실시간 AI를 위한 구성 가능한 클라우드 규모 DNN 프로세서

제레미 파워스 칼린 옵챠로프 마이클 파파마이클 토드 매센길 다니엘 로 슬로미 알칼레이 마이클 하셀만 로건 아담스 마흐디 간디 스티븐 하일 프레락 파텔 아담 사펙 가브리엘 와이즈 리사 우즈 시타람 랑카스티븐 K. 라인하르트아드리안 M. 콜필드에릭 S. 정더그 버거

Microsoft

요약 - *대화형* AI 기반 서비스에는 심층 신경망(DNN) 모델, 즉 '실시간 AI'의 저지연 평가가 필요합니다. 범용 아키텍처의 성능 향상이 감소하는 것과 더불어 계산 비용이 많이 드는 최신 DNN에 대한 수요가 증가함에 따라 특수 신경 처리 장치(NPU)가 폭발적으로 증가했습니다. 대화형 서비스를 위한 NPU는 두 가지 요구 사항을 충족해야 합니다: (1) 짧은 지연 시간, 높은 처리량, 고효율의 DNN 모델 실행, (2) 고비용의 실리콘 업데이트 없이 진화하는 최신 모델(예: RNN, CNN, MLP)을 수용할 수 있는 유연성입니다.

이 백서에서는 실시간 AI를 위한 프로덕션 규모 시스템인 프로젝트 브레인웨이브(Project Brain- wave)의 NPU 아키텍처에 대해 설명합니다. 브레인웨이브 NPU는 배치 크기 1의 대규모 RNN에서 최신 GPU에 비해 지연 시간과 처리량이 10배 이상 개선되었습니다. 단일 명령어에서 7백만 개 이상의 연산을 처리할 수 있는 분산 마이크로아키텍처와 결합된 싱글 스레드 SIMD ISA를 사용하여 이러한 성능을 달성했습니다. 공간적으로 분산된 마이크로 아키텍처는 최대 96,000개의 곱셈-누적 유닛으로 확장되며, 계층적 명령 디코더 및 스케줄러와 독립적으로 주소 지정이 가능한 수천 개의 고대역폭 온칩 메모리가 결합되어 여러 수준의 세분화된 SIMD 병렬 처리를 투명하게 활용할 수 있습니다. FPGA를 타깃으로 하는 경우, 네이티브 데이터 경로 및 수치 정밀도와 같은 마이크로 아키텍처 파라미터를 컴파일 시 모델에 '합성 특화'할 수 있으므로 강화된 NPU와 경쟁할 수 있는 높은 FPGA 성능을 구현할 수 있습니다. 인텔 스트라틱스 10 280 FPGA에서 실행할 경우, 브레인웨이브 NPU는 메모리 집약적인 대규모 RNN에서 배치 없이 10테라플롭에서 35테라플롭 이상의 성능을 달성할 수 있습니다.

***색인* 용어-신경망 하드웨어; 가속기 아키텍텍-tures; field programmable gate arrays**

I. 소개

DNN 모델의 계산 복잡성이 증가함에 따라 심층 신경망(DNN)의 하드웨어 가속화가 일반화되고 있습니다. 범용 CPU에 비해 가속기는 최첨단 모델을 훈련하고 제공하는 데 드는 비용과 지연 시간을 모두 줄여줍니다. 다행히도 DNN 모델에서 사용할 수 있는 높은 수준의 병렬 처리 덕분에 실리콘 가속을 사용할 수 있습니다. 진화하는 DNN 전용 기능을 통해 GPGPU는 특히 DNN 워크로드를 가속화하는 데 성공했습니다. 또한 학계 연구자, 스타트업, 대기업을 중심으로 새로운 신경 처리 장치(NPU) 아키텍처의 캄브리아기 폭발이 일어나고 있습니다. 그러나 훈련과 추론(훈련된 모델을 평가)은 서로 다른 요구 사항을 가지고 있습니다. 훈련은 주로 처리량에 제한이 있 는 워크로드이며 다음과 같은 지연 시간에 민감하지 않습니다.

단일 샘플을 처리합니다. 반면에 추론은 지연 시간에 훨씬 더 민감할 수 있습니다. DNN은 웹 검색, 광고, 대화형 음성, 실시간 비디오(예: 자율주행차)와 같은 실시간 대화형 서비스에서 점점 더 많이 사용되고 있으며, 원활한 사용자 경험을 제공하고, 서비스 수준 계약(SLA)을 충족하고, 안전 요건을 충족하기 위해 짧은 지연 시간이 요구됩니다.

GPGPU와 같이 딥 파이프라인을 갖춘 고도로 병렬화된 아키텍처는 평가를 일괄 처리하여 요청 내 및 요청 간 병렬성을 활용함으로써 DNN 모델에서 높은 처리량을 달성합니다. 이 접근 방식은 학습 데이터 세트를 '미니배치'로 분할하여 일관성에 큰 영향을 주지 않으면서 처리량을 높일 수 있는 오프라인 학습에 효과적입니다. 그러나 배치 처리량에 최적화된 시스템은 일반적으로 리소스의 일부만 단일 요청에 적용할 수 있습니다. 온라인 추론 환경에서는 요청이 한 번에 하나씩 도착하는 경우가 많으므로 처리량 아키텍처는 이러한 요청을 개별적으로 처리하여 처리량을 줄이면서도 배치에 상응하는 지연 시간을 유지하거나, 여러 요청이 도착하여 배치가 형성될 때까지 기다리면서 지연 시간을 늘려야 합니다.

저희는 다른 접근 방식을 사용하는 DNN 인퍼런스를 위한 전체 시스템 아키텍처를 개발했습니다 [1], [2]. 이 시스템은 요청 간 병렬 처리를 활용하여 지연 시간을 희생하면서 처리량을 높이는 대신, 개별 요청에서 최대한 많은 병렬 처리를 추출하여 지연 시간을 줄입니다. 처리량을 희생하지 않고 단일 요청 지연 시간을 줄임으로써 처리량을 직접적으로 향상시킬 수 있습니다. 일괄 처리 없이 DNN 추론을 설명하기 위해 "실시간 AI"라는 용어를 사용합니다. 프로젝트 브레인웨이브(줄여서 BW)라고 불리는 이 시스템은 와트당 처리량 기준으로 GPGPU와 같은 동급 기술보다 훨씬 낮은 지연 시간을 달성하며, 경쟁력 있는 처리량을 제공합니다.

이 백서에서는 BW 시스템의 핵심인 BW NPU의 아키텍처와 마이크로아키텍처에 대해 자세히 설명합니다. 현재 형태에서 BW NPU는 FPGA에 합성된 DNN에 최적화된 '소프트 프로세서'입니다. FPGA보다 낮은 클럭 속도와 더 높은 면적 오버헤드에도 불구하고 BW NPU는 대규모 RNN 벤치마크에서 배치 없이 35테라플롭을 유지하면서 실시간 AI에 대한 기록적인 성능을 달성합니다. 그러나 BW NPU가 개별 DNN 요청에 대해 낮은 지연 시간을 달성하기 위해 사용하는 기술 중 하나만 재구성 가능한 로직에 연결되며, 나머지는 클럭 속도가 더 높지만 유연성이 떨어지는 "하드 NPU"에 적용될 수 있습니다.

BW 시스템과 NPU의 주요 측면은 다음과 같습니다:

- **아키텍처:** BW NPU는 행렬-벡터 및 벡터-벡터 연산으로 구성된 단일 스레드 SIMD ISA를 구현하며, 이는 멀티스레드(예: GPU) 또는 세분화된 MIMD(예: 그래프코어[3])를 사용하는 대부분의 최신 가속기와 대조됩니다. 대규모 병렬 가속기에 단일 스레드 모델을 사용하는 것이 직관적이지 않은 것처럼 보일 수 있지만, BW NPU는 개별 요청에서 높은 활용도를 제공하기 위해 충분한 SIMD 및 파이프라인 병렬성을 추출할 수 있습니다. 또한 단일 스레드 모델은 컴파일러나 프로그래머에 의존하여 병렬성을 추출하는 대신 BW NPU에서 실행되는 소프트웨어가 가속화된 하위 그래프에서 연산자를 선형화하기 때문에 소프트웨어에 대한 부담도 줄여줍니다.
- **메모리 시스템:** 지연 시간을 최소화하기 위해 BW 시스템은 일반적으로 저전력으로 초당 테라바이트의 대역폭을 제공하는 분산형 온칩 SRAM 메모리에 DNN 모델 가중치를 고정합니다(다른 최신 NPU[4]에서와 같이). 또한, 현재 퓨어스토리지의 FPGA에는 로컬 DRAM이 있어 CNN과 같이 계산 집약적인(따라서 대역폭 제약이 적은) 모델에 사용하거나 제한된 수의 FPGA에서 대규모 모델을 실행하는 데 사용할 수 있습니다.
- **마이크로아키텍처:** BW NPU 마이크로아키텍처는 병렬성을 추출하기 위해 세 가지 기술을 사용합니다. 첫 번째는 계층적 디코드 및 디스패치(HDD)로, 컴파운드 SIMD 연산을 더 많은 수의 프리미티브 연산으로 연속적으로 확장하여 기능 단위로 확장하는 방식입니다. 예를 들어, 단일 컴파운드 매트릭스-벡터 명령어는 결국 10,000개 이상의 프리미티브 연산을 생성하게 됩니다. 둘째, BW NPU는 콤파운드 연산을 고정된 기본 벡터 크기(예: 벡터당 100~400개 연산)의 연산으로 분할하는 벡터 수준 병렬 처리(VLP)를 사용하며, 이러한 벡터 연산을 통해 연산 단위로 확장되는 프리미티브가 형성됩니다(ILP 머신의 스칼라 명령어와 유사). 셋째, BW 툴체인은 이러한 병렬화된 벡터 연산을 벡터 기능 단위의 평평한 1차원 네트워크에 매핑하고 데이터 흐름 방식으로 연결하여 벡터가 한 기능 단위에서 다른 기능 단위로 직접 흐르도록 함으로써 파이프라인 거품을 최소화합니다. 행렬-벡터 곱셈 또는 컨볼루션과 같은 고수준 연산은 평평한 벡터 단위 공간에 중첩됩니다.
- **시스템:** BW 시스템에서 BW NPU를 호스팅하는 FPGA 칩은 데이터센터 네트워크에 직접 연결되므로 오버헤드가 거의 없이 원격 서버로부터 DNN 추론 요청 스트림을 수신할 수 있습니다. 여러 구성 요소가 있는 모델은 여러 개의 FPGA에 걸쳐 분할할 수 있습니다. 현재 BW 시스템은 프로덕션 환경에서 다중 FPGA 모델을 실행하고 있지만, 이 백서에서는 개별 FPGA에 완전히 적합한 인기 모델의 단일 노드 평가에 초점을 맞추고 있습니다.
- **합성 전문화:** BW NPU는 높은 수준의 단일 스레드 추상화를 제공하므로, 기반이 되는

마이크로아키텍처는 매우 다양할 수 있습니다. 이러한 유연성 덕분에 BW NPU는 여러 세대의 FPGA 또는 ASIC 기술과 같은 다양한 구현에 매핑될 수 있습니다. 그러나 이러한 유연성 덕분에 마이크로아키텍처는 특정 기술 유형 및 세대 내에서 달라질 수 있습니다. BW NPU는 특정 DNN 모델 또는 모델 클래스에 대해 마이크로아키텍처 리소스를 최적화할 수 있는 네 가지 합성 시간 파라미터를 허용합니다. 이러한 파라미터는 데이터 유형(정밀도), 기본 벡터 크기, 데이터 레인 수, 행렬-벡터 타일 엔진 크기입니다. 이러한 매개변수화를 통해 다양한 모델 유형에 걸쳐 일반적인 지원을 구현해야 하는 강화된 구현과 달리, 여러 모델 클래스 각각에 대해 더 간결한 마이크로아키텍처를 구현할 수 있습니다.

Stratix 10 FPGA에서 실행되는 다양한 중대형 RNN 벤치마크에서 BW NPU는 4ms 미만의 레이턴시에서 11~35.9테라플롭스 범위의 처리량을 달성합니다. 이러한 결과는 동등한 모델 점수 정확도를 제공하는 저정밀 부동소수점 형식을 사용합니다. 더 중요한 것은 이러한 결과가 일괄 처리 없이 달성되었기 때문에 시스템이 실시간으로 요청을 개별적으로 처리할 수 있다는 점입니다. BW 시스템은 DNN 추론의 경우 시스템이 높은 처리량을 달성하기 위해 낮은 지연 시간을 희생할 필요가 없음을 보여줍니다.

II. 배경

BW 시스템은 실시간 AI 요구 사항이 있는 프로덕션 서비스를 실행하는 기존 하이퍼스케일 클라우드 인프라와 통합됩니다[5]. 이 섹션에서는 대상 데이터센터 아키텍처의 주요 특징과 DNN 서비스 스택의 주요 계층에 대한 배경 지식을 제공합니다.

