시스템 저널,* 1-27쪽, 2020.

[60] E. 누비타디 *외*, "지속적 실시간 AI를 위한 인텔 스트라틱스 10 FPGA 평가 및 향상", *필드 프로그래머블 게이트 어레이(FPGA) 국제 심포지엄*, 2019, 119-119쪽.

[61] --, "차세대 딥 뉴럴 네트워크를 가속화하는 데 있어 FPGA가 GPU를 이길 수 있을까?", *Field-Programmable Gate Array(FPGA) 국제 심포지엄*, 2017, 5-14페이지.

[62] --, "분석 서버에서 순환 신경망 가속화: FPGA, CPU, GPU, ASIC의 비교", *국제 필드 프로그래머블 로직 및 애플리케이션 컨퍼런스(FPL)*에서. IEEE, 2016, 1-4쪽.

[63] D. J. Moss *외*, "인텔 HARPv2 제온+FPGA 플랫폼을 위한 맞춤형 매트릭스 곱셈 프레임워크: 딥 러닝 사례 연구", *국제 현장 프로그래밍 가능 게이트 어레이(FPGA) 심포지엄*, 2018, 107-116쪽.

[64] D. Lustig 및 M. Martonosi, "세분화된 CPU-GPU 동기화를 통한 GPU 오프로드 지연 시간 감소", *고성능 컴퓨터 아키텍처에 관한 국제 심포지엄(HPCA)*, 2013, 354-365쪽.