A. 하이퍼스케일 데이터센터 가속 아키텍처

그림 1은 하이퍼스케일 데이터센터의 구성 요소를 보여줍니다. 여기서 설명하는 가속 아키텍처는 전 세계적으로 대규모로 생산되고 있습니다. 모든 표준 듀얼 소켓 서버에는 FPGA 및/또는 ASIC이 포함된 하나 이상의 PCIe 연결 가속기 카드가 호스팅됩니다. 가속기 카드는 데이터센터 네트워크에 직접 액세스할 수 있으며 서버 NIC와 TOR(Top-of-Rack) 스위치 사이에 인라인으로 배치됩니다. 가속기는 온칩 RDMA와 유사한 무손실 프로토콜을 사용하여 동일한 데이터센터에 위치한 수십만 대의 서버와 짧은 지연 시간으로 직접 지점 간 통신을 할 수 있습니다. 그림 1에 표시된 데이터센터 아키텍처를 사용하면 가속기를 논리적으로 분리하여 루프에 소프트웨어 없이 하드웨어 마이크로서비스 인스턴스로 풀링할 수 있습니다. 초기화되고 분산 리소스 관리자에 등록되면 지정된 하드웨어 마이크로서비스가 시스템의 구독 CPU에 게시되고 IP 주소를 통해 직접 액세스됩니다.

그림 1의 시스템은 클라우드 규모에서 가속기를 관리하고 아키텍처를 구축하는 방식에 근본적인 영향을 미칩니다. 특정 가속 시나리오에 할당된 CPU 및 FPGA 리소스를 독립적으로 확장할 수 있으므로 용량이 좌초되는 것을 방지하고 전반적인 데이터센터 활용도를 개선할 수 있습니다. 대규모,

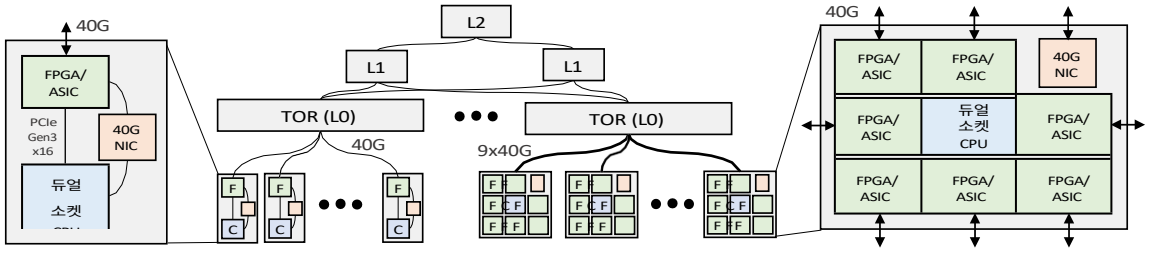


그림 1. 클라우드 규모의 BW 시스템. 왼쪽에서 오른쪽으로, 범프인더와이어 가속기가 있는 서버, 하이퍼스케일 데이터센터 네트워크에 직접 연결된 가속기, 가속기 어플라이언스.

분할 가능한 문제를 여러 가속기에 걸쳐 공간적으로 분산할 수 있습니다. 예를 들어, 양방향 RNN을 두 개의 독립적인 FPGA에 분할하여 서버가 포워드 및 리버스 RNN FPGA를 개별적으로 호출하고 출력을 연결했습니다.

B. DNN 가속 플랫폼

DNN 가속 플랫폼은 스케일 인프라에서 실행되며 세 가지 구성 요소로 이루어져 있습니다: (1) 사전 학습된 DNN 모델 체크포인트를 소프트웨어 및 가속기 실행 파일로 변환하는 툴 플로우, (2) CPU와 분산 하드웨어 마이크로서비스 간에 모델 실행을 오케스트레이션하는 연합 런타임, (3) FPGA에 인스턴스화된 프로그래밍 가능한 BW NPU입니다.

툴플로우의 초기 단계에서는 사전 학습된 DNN 모델을 DNN 프레임워크(예: 텐서플로우[6])에서 BW의 그래프 중간 표현(GIR)으로 내보냅니다. GIR은 목표 가속기 수와 가속기당 사용 가능한 온칩 메모리 등 백엔드 시스템의 목표 제약 조건에 따라 일련의 최적화 및 변환을 거칩니다. 지연 시간에 민감한 실시간 시나리오에서는 툴플로우가 단일 FPGA의 용량을 초과하는 큰 그래프를 가속기의 온칩 메모리에 개별적으로 파라미터를 고정할 수 있는 하위 그래프로 분할하는 경우가 많습니다. 이러한 파티셔닝은 종종 대규모 RNN 및 MLP의 배포를 방해하는 메모리 용량/대역폭 트레이드 오프 문제를 방지합니다. BW NPU에서 지원되지 않거나 수익성 있게 가속화할 수 없는 연산은 CPU 코어에서 실행할 수 있도록 하위 그래프로 그룹화됩니다.

파티셔닝이 완료되면 FPGA 및 CPU 하위 그래프가 각각 BW NPU 및 CPU ISA 바이너리로 컴파일됩니다. (BW NPU ISA는 섹션 IV-C에 설명되어 있습니다.) 생성된 백엔드 실행 파일은 패키징되어 원격 하드웨어 마이크로서비스 호출을 통해 시작된 CPU 서브 그래프와 가속기 서브 그래프를 모두 실행하는 클라우드의 페더레이션 CPU 런타임에 배포됩니다.

III. 지연 시간 인식 설계를 위한 임계 경로 방법론

이 섹션에서는 BW NPU 아키텍처를 자세히 설명하기 전에 단일 요청의 지연 시간에 최적화된 실시간 NPU의 설계 및 평가를 엄격하게 안내하는 임계 경로 방법론에 대해 설명합니다. 초당 작업 수 및 에너지 효율성과 같은 독립형 메트릭은 널리 사용되는 최적화 목표이지만, 이러한 메트릭은 다음을 포착하지 못합니다.

일괄 처리의 효과는 인위적으로 사용률을 높이는 동시에 지연 시간을 증가시킬 수 있습니다.

모델	차원	Ops	주기			데이터
			UDM	SDM	BW NPU	
LSTM	2000x2000	64M	19	352	718	32MB
GRU	2800x2800	94M	31	520	662	47MB
CNN	크기 :28x28x128 K:128x3x3	231M	13	1204	1326	247KB
CNN	크기 :56x56x64 K:256x1x1	103M	13	549	646	200KB

표 I
LSTM, GRU 및 CNN의 임계 경로 분석.

이러한 격차를 해소하기 위해, 저희는 임계 경로 분석에 기반한 지연 시간 중심 메트릭을 추가로 도입했습니다. (1) 무한 리소스가 있는 무제한 데이터 흐름 머신(UDM)에서 모델을 서비스하는 데 필요한 사이클 수, (2) 타르 겟 가속기 구현과 동일한 수의 기능 유닛을 공유하는 구조적으로 제한된 데이터 흐름 머신(SDM)에서 서비스하는 데 필요한 사이클 수입니다. 임계 경로를 모델링할 때는 기능 단위 레이턴시만 UDM과 SDM에서 계산됩니다. 이러한 메트릭은 이상적인 구현과의 지연 시간 격차를 특성화하여 강력한 NPU 평가를 가능하게 합니다. UDM은 단일 DNN 요청의 사용 가능한 모든 병렬 처리를 포착하는 하한 지연 시간을 반영하는 반면, SDM은 현실적인 리소스 제약 조건에서 가능한 가장 낮은 지연 시간을 반영하고 구현이 높은 마이크로 아키텍처 효율성으로 단일 DNN 요청의 사용 가능한 병렬 처리를 얼마나 잘 활용하는지를 평가합니다.

LSTM 예시. 그림 2는 최신 음성 및 텍스트 생성 DNN 모델에서 일반적으로 사용되는 장단기 메모리(LSTM) 블록[7]의 데이터 흐름에 적용된 임계 경로 분석을 보여줍니다. 표 I은 단일 2000차원 LSTM 평가에서 UDM 및 SDM 지연 시간을 BW NPU와 비교한 것입니다(자세한 내용은 섹션 VII 참조). LSTM은 시간 단계당 6400만 개의 연산이 필요하며 이상적인 UDM에서는 19사이클에서 실행할 수 있습니다. 실제 BW NPU에서와 같이 96,000개의 MAC(멀티플라이 누산기)으로 제한되는 보다 현실적인 SDM은 352사이클에서 모델을 제공합니다. SDM과 UDM 간의 18배의 격차는 더 많은 리소스를 사용하여 더 많은 성능 향상을 얻을 수 있음을 시사합니다. 실제 BW NPU 구현은 이상적인 SDM의 2배 이내인 718사이클로 LSTM에 서비스를 제공하므로 마이크로 아키텍처 효율성이 우수함을 나타냅니다. 참고로

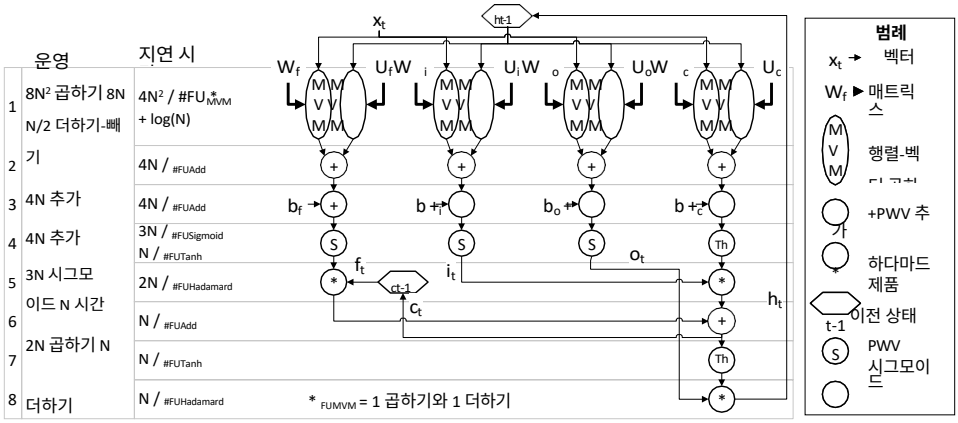


그림 2. LSTM 임계 경로 분석. 연산 수와 지연 시간은 LSTM 차원(N)과 기능 단위 수(#FU)의 함수로 표시됩니다.

표 I은 또한 LSTM을 ResNet-50의 대표적인 2-D CNN 레이어와 비교합니다[8]. 전반적으로 LSTM은 2-D CNN보다 가용 병렬 처리 능력이 낮고 데이터 요구 사항이 높기 때문에 가속하기가 훨씬 더 어렵습니다.

데이터 및 구조적 위험, 파이프라인 정체, 제한된 메모리 대역폭과 같은 마이크로 아키텍처의 비효율성으로 인해 NPU 구현이 이상적인 SDM 지연 시간에 접근하지 못합니다. 이 백서의 나머지 부분에서는 높은 마이크로 아키텍처 효율성으로 단일 DNN 요청에서 사용 가능한 병렬 처리의 대부분을 추출하는 것이 가능하다는 것을 보여줍니다.

IV. 아키텍처

A. 개요

BW NPU 아키텍처의 목표는 다음과 같습니다: (1) 프로그래머와 컴파일러가 쉽게 타겟팅할 수 있는 간단한 프로그래밍 추상화를 제공하고, (2) 기본 마이크로 아키텍처가 사용 가능한 병렬 처리를 효율적으로 활용할 수 있도록 잠재적으로 큰 DNN 연산에 대한 충분한 정보를 인코딩하고, (3) RNN, MLP 및 CNN을 아우르는 광범위한 DNN 모델을 처리할 수 있는 충분한 유연성을 지원하는 것입니다.

이러한 목표를 달성하기 위해 BW NPU는 1차원 및 2차원 고정 크기 "네이티브" 벡터와 매트릭스를 일류 데이터 유형으로 사용하는 복합 연산으로 구성된 단일 스레드 SIMD ISA를 채택합니다. 벡터 기반 처리는 잘 정립된 패러다임이지만, BW NPU ISA는 저지연 DNN 서비스의 요구사항에 맞춘 고유한 기능을 도입했습니다. 대규모 DNN 모델의 하위 그래프는 그래프 에지 간의 명시적 통신을 효율적으로 캡처하는 원자 명령어 체인(명령어 간 명명된 스토리지 없이)을 통해 인코딩할 수 있어 소프트웨어 개발을 간소화하고 하드웨어의 복잡성을 줄일 수 있습니다(섹션 V 참조). 또한 BW NPU 아키텍처는 지연 시간이 짧은 DNN 서비스를 위해 최적화된 특수 명령어, 데이터 유형, 메모리 추상화를 제공합니다.

BW NPU 행렬/벡터 데이터 경로는 코프로세서로 구현되며, 기존 스칼라 코어를 사용하여 명령어 대기열을 통해 데이터 경로에 BW NPU 명령어를 발행합니다. 스칼라 코어는 중요한 요구 사항인 동적 입력 종속 제어 흐름을 포함하여 BW NPU의 제어 흐름을 제공합니다.

가변 길이의 시간 간격이 있는 단일 배치 RNN과 같은 특정 모델의 경우.

B. 행렬-벡터 곱셈

기존의 기본 선형대수 루틴(BLAS)[9]에서는 BLAS 선형대수 루틴을 수행할 수 있는 세 가지 표준 레벨이 있습니다: (1) 벡터 전용 연산으로 구성된 레벨 1(L1), (2) 행렬-벡터 연산으로 구성된 레벨 2(L2), (3) 행렬-행렬 연산으로 구성된 레벨 3(L3)입니다. L1은 L1-L3을 구현하는 데 사용할 수 있으며, 마찬가지로 L2는 L2-L3을 구현하는 데 사용할 수 있습니다.

상업적으로 배포된 클라우드 NPU인 TPU1[10]은 수축기 2차원 행렬-행렬 곱셈 어레이를 사용하여 L3에 가장 가깝게 부합합니다. L3 행렬-행렬 곱셈은 높은 데이터 재사용률($O(N)$)과 대역폭 요구 사항 감소로 인해 하드웨어 구현에서 확장하기가 가장 간단하지만 DNN 모델에서 고밀도 레이어가 필요합니다.

를 일괄 처리하는 것이 효율적입니다. 즉, L3 행렬-행렬 곱셈은 배치되지 않은 실행에 상대적으로 적합하지 않습니다. 반면에 L2 행렬-벡터 곱셈은 단일 요청 서빙, 특히 메모리 집약적인 RNN과 고밀도 레이어가 많은 MLP에서 매우 바람직합니다. 따라서 BW NPU 아키텍처는 핵심 연산으로 행렬-벡터 곱셈에 중점을 둡니다. 많은 고차원 모델(예: 고밀도 행렬-행렬 곱셈, 1D 또는 2D CNN 등)은 행렬-벡터 곱셈으로 효율적으로 선형화할 수 있는 반면, 그 반대의 경우는 더 어렵습니다. 최적화된 배치 1 서비스를 제공하고 다양한 모델을 지원하기 위해 명시적 행렬-행렬 곱셈 및/또는 컨볼루션 커널과 같은 고차원 프리미티브의 생성은 특별히 피했습니다. 하드웨어에서 저차원 프리미티브를 최적화하는 것은 더 어렵지만, 섹션 7에서 설명한 대로 높은 수준의 병렬 처리까지 달성하고 확장할 수 있다는 사실을 발견했습니다, 유연성을 확보할 수 있어 매우 유용했습니다.

C. BW NPU 명령어 세트 아키텍처

BW NPU 명령어 세트는 시간이 지남에 따라 LSTM, GRU, 1D(텍스트) CNN, 2D(이미지) CNN을 아우르는 프로덕션 DNN 모델의 요구 사항을 수용하도록 발전했습니다,

이름	설명	IN	피연산자 1	피연산자 2	OUT
v rd	벡터 읽기	-	MemID	메모리 색인	V
v wr	벡터 쓰기	V	MemID	메모리 색인	-
m rd	매트릭스 읽기	-	MemID(NetQ 또는 DRAM만 해당)	메모리 색인	M
m wr	매트릭스 쓰기	M	MemID(MatrixRf 또는 DRAM만 해당)	메모리 색인	-
mv mul	행렬-벡터 곱하기	V	MatrixRf 인덱스	-	V
vv 추가	PWV 추가	V	AddSubVrf 인덱스	-	V
vv a sub b	PWV 빼기, IN은 마이너스입니다.	V	AddSubVrf 인덱스	-	V
vv b sub a	PWV 빼기, IN은 서브트래헨드	V	AddSubVrf 인덱스	-	V
vv 최대	최대 PWV	V	AddSubVrf 인덱스	-	V
vv mul	하다마드 제품	V	MultiplyVrf 인덱스	-	V
v relu	PWV ReLU	V	-	-	V
v 기호	PWV 시그모이드	V	-	-	V
v tanh	PWV 쌍곡선 탄젠트	V	-	-	V
s wr	스칼라 제어 레지스터 쓰기	-	스칼라 레그 인덱스	스칼라 값	-
엔드 체인	명령 체인 종료	-	-	-	-

PWV = 포인트 단위 벡터 연산. IN = 암시적 입력(V: 벡터, M: 행렬, -: 없음). OUT = 암시적 출력.

표 II
단일 스레드 BW NPU ISA는 DNN 모델을 타겟팅하기 위한 작고 간단한 추상화를 제공합니다.

단어/문자 임베딩, 고밀도 MLP를 지원합니다. 표 II에서는 자주 사용되는 BW NPU 명령어 샘플을 제공하며, 아래에서 더 자세히 설명합니다.

데이터 유형, 스토리지, I/O. BW NPU에서 모든 명령은 N 길이의 1D 벡터 또는 $N \times N$ 2D 행렬에서 작동합니다. 벡터와 행렬은 별도의 데이터 유형으로 취급되며 독립적인 레지스터 파일에 저장됩니다. N은 주어진 BW NPU 구현에서 고정된 값입니다. 이상적인 벡터 크기는 대상 모델 세트에 따라 달라지며, 너무 큰 벡터는 비효율적인 패딩이 필요하고, 너무 작은 벡터는 오버헤드 제어를 증가시킵니다. 섹션 VI에서는 FPGA 기반 BW NPU의 맥락에서 네이티브 차원의 합성 특수화를 사용하여 하드웨어 인스턴스를 모델에 맞게 조정하는 방법에 대해 자세히 설명합니다.

벡터 및 행렬 읽기 및 쓰기 연산(v rd, v wr, m rd, m wr)은 첫 번째 피연산자를 사용하여 특정 레지스터 파일, DRAM 또는 네트워크 I/O 대기열일 수 있는 메모리 대상을 선택합니다. 두 번째 피연산자는 네트워크 I/O의 경우를 제외하고 메모리 인덱스를 제공합니다. 다른 인스트럭션의 경우, 액세스되는 메모리 구조(있는 경우)는 BW NPU 메모리가 특정 기능 유닛에 긴밀하게 결합되어 있기 때문에 옵코드에 의해 암시적으로 식별됩니다. 이러한 명령어의 경우 인덱스 피연산자만 필요합니다.

매트릭스 스토리지는 상수 값(모델 가중치)에 특화되어 있습니다. 행렬은 네트워크(초기화를 위해) 또는 DRAM에서만 읽을 수 있으며, 행렬 레지스터 파일(MRF) 또는 DRAM에만 쓸 수 있습니다. MRF는 행렬-벡터 곱하기(mv mul)의 암시적 피연산자로만 읽습니다.

명령어 체인. BW NPU ISA의 기본 기능은 종속 명령어 시퀀스가 한 연산에서 다음 연산으로 직접 값을 전달하는 명시적 명령어 체인입니다. 명시적 체이닝을 통해 마이크로아키텍처는 복잡한 하드웨어 종속성 검사나 멀티포트 레지스터 파일 없이도 상당한 파이프라인 병렬성을 활용할 수 있습니다. 섹션 5에서 자세히 설명하겠지만, 이 체인 지원 파이프라인 병렬 처리를 통해 마이크로 아키텍처는 짧은 지연 시간으로 중요한 직렬 종속성을 해결할 수 있습니다.

표 II의 IN 및 OUT 열은 암시적인

(체인) 입력 및 출력 피연산자가 각 명령어에 대해 필요합니다. 체인은 체인 입력 없이 체인 출력을 생성하는 유일한 명령어인 v rd 또는 m rd로 시작해야 합니다. 벡터 체인은 벡터를 소비하고 생성하는 연산을 얼마든지 포함할 수 있지만, 연산의 길이와 순서는 마이크로 아키텍처 구현에 의해 제약을 받습니다. 벡터 체인은 이전 명령의 벡터 출력을 가라앉히는 v wr 연산으로 종료됩니다. 체인 의미론의 특수한 경우로, 벡터 체인은 최종 벡터 값을 여러 대상에 멀티캐스트하는 여러 개의 v wr 연산으로 끝날 수 있습니다.

매트릭스 체인은 항상 정확히 두 개의 명령어, 즉 m rd와 m wr로 구성되며 네트워크에서 매트릭스를 초기화하고 MRF와 DRAM 간에 매트릭스를 이동하는 역할만 수행합니다.

메가-SIMD 실행. BW NPU는 s wr을 사용하여 스칼라 제어 레지스터를 설정함으로써 기본 차원의 배수에 대한 연산을 가능하게 합니다. 예를 들어 행=4, 열=5로 설정하면 후속 mv mul 연산이 다음과 같이 처리됩니다.
20개의 입력 벡터(*5N* 값)를 소비하고 4개의 출력 벡터(*4N* 값)를 생성하는 타일형 $4N \times 5N$ 행렬로 구성된 20개의 연속된 MRF 항목. 체인의 다른 명령어들도 적절하게 스케일링됩니다(예: 피드하는 v rd 연산).
의 경우, mv mul은 충분한 입력을 제공하기 위해 5개의 연속된 VRF 항목을 읽고, 체인 끝에 있는 v wr은 4개의 연속된 VRF 항목을 씁니다. 이 기능은 모델을 매개변수화하는 데 많이 사용되어 왔으며, 명령어 병목 현상을 줄일 수 있다는 추가적인 이점이 있습니다. 섹션 VII에서 평가한 가장 큰 GRU에서는 단일 명령어가 7백만 개 이상의 연산을 전송하도록 구성할 수 있습니다.

성능 잠재력에도 불구하고 BW NPU ISA는 간결하고 비교적 읽기 쉬운 프로그래밍 모델을 제공합니다. 아래에 요약된 것처럼 완전히 파라미터화되고 성능이 조정된 LSTM은 100줄 미만의 코드로 표현할 수 있습니다:

```
void LSTM(int steps) {
  for ( int t = 0; t < steps; t++) {
    v rd(NetQ);
    v wr( InitialVrf , ivrf xt ) ;
    // xWf = xt * Wf + bf
```

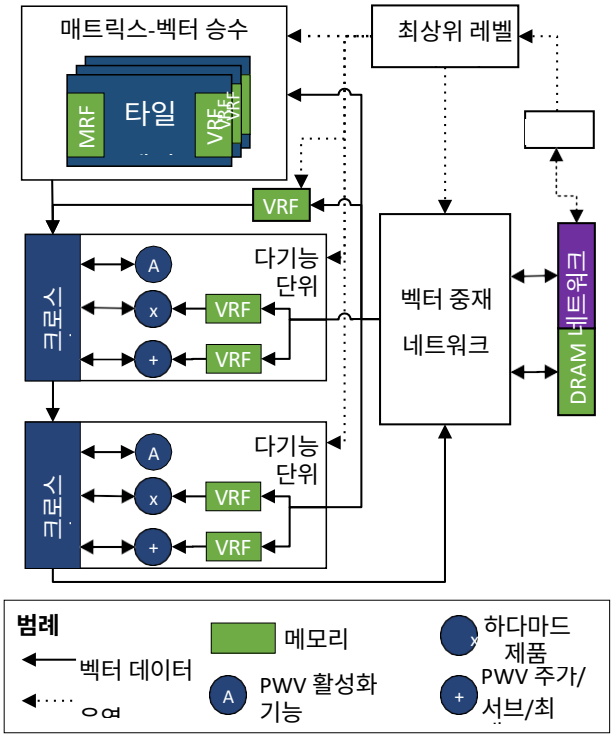


그림 3. 마이크로아키텍처 개요.

```
v_rd( InitialVrf , ivrf xt ) ;
mv_mul(mrf Wf);
VV_ADD(ASVRF BF) ;
v_wr(AddSubVrf, asvrf xWf);
// xWi = xt * Wi + bi ...
// xWf = xt * Wo + bo ...
// xWc = xt * Wc + bc ...
// f 게이트 -> 곱하기 c prev v_rd(
InitialVrf , ivrf h_prev ) ; mv
mul(mrf Uf);
vv_add(asvrf xWf);
v_sigm(); // ft
VV_MUL(MUL_VRF C PREV);
v_wr(AddSubVrf, asvrf ft mod);
// i 게이트 ...
// O 게이트 ...
// c gate -> store ct and c prev v
rd( InitialVrf , ivrf h prev ) ; mv
mul(mrf Uc);
vv_add(asvrf xWc);
v_tanh() ;
VV_MUL(MUL_VRF IT) ;
VV_ADD(ASVRF FT MOD); // CT
v_wr(MultiplyVrf, mulvrf c_prev); v
wr( InitialVrf , ivrf ct ) ;
// produce ht , 저장 및 네트워크에 전송
v_rd( InitialVrf , ivrf ct ) ; v
tanh() ;
VV_MUL(MUL_VRF OT); // HT
v_wr( InitialVrf , ivrf h_prev ) ; v
wr(NetQ);
  }
}
```

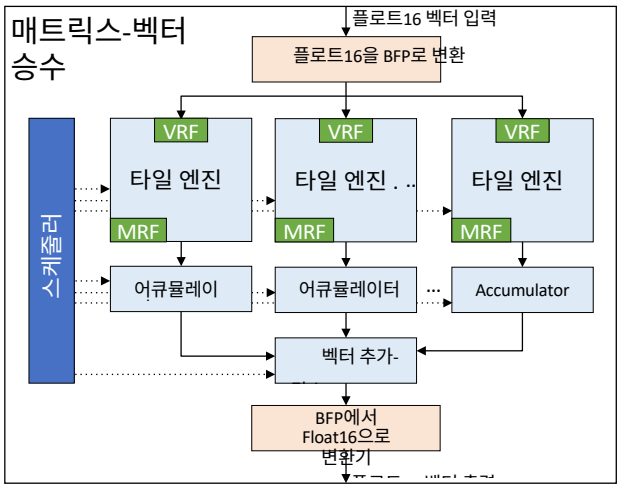


그림 4. 행렬-벡터 승수 개요.

V. 마이크로 아키텍처

서론과 섹션 III에서 설명한 바와 같이 BW NPU 마이크로아키텍처의 목표는 높은 하드웨어 효율로 단일 DNN 요청의 벡터 수준 병렬 처리(VLP)를 최대로 활용하는 것입니다. 실제로 위험 요소, 디코딩 비효율성, 모델의 고유한 직렬 데이터 종속성(예: LSTM 시간 단계 간)으로 인한 파이프라인 중단으로 인해 단일 요청 내에서 활용 가능한 VLP가 제한됩니다.

그림 3은 BW NPU 마이크로아키텍처의 개략적인 모습을 보여줍니다. 주요 목표는 명령어 체인(섹션 IV에서 설명)을 함수 유닛을 통해 흐르는 벡터 요소의 연속적이고 중단 없는 스트림에 매핑하고 실행하는 것입니다. 함수 유닛은 명령어 체인 구조를 반영하는 선형 파이프라인을 형성하며, 맨 앞에는 매트릭스-벡터 승수(MVM)가 있습니다. 벡터 중재 네트워크는 메모리 구성 요소인 파이프라인 레지스터 파일(MRF 및 VRF), DRAM, 네트워크 I/O 큐 간의 데이터 이동을 관리합니다. 최상위 스케줄러는 스칼라 제어 프로세서(현재 구현에서는 Nios 소프트 프로세서)로부터 수신한 BW ISA 명령어 체인을 기반으로 기능 유닛과 벡터 중재 네트워크에 대한 제어 신호를 구성합니다.

이 섹션의 나머지 부분에서는 마이크로 아키텍처의 세 가지 주요 측면, 즉 MVM, 벡터 다기능 유닛 및 이를 구동하는 계층적 제어 구조에 대해 자세히 설명합니다.

A. 매트릭스-벡터 승수

매트릭스-벡터 멀티플라이어(MVM)는 BW NPU의 주축입니다. MVM은 최대 100,000개의 독립적인 승수와 누산기로 구성된 도트 프로덕트 단위의 네트워크에서 확장됩니다. 매트릭스-매트릭스 멀티플라이어와 달리 MVM은 메모리 대역폭이 제한됩니다. 이러한 병목 현상을 완화하기 위해 모든 도트 프로덕트 유닛에 대한 각 입력에는 최대 처리량으로 유닛에 공급할 수 있는 전용 메모리 포트가 필요합니다.

MVM의 계층적 보기는 그림 4에 나와 있습니다. 최상위 레벨에서 MVM은 일련의 매트릭스 벡터 타일 엔진을 인스턴스화하며, 각 엔진은 네이티브 크기의 MVM을 구현합니다. 차례로 각 타일 엔진은 일련의 점으로 구성된

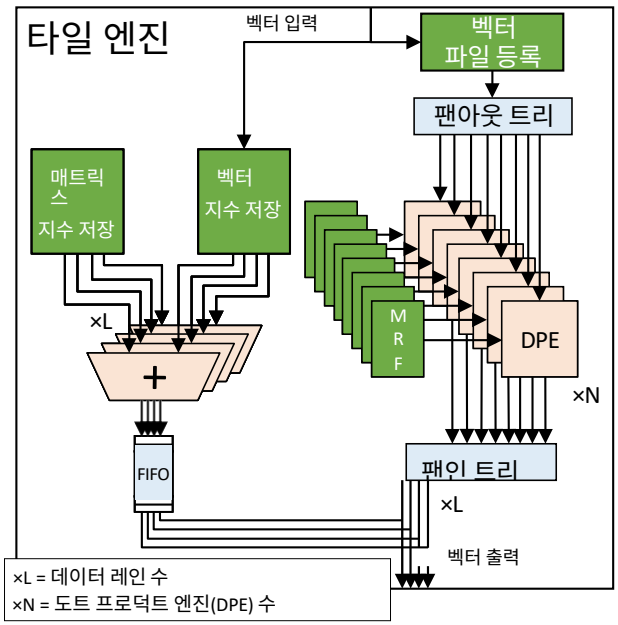


그림 5. 매트릭스-벡터 타일 엔진 마이크로아키텍처.

곱셈 엔진이 있습니다. 이름에서 알 수 있듯이 각 도트 곱 엔진은 입력 벡터에 매트릭스 타일의 기본 행 하나를 곱하는 것에 해당하는 도트 곱 계산을 담당합니다. 도트 곱셈 엔진은 누적 트리에 공급되는 병렬 곱셈기 레인으로 구성됩니다. 이러한 레인은 매트릭스 타일의 행 열 내에서 병렬성을 제공합니다. 이를 종합하면 MVM은 MVM 간, MVM 타일링, 타일의 행 간, 행의 열 내 등 4차원의 병렬 처리를 활용합니다. MVM의 총 처리량(사이클당 FLOPs)은 2 × # 타일 엔진 × # DPE × # 레인으로 표현할 수 있습니다.

L2 MVM 대역폭 유지. 각 승수는 사이클당 하나의 벡터 요소와 하나의 행렬 요소를 수신합니다. 행렬 요소는 해당 승수에 인접한 매트릭스 레지스터 파일(MRF)의 전용 포트를 통해 전달됩니다. MRF는 타일 엔진 전반의 네이티브 타일에 의해 뱅크됩니다. 각 뱅크는 행별로 하위 뱅크화되어 타일 엔진의 첫 번째 하위 뱅크(첫 번째 도트 프로덕트 엔진과 연결됨)에는 MRF 뱅크에 있는 모든 매트릭스 타일의 첫 번째 행이 포함되도록 합니다. 각 행의 요소는 SRAM에서 인터리브되어 각 SRAM 읽기 포트가 승수에 직접 연결될 수 있습니다. 이 조직은 로컬 컴퓨팅 타일로 MRF 읽기 포트 수를 확장합니다. MRF 항목은 DRAM 또는 네트워크 입력 대기열에서만 쓸 수 있으므로 쓰기 대역폭 요구 사항이 훨씬 낮습니다.

MVM 성능은 MRF 대역폭에 따라 달라지므로 높은 활용도를 달성하려면 필요한 행렬 피연산자가 사용 가능한 온칩 SRAM에 맞아야 합니다. 섹션 II에서 설명한 바와 같이 BW NPU 아키텍처의 전체 시스템 설계는 단일 가속기의 온칩 메모리가 모두 소진된 경우 여러 가속기에 걸쳐 대규모 멀티 컴포넌트 RNN/MLP 모델을 분할할 수 있도록 지원합니다. CNN은 컴퓨팅 집약적이기 때문에 DRAM에서 새로운 피연산자를 전송할 때 현재 MRF 콘텐츠에 대한 연산과 겹칠 수 있습니다.

B. 복합기

MVM의 출력은 일련의 벡터 다기능 유닛(MFU)을 통해 라우팅됩니다. MFU는 곱셈과 덧셈과 같은 벡터-벡터 연산은 물론 ReLU, 시그모이드, 탄과 같은 단항 벡터 활성화 함수를 지원합니다. 이러한 기능은 ISA에서 표 II의 vv * 및 v * 연산으로 노출됩니다. 더하기/빼기 및 곱하기 함수 유닛과 연결된 전용 벡터 레지스터 파일(VRF)은 이러한 연산에 필요한 보조 피연산자를 제공합니다. 각 MFU에는 비차단 크로스바를 통해 MFU의 입력 및 출력 포트에 연결된 벡터 기능 유닛 세트(현재 설계에서는 3개)가 포함되어 있습니다. 크로스바는 내부 기능 유닛의 시퀀스 또는 하위 시퀀스(완전한 바이패스 포함)를 통해 MFU의 입력을 출력으로 라우팅하도록 현재 인스트럭션 체인에 따라 구성됩니다. 구성이 완료되면 벡터 시퀀스를 MFU를 통해 파이프라인으로 연결할 수 있습니다. 여러 MFU를 연결하여 더 긴 벡터 연산 시퀀스를 지원할 수 있습니다. 두 개의 MFU가는 대부분의 인스트럭션 체인을 지원하기에 충분합니다.

C. 계층적 디코딩 및 디스패치(HDD)

BW NPU는 그림 3과 같이 기존 스칼라 제어 프로세서(자체 스칼라 명령어 세트 포함)를 사용하여 최상위 스케줄러에 비동기적으로 BW NPU 명령을 동적으로 발행합니다. 현재 유니티의 FPGA 구현에서 제어 프로세서는 소프트웨어 매크로를 통해 BW NPU 명령어를 생성하기 위한 맞춤형 C 라이브러리와 결합된 기성품 Nios II/f 소프트 프로세서로 구현됩니다.

최상위 스케줄러는 공간적으로 분산된 많은 컴퓨팅 리소스의 작동을 제어하기 위해 각 명령어 체인을 수천 개의 기본 연산으로 디코딩해야 합니다. 그림 6에 표시된 계층적 디코드 및 디스패치(HDD) 로직은 복합 연산을 수천 개의 컴퓨팅 유닛과 수십 개의 레지스터 파일 및 스위치를 관리하는 분산 제어 신호로 확장합니다. 최상위 스케줄러는 6개의 디코더와 4개의 세컨드 레벨 스케줄러로 디스패치하고, 이 스케줄러는 다시 41개의 디코더로 디스패치합니다. 이 방식은 각 단계의 버퍼링과 결합되어 전체 컴퓨팅 파이프라인이 4클럭 사이클마다 평균적으로 하나의 복합 명령어를 Nios에서 전송하여 계속 실행되도록 합니다.

그림 6은 단일 명령어에서 가장 큰 기능 단위(MVM)가 어떻게 제어되는지 보여줍니다. Nios 프로세서가 N개의 정적 명령어를 T회 반복하여 최상위 스케줄러로 스트리밍하면서 디코딩 정보의 확장이 위에서 아래로 이루어집니다. 다음으로 최상위 스케줄러는 명령어의 MVM 관련 부분을 두 번째 레벨 스케줄러로 보내고, 이 스케줄러는 대상 행렬의 R 행과 C 열을 따라 연산을 확장합니다(섹션 IV-C에 설명된 행 및 열 제어 레지스터 사용). 이러한 MVM 스케줄은 E 매트릭스-벡터 타일 엔진에 매핑되며, 타일 엔진과 관련 벡터 레지스터 파일 및 누적 유닛, 모놀리식 더하기-줄이기 유닛에 대해 각각 E 디코더 세트로 연산을 전송합니다. 마지막으로, 이러한 디코더는 데이터 플레인으로 분산되는 제어 신호를 생성하며, 각 타일 엔진 디스패처는 수백 개의 도트 프로덕트 엔진으로 분산됩니다.

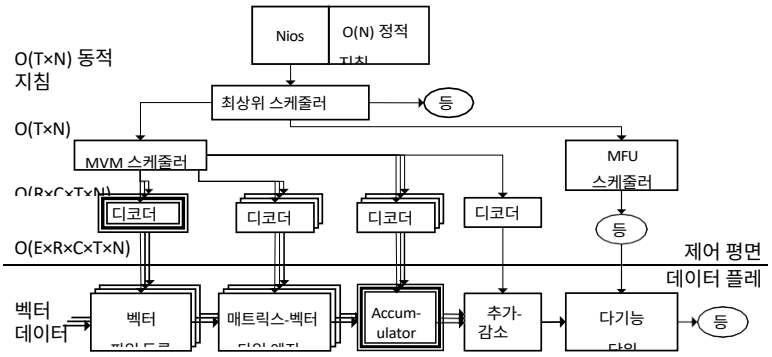


그림 6. 계층적 디코드 및 디스패치(HDD)를 매트릭스-벡터 멀티플라이어로 전송.

VI. 합성 전문화

BW 마이크로아키텍처는 효율성을 위해 특정 모델에 맞게 커스터마이징할 수 있는 완전히 파라미터화할 수 있는 프로세서 제품군으로 볼 수 있습니다. 하드 BW 프로세서는 높은 클럭 주파수에서 작동할 수 있지만 컴파일 타임에 파라미터를 선택해야 하는 반면, FPGA를 대상으로 하는 소프트 BW 프로세서는 특정 모델에 맞게 합성 특화할 수 있습니다.

데이터 경로 전문화. BW 아키텍처는 특정 모델에 마이크로 아키텍처 인스턴스를 특화하기 위해 사용할 수 있는 7가지 주요 파라미터를 노출합니다. (1) 기본 벡터 차원을 모델의 파라미터에 맞추면 모델 평가 시 패딩과 낭비를 최소화하는 경향이 있습니다,
(2) 레인 폭을 늘리면 행 내 수준 병렬 처리를 향상시키는 데 사용할 수 있고, (3) 행렬 곱하기 타일을 늘리면 대형 모델에서 하위 행렬 병렬 처리를 활용할 수 있습니다. 섹션 VII의 결과는 모델에 파라미터를 커스터마이징할 때의 시사점을 보여줍니다.

좁은 정밀도 데이터 유형. 광범위한 문헌(예: [11]-[14])에 따르면 심층 신경망은 데이터 유형과 희소성에서 압축성이 매우 높습니다. FPGA에서는 기본 벡터 수준에서 숫자 그룹 전체에 걸쳐 5비트 지수를 공유하는 좁은 정밀도 블록 부동 소수점 형식[15]을 사용합니다(예: 128개의 독립 부호와 맨티사당 단일 5비트 지수). 보조 연산(예: 점 단위 벡터 곱하기 또는 탄)은 하드웨어에서 여전히 float16으로 실행되므로 BFP로 인한 양자화 노이즈는 도트 곱에만 영향을 미칩니다. 유니티는 BFP의 변형을 사용하여 프로덕션 최신 DNN 모델과 대규모 ImageNet 모델(예: ResNet-50)에서 단 몇 번의 미세 조정을 통해 정확도에 미치는 영향이 미미한(기준선의 1~2% 이내) 2~5비트 수준으로 사마귀를 성공적으로 트리밍했습니다. 유니티의 변형 BFP를 사용하면 하이퍼파라미터 튜닝(예: 레이어 수 또는 치수 변경)이 필요하지 않습니다.

공유 지수와 좁은 맨티사를 사용하면 공유 지수를 사용하면 MAC당 비용이 많이 드는 시프터가 필요 없고 좁은 비트폭 곱셈이 조회 테이블과 DSP에 매우 효율적으로 매핑되므로 부동 소수점(전통적으로 FPGA의 아킬레스건)의 비용이 상당히 감소합니다. TI는 FPGA에서 좁은 정밀도를 최대한 활용하기 위해 2비트 또는 3비트를 패킹하는 등 다양한 전략을 사용합니다.

셀에 최적화된 소프트 로직 배율기 및 가산기와 결합된 DSP 블록으로 곱셈을 수행하면 Stratix 10 280 FPGA에 최대 96,000개의 MAC을 배포할 수 있습니다.

VII. 평가

이 섹션에서는 BW NPU 아키텍처의 확장성을 입증하는 3세대 인텔 FPGA의 하드웨어 구현 결과를 소개합니다.

A. FPGA 구현 결과

3세대 인텔 FPGA, 성숙 단계에 접어든(오늘날 기준으로는 비교적 작은) 스트라틱스 V D5 디바이스, 최신 FPGA 디바이스인 아리아 10 1150 및 스트라틱스 10 280[1]을 대상으로 했습니다. 모든 BW NPU 인스턴스는 실제 하드웨어에 대해 검증되었으며 보고된 리소스 사용량 및 클럭 주파수 결과는 Stratix V의 경우 Quartus 15.1.1, Arria 10의 경우 Quartus Prime 17.0.0, Stratix 10의 경우 Quartus Prime 17.1ir.2의 최종 포스트 핏 보고서를 기반으로 합니다.

섹션 VI에서 설명한 높은 수준의 합성 시간 매개변수화를 통해 각 BW NPU 인스턴스를 서비스할 모델의 요구 사항을 충족하도록 조정하는 동시에 파생되지 않은 FPGA 하드웨어를 가장 효율적으로 사용할 수 있습니다. 표 III은 3세대 인텔 FPGA 디바이스를 대상으로 한 BW S5, BW A10, BW S10의 세 가지 BW 구성에 대한 HW 구현 결과를 보여줍니다. 워크로드 및 모델 요구 사항에 따라 주어진 타겟 FPGA의 제한 리소스를 모두 소진할 때까지 다양한 차원에서 BW NPU 설계를 확장할 수 있습니다. 이 평가의 나머지 부분은 BW S10 인스턴스에 중점을 둡니다.

B. RNN 성능 분석

딥스피치[17]와 같은 인기 있는 DNN 모델의 대표 레이어가 포함된 마이크로 벤치마크 세트인 딥벤치[16]를 사용하여 BW S10 인스턴스의 성능을 물리적으로 측정합니다. 우리는 낮은 배치 크기에서 GRU/LSTM 추론 테스트에 중점을 둡니다. 스트라틱스 10 280에서 구현된 BW NPU와 최신 NVIDIA Titan Xp GPU에서 딥벤치가 발표한 결과 사이의 지연 시간 및 컴퓨팅 처리량 추세를 비교합니다.

표 IV는 실험에 사용된 하드웨어를 요약한 것입니다. 두 장치 모두 비슷한 공정으로 만들어졌지만

1Stratix 10 결과는 사전 생산된 실리콘에서 측정되었습니다.

BW NPU 인스턴스	#MV 타일	#레인	네이티브 딤.	MRF 크기	#MFU	대상 장치	#ALM(%)	#M20K(%)	DSP(%)	주파수(MHz)	피크 TFLOPS
BW S5	6	10	100	306	2	스트라틱스 V D5	149641 (87%)	1192 (59%)	1047 (66%)	200	2.4
BW A10	8	16	128	512	2	Arria 10 1150	216602 (51%)	2171 (80%)	1518 (100%)	300	9.8
BW S10	6	40	400	306	2	스트라틱스 10 280	845719 (91%)	8192 (69%)	5245 (91%)	250	48

표 III
다양한 BW NPU 구성 및 FPGA에 대한 하드웨어 구현 결과. 괄호 안의 숫자는 전체 디바이스 리소스 중 차지하는 비율을 나타냅니다.

노드의 경우, 주로 사용되는 수치 형식이 다르기 때문에 TDP와 피크 TFLOPS가 다릅니다. 원시 처리량과 레이턴시 수치를 모두 제시하고 하드웨어 사용률, 관찰된 추세, 이상적인 데이터 플로우 머신과의 비교를 바탕으로 결론을 도출합니다.

	Titan Xp	BW S10
숫자 유형	Float32	BFP(1초.5e.2m)
피크 TFLOPS	12.1	48.0
TDP(W)	250	125
프로세스	TSMC 16nm	인텔 14nm

표 IV
하드웨어 사양 실험

1) 배치 없음: 배치 크기가 1이면 요청이 도착하는 즉시 처리되므로 클라우드 서비스 지연 시간이 가장 짧습니다. 또한 배치 대기열과 런타임이 필요하지 않으므로 소프트웨어 API와 배포 복잡성을 더욱 간소화합니다. 표 V는 각 벤치마크의 원시 유효 TFLOPS와 실행 레이턴시를 보여줍니다. BW NPU는 배치 1에서 *모든* 딥 벤치 레이어를 4ms 미만으로 실행할 수 있으며, 수백 개의 타임스텝에 걸쳐 대규모 GRU의 경우 최대 **35.9 유효 TFLOPS에** 도달할 수 있습니다. 이는 Titan Xp에 비해 약 두 배의 이점을 제공합니다. 이는 부분적으로는 좁은 정밀도에서 높은 피크 TFLOPS를 제공하는 BW NPU에 기인하지만, 더 중요한 이유는 하드웨어 활용도가 향상되었기 때문입니다. 그림 7은 각 레이어에서 도달한 피크 TFLOPS의 백분율인 사용률을 보여줍니다. 배치 크기 1에서 BW NPU는 중대형 LSTM/GRU(1500 차원 *이상)의* 경우 피크 FLOPS의 23%~75%에 도달합니다. 이는 Titan Xp의 활용률에 비해 4~23배 향상된 수치입니다. BW NPU는 온칩 RAM 대역폭, 파이프라인 종속 RNN 벡터를 완전히 노출하고 모든 수준의 매트릭스-벡터 병렬 처리를 활용하여 컴퓨팅 유닛을 계속 바쁘게 사용할 수 있습니다. 이는 배치 크기 1에서 낮은 지연 응답 시간으로 이어집니다. GRU/LSTM 숨겨진 차원이 감소함에 따라 병렬 처리량과 총 연산량도 감소합니다. 이에 따라 Titan Xp와 BW 모두에서 컴퓨팅 활용도가 떨어집니다. 또한, 작은 LSTM/GRU의 경우 (1) 상대적으로 큰 네이티브 타일 차원으로 인해 작업이 낭비될 수 있고, (2) 종속 데이터가 빠르게 다시 쓰여지는 것을 지연시키는 딥 파이프라인으로 인해 BW의 활용도가 떨어집니다. 그러나 BW의 재구성 가능한 아키텍처를 통해 전체 DNN 차원에 따라 다양한 병렬 처리 정도(예: 기본

벤치마크	장치	지연 시간(ms)	TFLOPS	사용률
GRU h=2816 t=750	SDM	1.581	-	-
	BW	1.987	35.92	74.8
	Titan Xp	178.60	0.40	3.3
GRU h=2560 t=375	SDM	0.661	-	-
	BW	0.993	29.69	61.8
	Titan Xp	74.62	0.40	3.3
GRU h=2048 t=375	SDM	0.438	-	-
	BW	0.954	19.79	41.2
	Titan Xp	51.59	0.37	3.0
GRU h=1536 t=375	SDM	0.266	-	-
	BW	0.951	11.17	23.3
	Titan Xp	31.73	0.33	2.8
GRU h=1024 t=1500	SDM	0.558	-	-
	BW	3.792	4.98	10.4
	Titan Xp	59.51	0.32	2.6
GRU h=512 t=1	SDM	0.00017	-	-
	BW	0.013	0.01	0.2
	Titan Xp	0.06	0.05	0.4
LSTM h=1024 t=25	SDM	0.037	-	-
	BW	0.074	2.62	5.4
	Titan Xp	5.27	0.32	2.7
LSTM h=1536 t=50	SDM	0.145	-	-
	BW	0.145	13.01	27.1
	Titan Xp	6.20	0.30	2.5
LSTM h=1024 t=25	SDM	0.011	-	-
	BW	0.074	5.68	11.8
	Titan Xp	1.87	0.22	1.9
LSTM h=512 t=25	SDM	0.0038	-	-
	BW	0.077	1.37	2.8
	Titan Xp	1.26	0.08	0.7
LSTM h=256 t=150	SDM	0.0126	-	-
	BW	0.425	0.37	0.8
	Titan Xp	1.99	0.08	0.7

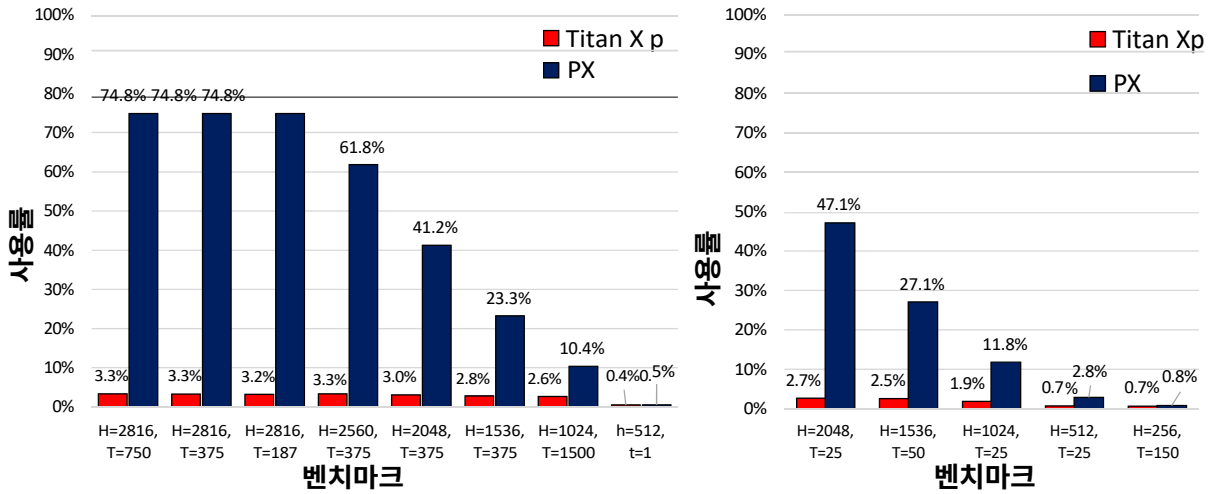
차원 축소)를 조정할 수 있다는 점을 강조합니다, 사용률을 회복하고 지연 시간을 줄일 수 있습니다.

2) 임계 경로 분석. 섹션 III의 임계 경로 분석 방법을 사용하여 동일한 클럭 주파수를 가진 BW와 SDM 간의 지연 시간 차이를 분석합니다.

표 V
딥벤치 RNN 추론 성능 결과

컴퓨팅 리소스 수를 BW S10(250MHz, 96000 MVM MAC 및 기타 모든 컴퓨팅 밸런스)으로 가정합니다. 표 V의 SDM 레이턴시를 BW S10과 비교하면, 대규모 GRU 및 LSTM(차원 2000 *조과)의* 경우 BW S10이 2.17배 이내임을 알 수 있습니다. 그러나 BW S10의 고차원 MVM과 딥 파이프라인으로 인해 평가된 모든 모델의 정상 상태 시간 단계당 지연 시간이 252~296마이크로초 사이로 크기에 관계없이 본질적으로 동일하기 때문에 나머지 모델에서는 이 계수가 떨어집니다. 향후 작업에서는 MVM의 세분성을 줄이고 MFU 리소스를 늘려 SDM처럼 DNN 그래프의 여러 스퍼를 병렬로 평가하는 데 중점을 둘 것입니다.

3) 소규모 배치: 일부 클라우드 서비스는 약간 더 긴 응답 대기 시간을 허용할 수 있으며, 이 경우 소량의 일괄 처리를 사용할 수 있습니다. 그림 8에서는 배치 수에 따른 사용률 확장을 보여줍니다. DeepBench는 추론 배치 크기를 4로 제한하지만, 다음과 같은 장점도 있습니다.



(a) GRU (b) LSTM

그림 7. 딥벤치 RNN 추론 실험에서의 하드웨어 활용도. h = 숨겨진 차원, t = 시간 단계

는 32의 배치 크기를 비교 지점으로 표시합니다. BW는 한 번에 하나의 입력을 실행하므로 배치 크기가 증가해도 사용률은 일정하게 유지됩니다. 배치 크기가 2를 초과하는 경우 모든 입력 배치에서 각 RNN 타임스텝에 대한 계산을 인터리빙하여 종속성을 더욱 분리함으로써 추가적인 펌웨어 최적화가 필요하다는 점에 유의하세요. 이렇게 하면
는 깊은 BW 파이프라인을 항상 채울 수 없는 소형 LSTM/GRU 레이어의 활용도를 높이는 데 특히 효과적입니다. 이 최적화는 향후 작업을 위해 남겨져 있습니다.

이와는 대조적으로, GPU SM을 채우기 위한 독립적인 병렬 작업이 더 많아지기 때문에 배치 크기가 증가함에 따라 GPU 사용률도 비례적으로 증가합니다. 그러나 배치 크기가 4인 경우, 대규모 RNN의 경우에도 Titan Xp의 사용률은 13% 미만으로 유지됩니다. 배치 크기를 32로 늘리면 GPU 사용률이 높아지지만, 실제로는 SLA를 위반하지 않고는 클라우드에서 DNN을 서비스하는 데 이러한 큰 배치 크기를 사용할 수 없습니다.

4) 전력 효율성: 모든 온칩 IP/리소스를 사용하는 파워 바이러스 설계를 실행하여 Stratix 10 280 FPGA의 피크 칩 전력 소비를 125W로 측정했습니다. 각 딥 벤치 실험에 대한 전력 소비를 측정하지는 않았지만 125W 피크 전력 소비 수치를 기반으로 보수적으로 추정하면 디바이스 활용도가 높은 대형 모델을 실행할 때 BW의 전력 효율은 **287GFLOPS/W에 달할 것입니다.**

C. CNN 성능 분석

BW NPU 아키텍처는 또한 짧은 지연 시간으로 대규모 CNN 모델을 가속화하고 제공할 수 있습니다. 이 섹션에서는 ResNet-50[8]을 기반으로 프로덕션 이미지 기반 피처라이저를 실행하는 CNN에 특화된 BW NPU 변형의 예비 결과에 대해 보고합니다. 토폴로지 및 계산 요구 사항은 CPU에서 실행되는 시나리오별 분류기(예: Bing 랭킹 파이프라인의 의사 결정 트리)로 대체된 최종 고밀도 레이어를 제외하고는 원래 보고된 모델과 거의 동일합니다.

표 VI는 호스팅된 BW NPU에서 ResNet-50 기반 피처라이저를 독립형으로 실행할 때의 레이턴시와 처리량을 비교한 것입니다.

표 VI
ARRIA 10의 브레인웨이브 NPU는 경쟁력 있는
배치 크기 1에서NVIDIA P40 GPU에 대한 처리량 및 지연 시간
를 RESNET-50 기반 이미지 피처라이저에 적용합니다.

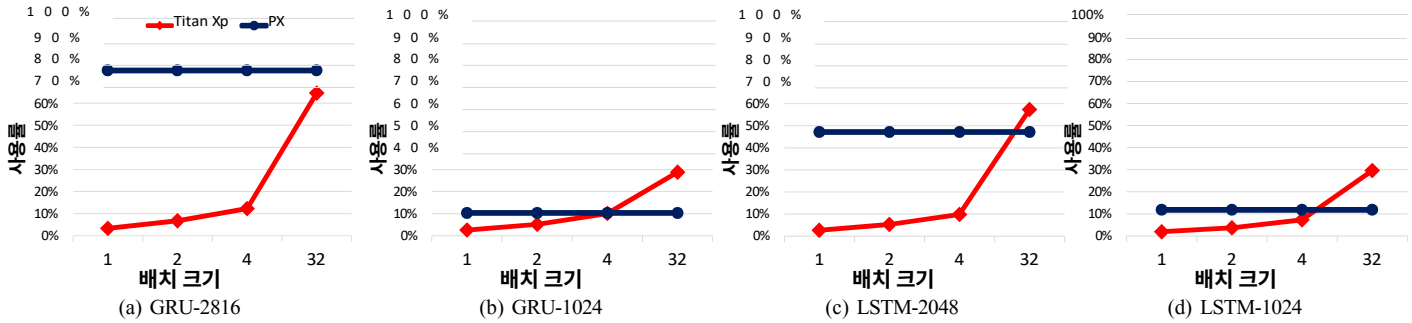
	Nvidia P40	BW CNN A10
기술 노드	16nm TSMC	20nm TSMC
프레임워크	TF 1.5 + TensorRT 4	TF + BW
정밀도	INT8	BFP(1초.5e.5m)
IPS(배치 1)	461	559
지연 시간(배치 1)	2.17ms	1.8ms

Arria 10 1150(TSMC 20nm FPGA)에서 하이엔드 추론에 최적화된 Nvidia P40 GPU(TSMC 16nm)와 비교했습니다. 실제 하드웨어에서 측정한 결과에는 단일 요청을 계산하는 데 걸리는 지연 시간과 호스트 CPU와 가속기 간의 PCI Express를 통한 전송 시간이 포함됩니다.

배치 크기 1에서 INT8 정밀도를 사용하는 하이엔드 P40은 초당 461개의 추론(IPS)을 달성하는 반면, Arria 10의 BW NPU는 559개의 IPS를 달성합니다. 언로드된 시스템에서 BW NPU는 모델의 단일 인스턴스를 1.8ms에 처리하는 반면, P40은 2.17ms를 달성합니다. 예를 들어 배치 크기가 16인 경우 P40은 2,270 IPS를 달성하지만, 배치 대기열에서 입력을 형성하는 데 필요한 시간은 포함되지 않은 배치당 레이턴시가 7ms로 증가합니다. 이러한 결과는 BW NPU가 단일 배치, 저지연 서빙에 효과적인 아키텍처이며, 컴퓨팅 집약적인 CNN에서는 하이엔드 차세대 GPU와 경쟁하고, RNN에서는 훨씬 더 빠른 성능을 제공한다는 것을 보여줍니다.

VIII. 관련 작업

지난 몇 년간 딥 러닝의 끊임없는 성공(예: [8], [18])은 시스템/AI 연구의 폭발적인 증가와 딥 러닝을 위한 소프트웨어 프레임워크의 대중화를 촉진했습니다(예: 텐서플로우 [6], Caffe [19] 등). 이에 따라 아래에서 설명하는 것처럼 많은 가속기가 제안되었습니다. 단일 스레드의 조합



(a) GRU-2816 (b) GRU-1024 (c) LSTM-2048 (d) LSTM-1024

그림 8. 배치 크기 증가에 따른 사용률 확장.

프로그래밍 모델, CNN 및 RNN에 적합한 평면 벡터 공간, 온칩 메모리 내 모델 고정, 합성 시 다양한 벡터 길이 및 데이터 유형에 대한 전문화 등은 BW NPU를 관련 작업의 방대한 분량과 차별화합니다.

컴퓨터 비전을 위한 컴퓨팅 집약적인 딥 CNN을 가속화하는 데 초점을 맞춘 ASIC 기반 DNN 설계가 많이 등장했습니다(예: [20]-[32]). 2-D CNN 모델은 높은 수준의 병렬 처리와 데이터 재사용을 보여주기 때문에 메모리 집약적인 RNN에 비해 가속화의 대상이 되기 쉽습니다.

"DianNao" AI 프로세서 제품군[4], [20], [33]-[35]는 소형 임베디드 클라이언트부터 AI 슈퍼컴퓨팅에 이르는 다양한 설계를 교육과 서비스 모두에 적용합니다. DaDian-Nao 접근 방식은 모델 고정을 사용하여 에너지 전송과 메모리 병목 현상을 최소화하여 공간적으로 분산된 컴퓨팅 유닛에 공급하며, 이는 데이터센터 규모에 적용하는 전략과 유사합니다. Eyeriss[36]는 딥 CNN 모델을 가속화하기 위한 공간 데이터 흐름 엔진을 탐색하고 GPU보다 10배 더 높은 에너지 효율을 달성합니다. 다른 접근 방식에서는 비트 수준에서 불필요한 연산을 제거하는 방법도 모색하고 있습니다[37]-[39].

클라우드 규모 워크로드의 상당 부분은 RNN, MLP, 1-D CNN 및 기타 메모리 집약적인 DNN 알고리즘을 필요로 하는 텍스트 중심 시나리오에 의해 구동되며, 이는 많은 기존 접근 방식에서는 고려하지 않습니다. EIE[12]와 ESE[40]는 압축 모델[41]을 직접 처리하는 실행 엔진을 통해 RNN/MLP의 서비스를 최적화하는 밀접하게 관련된 두 가지 작업입니다. Deephi [42]는 FPGA에서 압축된 모델에서 계산하는 CNN 최적화 및 RNN 최적화 엔진의 생성 인스턴스를 제공하지만, BW NPU와 달리 비압축 모델에서 고성능을 달성하는 컨버지드 엔진은 제공하지 않습니다.

DNN의 상업적 실행 가능성은 구글의 TPU [10], [43], [44], 엔비디아의 SCNN [45], 웨이브(Wave) 등 대기업과 스타트업의 다양한 제품을 탄생시켰습니다.
컴퓨팅 [46], 그래프코어 [3], 모비디우스 [47] 등(전체 목록은 [48]에서 확인할 수 있습니다). TPU1은 CNN 및 RNN 추론을 위해 대규모로 배포된 DNN 가속기의 첫 번째 보고 사례입니다. 이 가속기는 CNN에 대해 높은 수준의 효율성을 달성하지만, 고효율(최소 16개)을 위해 배치가 필요하고 RNN에서 상대적으로 낮은 하드웨어 사용률로 성능을 발휘합니다.
(4% 미만), 높은 미니배치 크기[10]에서도 마찬가지입니다.

추상화. DNN 가속기를 일반화하기 위한 노력

더 넓은 범위의 DNN 모델을 지원할 수 있도록 DNN 전용 명령어 집합 아키텍처(ISA)에 대한 연구로 이어졌습니다. 예를 들어, 캠브리콘(Cambricon)[49]은 스칼라, 벡터, 행렬, 제어 명령어와 같은 데이터 유형을 통합하여 광범위한 DNN에 대한 커버리지를 제공하는 로드 스토어 아키텍처로, 범용 ISA보다 높은 코드 밀도를 달성하고 기존 가속기보다 더 광범위한 모델을 지원할 수 있습니다. BW NPU는 이와 유사한 전략을 사용하여 적은 면적의 풋프린트를 차지하면서도 광범위한 DNN 모델을 효율적으로 실행할 수 있도록 최적화된 특수 ISA를 활용합니다.

모델 압축 및 좁은 정밀도. 신경망의 활성화와 가중치의 수치 정밀도를 낮추는 것은 컴퓨팅 성능과 전력 효율을 향상시키는 전략 중 하나입니다. 이항 신경망과 고정 소수점 숫자 표현은 우수한 효율성을 달성하기 위해 널리 사용되는 접근 방식입니다 [11], [13], [14], [50]-[55]. 그러나 이러한 작업은 단일 유형의 신경망, 즉 CNN만을 대상으로 한다는 점에서 한계가 있습니다. RNN을 포함한 보다 다양한 신경망을 지원하기 위해 BW NPU는 수치 양자화에 대한 새로운 접근 방식을 활용하는데, 이는 대상 애플리케이션에 따라 양자화 비트 수를 확장할 수 있을 뿐만 아니라 공유 벡터 지수를 활용하여 더 넓은 동적 범위를 구현함으로써 모델 정확도의 손실을 무시할 수 있거나 전혀 없앨 수 있게 해줍니다.

모델 압축과 가중치 가지치기는 좋은 효율성을 달성하기 위한 또 다른 접근 방식입니다. 이러한 예로는 심층 압축[41]이 있지만 다른 방법도 있습니다[56]-[61]. 이러한 접근 방식은 과도하게 프로비저닝된 대규모 모델에 대해서는 훌륭한 결과를 얻을 수 있지만, 모델 정확도에 심각한 단점이 있기 때문에 임의의 크기와 복잡성을 가진 모델에 일반화하기는 어렵습니다.

고성능을 달성하기 위해 BW NPU가 사용하는 핵심 전략은 신경망을 고정하는 것으로, 실시간으로 모델을 제공하는 데 필요한 높은 메모리 읽기 대역폭을 달성하기 위해 모델 가중치를 온칩 메모리에 고정할 수 있다는 아이디어입니다. 바이두의 퍼시스턴트 RNN[62]은 분산 학습을 위해 GPU 장치를 대상으로 유사한 접근 방식을 사용하지만, GPU의 온칩 메모리는 용량이 더 제한적이고 정밀도가 유연하지 않습니다. BW NPU는 완전히 구성 가능한 FPGA에 피닝을 적용하고 FPGA의 스케일아웃 네트워크를 사용하여 모델 가중치가 증가하는 경우를 해결합니다.

온칩 용량을 초과합니다.

특히 컨볼루션 신경망(CNN)을 가속화하기 위해 FPGA를 대상으로 하는 DNN 가속기에 대한 많은 연구가 진행되어 왔습니다[42], [63]-[94]. BW NPU는 DNN 가속을 위해 최신 세대 인텔 스트라틱스 10 280 디바이스를 활용한 최초의 제품 중 하나로, RNN에서 GPU보다 훨씬 더 높은 성능을 달성했습니다[95]. 기타 연구 노력 또한 존재에 아날로그 및 [96]-[99]와 같은 뉴로모픽형 컴퓨팅 접근법에도 존재합니다. 그러나 이러한 접근 방식은 일반적으로 보다 인공적인 신경망에 비해 최첨단 정확도를 제공하지 않습니다. 효율적인 DNN에 대한 자세한 조사 가속기, [48] 및 [100]을 참조하세요.

IX. 결론

이 백서에서 설명한 시스템은 여러 기술을 사용하여 미니배칭 없이 실시간 AI를 위한 높은 처리량과 낮은 지연 시간을 달성합니다. 이 시스템은 온칩 메모리에 모델을 고정하고 단일 제어 스레드에서 메가-SIMD 병렬 처리를 추출하며, 일부 복합 명령어는 수백만 개의 독립적인 연산을 생성합니다. 계층적 디코드 및 디스패치는 이러한 연산을 고정 길이 벡터 연산으로 분할한 다음 분산된 기판에서 스케줄링하여 병렬로 작동하고 직접 생산자-소비자 데이터 흐름 라우팅을 활용하여 파이프라인 버블을 줄입니다. 또한 데이터 경로를 파라미터화하여 데이터 유형, 기본 벡터 크기, 레인 수, 행렬-벡터 단위 수 등 다양한 모델과 일치하도록 제공할 수 있습니다. 이러한 기술을 종합하면 동등한 공정 기술로 구축된 고성능 GPGPU보다 RNN 컬렉션의 활용도를 높이고 지연 시간을 줄일 수 있습니다. 더 큰 모델의 경우 지연 시간은 GPGPU보다 10~90배 낮으며, 배치 크기 32가 적용될 때까지 모든 벤치마크에서 유효 활용도가 GPU보다 높습니다. 이 시스템은 현재 많은 모델을 지원하며 대규모 프로덕션(수만 노드)에서 실행되고 있습니다.

데이터 흐름 분석 결과, 우리가 활용하기 위해 측정한 대규모 모델에서도 여전히 상당한 병렬성이 존재하는 것으로 나타났습니다. 리소스가 증가함에 따라 기본 벡터 길이도 증가해야 하므로 제어 오버헤드가 지배적이지 않습니다. 아직 단일 스레드 모델의 익스플로잇 가능한 DLP 배수의 한계와 이 모델이 몇 개 남지 않은 실리콘 프로세스 노드가 제공하는 증가된 영역으로 계속 확장될지 여부는 알 수 없습니다. 그러나 Stratix V에서 Arria 10으로, 그리고 Stratix 10으로 확장하는 것은 효과적이었으며 지속적인 활용이 이루어지고 있습니다. 이 분야에서 또 다른 미지의 영역은 이상적인 클럭 속도입니다. FPGA 클럭 속도는 고주파 ASIC에 비해 낮기 때문에 아키텍처의 활용도를 높일 수 있습니다. 프로덕션 실리콘으로 Stratix 10 구현 빈도를 높이면 성능은 향상되지만 파이프라인 버블이 증가하여 효율성은 떨어집니다. ILP를 활용하는 CPU와 마찬가지로, NPU 공간에서도 벡터 레벨 프로세싱을 활용하기 위한 주파수와 효율성 간의 최적의 균형을 찾아야 하는데, 현재로서는 이를 알 수 없습니다.

참조

[1] E. Chung *외*, "데이터센터 규모에서 지속적 신경망 가속화", *2017 IEEE Hot Chips 29 심포지엄*, 2017년 8월.

[2] --, "프로젝트 브레인웨이브를 통한 데이터센터 규모의 실시간 DNN 서비스", *IEEE MICRO: Hot Chips*, 2018년 4월.

[3] N. 툰과 S. 놀스, "그래프코어", https://www.graphcore.ai, 2017.

[4] Y. Chen *외*, "DaDianNao: 기계 학습 슈퍼컴퓨터," in *Proc. Int. (MICRO)*, 2014, 609-622쪽.

[5] A. Putnam *외*, "대규모 데이터센터 서비스 가속화를 위한 재구성 가능한 패브릭", *41st Annu. Int. (ISCA)*, 2014, 13-24쪽.

[6] M. Abadi *외*, "TensorFlow: 대규모 머신 러닝을 위한 시스템", *제 12회 USENIX 운영 체제 설계 및 구현 심포지엄(OSDI)*, 2016, 265-283쪽.

[7] S. Hochreiter와 J. Schmidhuber, "장단기 기억," *Neural Comput.*, 9권, 8호, 1735-1780쪽, 1997 년 11월.

[8] K. He *외*, "이미지 인식을 위한 심층 잔여 학습", *2016 IEEE 컴퓨터 비전 및 패턴 인식 컨퍼런스(CVPR)*, 2016년 6월, 770-778페이지.

[9] C. L. Lawson *외*, "Fortran 사용을 위한 기본 선형 대수 하위 프로그램", *ACM Trans. Math. Softw.*, vol. 3, 308-323쪽, 1979 년 9 월.

[10] N. P. Jouppi *외*, "데이터센터 내 텐서 프로세싱 유닛 성능 분석," *44th Annu. Int. Symp. on Computer Archi- tecture (ISCA)*, 2017, 1-12쪽.

[11] S. 굽타 *외*, "제한된 수치 정밀도를 가진 딥 러닝," *Proc. 기계 학습 컨퍼런스 - 37권*, 2015, 1737-1746쪽.

[12] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proc. Int. Symp. on Computer Architecture (ISCA)*, 2016, 243-254쪽.

[13] M. Courbariaux와 Y. Bengio, "BinaryNet: 가중치와 활성화가 +1 또는 -1로 제한된 심층 신경망 훈련", *CoRR*, vol.

[14] U. Ko¨ster *외*, "Flexpoint: 심층 신경망의 효율적인 학습을 위한 적응형 수치 형식," in *NIPS*, 2017.

[15] J. H. 윌킨슨, 대수 *과정에서의 반올림 오류*, 1판. 잉글우드 클리프스, 뉴저지: 프렌티스 홀, 1963.

[16] S. Narang과 G. Diamos, "Baidu DeepBench," https://github.com/baidu-research/DeepBench, 2017.

[17] A. Y. Hannun *외*, "Deep Speech: 엔드 투 엔드 음성 인식 확장," *CoRR*, vol.

[18] A. Krizhevsky, I. Sutskever 및 G. E. Hinton, "ImageNet 분류를 위한 심층 컨볼루션 신경망", *신경 정보 처리 시스템에 관한 제25회 국제 컨퍼런스 - 1권*, 2012, 1097-1105쪽.

[19] Y. Jia *외*, "Caffe: 빠른 기능 구현을 위한 컨볼루션 아키텍처," *제 22회 ACM 멀티미디어 국제 컨퍼런스*, 2014, 675-678쪽.

[20] Y. Chen *외*, "DianNao Family: 기계 학습을 위한 에너지 효율적인 하드웨어 가속기," *Commun. ACM*, vol. 59, no. 11, pp. 105-112, Oct.

[21] P. Whatmough, "DNN 엔진: 임베디드 대중을 위한 16nm 서브-uj 심층 신경망 추론 가속기", *2017 IEEE Hot Chips 29 심포지엄*, 2017년 8월.

[22] C. Farabet *외*, "Neuflow: 비전을 위한 런타임 재구성 가능한 데이터 흐름 프로세서," in *Proc. 임베디드 컴퓨터 비전 워크샵 (ECVW'11)*, 2011, (초청 논문).

[23] B. Moons and M. Verhelst, "실시간 대규모 컨브넷을 위한 0.3-2.6 TOPS/W 정밀 확장 가능 프로세서," 2016.

[24] R. LiKamWa *외*, "RedEye: 연속 모바일 비전을 위한 아날로그 컨브넷 이미지 센서 아키텍쳐," in *Proc. Int. (ISCA)*, 2016, 255-266쪽.

[25] P. Chi *외*, "PRIME: ReRAM 기반 주 메모리에서 신경망 계산을 위한 새로운 메모리 내 처리 아키텍처", *43rd Annu. Int. (ISCA)*, June 2016, 27-39쪽.

[26] S. 차크라다르 *외*, "컨볼루션 신경망을 위한 동적으로 구성 가능한 코프로세서", *37회 Annu. Int. (ISCA)*, 2010, 247-257쪽.

[27] S. Venkataramani *외*, "ScaleDeep: 딥 네트워크 학습 및 평가를 위한 확장 가능한 컴퓨팅 아키텍처", *44th Annu. Int. 컴퓨터 아키텍처 (ISCA)*, 2017, 13-26쪽.

[28] S. Li *et al.*, "DRISA: A DRAM-based Reconfigurable In-Situ Accelerator," in *Proc. Int. (MICRO)*, 2017, 288-301쪽.

[29] B. Reagen *외*, "Minerva: Enabling Low-Power, High-Accuracy Deep Neural Network Accelerators," in *Proc. Int. (ISCA)*, June 2016, 267-278쪽.

[30] M. Peemen *외*, "메모리 중심 가속기 설계를 위한 합 성 곱 신경망," *Proc. 31st IEEE Int. (ICCD)*, Oct 2013, 13-19쪽.

[31] S. W. Park *외*, "빅데이터 애플리케이션을 위한 테트라 병렬 MIMD 아키텍처를 갖춘 에너지 효율적이고 확장 가능한 딥 러닝/인공지능 프로세서," *IEEE Trans on Biomed Circuits Syst*, 9권 6호, 838-848쪽, 2015년 12월.

[32] V. Gokhale *외*, "심층 신경망을 위한 240G-ops/s 모바일 코프로세서," *2014 IEEE 컴퓨터 비전 및 패턴 인식 워크샵*, 2014년 6월, 696-701페이지.

[33] T. Chen *외*, "DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning," in *Proc. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014, 269-284쪽.

[34] Z. Du *et al.*, "ShiDianNao: 시디안나오: 센서에 더 가까운 비전 처리," in *Proc. Int. (ISCA)*, June 2015, 92-104쪽.

[35] D. Liu *외*, "PuDianNao: A Polyvalent Machine Learning Accelererator," in *Proc. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: ACM, 2015, 369-381쪽.

[36] Y. H. Chen, J. Emer, 및 V. Sze, "Eyeriss: 컨볼루션 신경망을 위한 에너지 효율적인 데이터 흐름을 위한 공간 아키텍처," *43rd Annu. Int. (ISCA)*, June 2016, 367-379쪽.

[37] P. Judd *외*, "Stripes: 비트-직렬 심층 신경망 컴퓨팅," in *49th Annu. Int. 마이크로아키텍처(MICRO)*, Oct 2016, pp. 1-12.

[38] J. Albericio *외*, "Cnvlutin: 비효율적인 뉴런이 없는 심층 신경망 컴퓨팅", *43rd Annu. Int. (ISCA)*, June 2016, 1-13쪽.

[39] --, "Bit-pragmatic Deep Neural Network Computing," in *Proc. Int. (MICRO)*, 2017, 382-394쪽.

[40] S. Han *et al.*, "ESE: 효율적인 음성 인식 엔진: FPGA에서 스파스 LSTM을 이용한 음성 인식 엔진," in *Proc. ACM/SIGDA Int. Symp. on Field- Programmable Gate Arrays*, 2017, 75-84쪽.

[41] S. Han, H. Mao, and W. J. Dally, "Deep Compression: 프루닝, 훈련된 양자화 및 허프만 코딩을 사용한 심층 신경망 압축," *학습 표현에 관한 국제 컨퍼런스*, 2016.

[42] K. Guo *외*, "모델에서 FPGA로: 효율적인 신경망 가속을 위한 소프트웨어-하드웨어 공동 설계", *2016 IEEE Hot Chips 28 심포지엄*, Aug 2016, 1-27쪽.

[43] C. Young, "Evaluation of the Tensor Processing Unit: 데이터 센터를 위한 심층 신경망 가속기", *2017 IEEE Hot Chips 29 심포지엄*, 2017년 8월.

[44] J. Dean, "기계 학습을 통한 인공 지능의 최근 발전과 컴퓨터 시스템 설계에 대한 시사점", *2017 IEEE Hot Chips 29 심포지엄*, 2017년 8월.

[45] A. Parashar *외*, "SCNN: 압축-희소 컨볼루션 신경망을 위한 가속기," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 27-40, Jun.

[46] C. Nicol, "심층 신경망 학습을 위한 데이터 흐름 처리 칩," *2017 IEEE Hot Chips 29 심포지엄*, 2017년 8월.

[47] D. Moloney, "임베디드 심층 신경망: 모든 것의 비용과 무의 가치," *2016 IEEE Hot Chips 28 심포지엄*, Aug 2016, 1-20쪽.

[48] V. Sze *외*, "심층 신경망의 효율적인 처리: 튜토리얼 및 설문조사," *Proceedings of the IEEE*, 105, 12권, 2295-2329쪽, 2017년 12월.

[49] S. Liu *외*, "Cambricon: 신경망을 위한 명령어 집합 아키텍처," in *Proc. Int. (ISCA)*, June 2016, 393-405쪽.

[50] M. Rastegari *외*, "XNOR-Net: Bi-nary Convolutional Neural Networks를 이용한 이미지넷 분류", *In Proceedings of the European Conference on Computer Vision*, 2016.

[51] B. Moons *외*, "근사 컴퓨팅을 통한 에너지 효율적인 ConvNets," *IEEE Winter Conf. on Appl. of Computer Vision (WACV)*, 2016, 1-8쪽.

[52] M. Courbariaux, Y. Bengio, 및 J.-P. David, "Binaryconnect: 전파 중 이진 가중치를 사용한 심층 신경망 훈련," in *Pro- ceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. 미국 매사추세츠주 케임브리지: MIT Press, 2015, 3123-3131쪽.

[53] P. Judd *외*, "심층 신경망에서 경계 기억을 위한 정확도 감소 전략," *CoRR*, vol.

[54] P. Gysel, M. Motamedi, 및 S. Ghiasi, "하드웨어 지향적 컨볼루션 신경망 접근법," *CoRR*, vol.

[55] P. Colangelo *외*, "세분화된 바이너리 신경망 가속- 통합 FPGA가 있는 인텔 제온 프로세서를 이용한 작업", *Proc. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 4월 2017, 135-135쪽.

[56] J. Cong 및 B. Xiao, *컨볼루션 신경망에서 계산 최소화*. Cham: 스프링거 국제 출판, 2014, 281- 290쪽.

[57] J. Yu *외*, "Scalpel: 기본 하드웨어 병렬 처리에 맞게 DNN 가지치기를 사용자 정의하기," in *Proc. Int. (ISCA)*, 2017, 548-560쪽.

[58] T.-J. Yang, Y.-H. Chen, V. Sze, "에너지 인식 가지치기를 이용한 에너지 효율적인 컨볼루션 신경망 설계", *2017 IEEE 컴퓨터 비전 및 패턴 인식 컨퍼런스(CVPR)*, 6071-6079쪽, 2017.

[59] Y. Kim *외*, "고속 저전력 모바일 애플리케이션을 위한 심층 컨볼루션 신경망의 압축", *CoRR*, vol.

[60] F. N. Iandola *외*, "SqueezeNet: 50배 적은 매개변수와 *1MB 미만의* 모델 크기로 Alexnet 수준의 정확도", *CoRR*, vol.

[61] S. 한, "효율적인 신경망을 위한 가중치 및 연결 학습", *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, 1135-1143쪽.

[62] G. Diamos *외*, "Persistent RNNs: Stashing Recurrent Weights On- Chip," in *Proc. 기계 학습 컨퍼런스*, 2016, 2024-2033쪽.

[63] Y. Shen, M. Ferdman, 및 P. Milder, "자원 분할을 통한 CNN 가속기 효율성 극대화", *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, 535-547쪽.

[64] J. Ouyang *외*, "SDA: SDA: 대규모 dnn 시스템을 위한 소프트웨어 정의 가속기", *2014 IEEE Hot Chips 26 심포지엄*, Aug 2014, 1-23쪽.

[65] --, "SDA: 범용 빅데이터 분석 시스템을 위한 소프트웨어 정의 가속기", *2016 IEEE Hot Chips 28 심포지엄,* 2016년 8월, pp. 1-23.

[66] J. Ouyang, "XPU: 다양한 워크로드를 위한 프로그래머블 FPGA 가속기", *2017 IEEE Hot Chips 29 심포지엄*, 2017년 8월.

[67] D. Shin and H.-J. Yoo, "DNPU: 온칩 스테레오 매칭을 지원하는 에너지 효율적인 심층 신경망 프로세서", *2017 IEEE Hot Chips 29 심포지엄*, 2017년 8월.

[68] A. Rahman, J. Lee, and K. Choi, "Efficient FPGA 가속화를 통한 컨볼루션 신경망의 논리적-3D 컴퓨팅 어레이 사용," in *Conf. on Design, Automation & Test in Europe (DATE)*, 2016, 1393-1398 페이지.

[69] A. Podili, C. Zhang, and V. Prasanna, "FPGA에서 컨볼루션 신경망의 빠르고 효율적인 구 현," in *Proc. 28th IEEE Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, July 2017, 11-18쪽.

[70] S. Li *외*, "CNN 희소화 및 가속을 위한 FPGA 설계 프레임워크", *Proc. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, April 2017, 28-28쪽.

[71] L. Lu *et al.*, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *Proc. 심포지엄, 현장 프로그래밍 가능 맞춤형 컴퓨팅 머신(FCCM)*, 2017년 4월, 101-108쪽.

[72] Y. Shen, M. Ferdman, 및 P. Milder, "Escher: 오프칩 전송을 최소화하기 위한 유연한 버퍼링을 갖춘 CNN 가속기", *25th IEEE Int. 심포지엄, 현장 프로그래밍 가능 맞춤형 컴퓨팅 머신(FCCM)*, 2017년 4월, 93-100페이지.

[73] M. Samragh, M. Ghasemzadeh, 및 F. Koushanfar, "효율적인 FPGA 구현을 위한 신경망 커스터마이징", *25th IEEE Int. 심포지엄, 현장 프로그래밍 가능 맞춤형 컴퓨팅 머신(FCCM)*, 2017년 4월, 85-92쪽.

[74] E. Kousanakis *외*, "하이브리드 누설 통합 및 발사 SNN을 컨베이어 HC-2ex FPGA 기반 프로세서에서 가속화하기 위한 아키텍처", *25th IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 4월 2017, 56-63페이지.

[75] S. Yin *외*, "공간 프로그래밍 가능 아키텍처에서 데이터 흐름 그래프 매핑을 위한 컨볼루션 신경망 학습 (초록 전용)," in *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, 295-295쪽.

[76] S. I. Venieris and C. S. Bouganis, "fpgaConvNet: A framework for 매핑 컨볼루션 신경망을 FPGA에 적용하기 위한 프레임워크," in *Proc. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, 40-47쪽.

[77] Y. Li *외*, "A 7.663-TOPS 8.2-W 에너지 효율적인 바이너리 컨볼루션 신경망용 FPGA 가속기(초록만)", *2017 ACM/SIGDA 국제 현장 프로그래밍 가능 게이트 어레이 심포지엄 Pro- ceedings of the 2017 ACM/SIGDA International Symposium on Field- Programmable Gate Array*, ser. FPGA '17. New York, NY, USA: ACM, 2017, 290-291쪽.

[78] H. Nakahara *외*, "A Batch Normalization Free Binarized Convolutional Deep Neural Network on a FPGA (Abstract Only)," in *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, pp. 290-290.

[79] Y. Umuroglu *외*, "FINN: 빠르고 확장 가능한 바이너리 신경망 추론을 위한 프레임워크," in *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, 65-74쪽.

[80] U. Aydonat *외*, "An OpenCLTMDeep Learning Accelerator on Arria 10", *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Array*, ser. FPGA'17. New York, NY, USA: ACM, 2017, 55-64쪽.

[81] Y. Ma *외*, "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks," in *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, pp. 45-54.

[82] C. Zhang과 V. Prasanna, "CPU-FPGA 공유 메모리 시스템에서 콘볼루션 신경망의 주파수 영역 가속", *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, 35-44쪽.

[83] J. Zhang과 J. Li, "컨볼루션 신경망을 위한 OpenCL 기반 FPGA 가속기의 성능 향상", *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, pp. 25-34.

[84] R. Zhao *외*, "소프트웨어 프로그래밍이 가능한 FPGA로 바이너리화된 컨볼루션 신경망 가속화", *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, 15-24쪽.

[85] E. Nurvitadhi *외*, "차세대 심층 신경망 가속화에 있어 FPGA가 GPU를 이길 수 있는가?" in *Proc. ACM/SIGDA Int. 현장 프로그래밍 가능 게이트 어레이*, 2017, 5-14쪽.

[86] J. Qiu *외*, "Convolutional Neural Network를 위한 임베디드 FPGA 플랫폼으로 더 깊이 들어가기", *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2016, 26-35쪽.

[87] N. Suda *et al.*, "Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks," in *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2016, pp. 16-25.

[88] C. Zhang *et al.*, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2015, 161-170쪽.

[89] W. Qadeer *외*, "컨볼루션 엔진: 특수 컴퓨팅의 효율성과 유 연 성 균형", *40th Annu. Int. 컴퓨터 아키텍처 (ISCA)*, 2013, 24-35쪽.

[90] K. Ovtcharov *외*, "데이터 센터에서 특수 하드웨어를 사용하여 대규모 딥 러닝을 가속화하기 위한 방향", *2015 IEEE Hot Chips 27 심포지엄*, 2015년 8월, 1-38페이지.

[91] C. Farabet *외*, "CNP: 컨볼루션 네트워크를 위한 FPGA 기반 프로세서," *2009 국제 현장 프로그래밍 가능 논리 및 응용 컨퍼런스*, 2009년 8월, 32-37쪽.

[92] C. Farabet, C. Poulet 및 Y. LeCun, "컨볼루션 네트워크를 사용한 임베디드 실시간 비전을 위한 FPGA 기반 스트림 프 로 세서 ", *2009 IEEE 제12회 컴퓨터 비전 워크숍 국제 컨퍼런스, ICCV 워크숍*, 2009년 9월, 878-885페이지.

[93] C. Ding *et al.*, "CirCNN: 블록 순환 가중치 행렬을 이용한 심층 신경망 가속 및 압축," in *Proc. Int. (MICRO)*, 2017, 395-408쪽.

[94] M. Alwani *외*, "Fused-layer CNN 가속기,", *49th Annu. Int. (MICRO)*, Oct 2016, 1-12쪽.

[95] D. 루이스 *외*, "StratixTM10 고집적 파이프라인 FPGA 아키텍텍- ture," in *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Array*, 2016, 159-168쪽.

[96] D. Kim *et al.*, "Neurocube: 고밀도 3D 메모리를 갖춘 프로그래밍 가능한 디지털 뉴로모픽 아키텍처," in *Proc. Int. (ISCA)*, June 2016, 380-392쪽.

[97] A. Shafiee *외*, "ISAAC: 크로스바에서 현장 아날로그 산술이 가능한 컨볼루션 신경망 가속기," *43rd Annu. Int. (ISCA)*, June 2016, 14-26쪽.

[98] S. B. Eryilmaz *외*, "전자 시냅스가 있는 뉴로모픽 아키텍처", *17th Int. 심포지엄,* 2016, 3월, 118-123쪽.

[99] S. K. Esser *외*, "빠르고 에너지 효율적인 뉴로모픽 컴퓨팅을 위한 컨볼루션 네트워크," *CoRR*, vol.

[100] A. Ling과 J. Anderson, "딥 러닝에서 FPGA의 역할", *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, 2017, 3-3쪽.