



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2020-0083123
(43) 공개일자 2020년07월08일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2018.01) G06F 9/345 (2006.01)
(52) CPC특허분류
G06F 9/30043 (2013.01)
G06F 9/30098 (2013.01)
(21) 출원번호 10-2019-0056970
(22) 출원일자 2019년05월15일
심사청구일자 2019년05월15일
(30) 우선권주장
1821300.9 2018년12월31일 영국(GB)

(71) 출원인
그래프코어 리미티드
영국 비에스1 2피에이치 브리스톨 와인 스트리트
11-19
(72) 발명자
알렉산더 알렌 그레이엄
영국 지엘12 8알와이 워튼-언더-에지 킹스우드 턴
데일 뷰 24
노우레스 시몬 크리스티안
영국 비에이2 9에이엔 코스톤 더 바튼 힐 하우스
고어 브루들라
영국 비에이1 3비더블유 바스 세인트 존스 로드
14
(74) 대리인
박장원

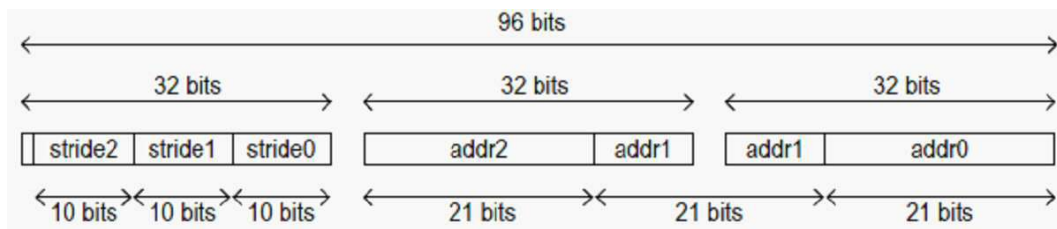
전체 청구항 수 : 총 26 항

(54) 발명의 명칭 로드-저장 명령

(57) 요약

프로세서는 적어도 하나의 레지스터 파일 내의 레지스터들 중에서, 2개의 로드 연산 각각의 대상(목족지), 저장 연산의 각각의 소스 및 3개의 메모리 어드레스를 보유하도록 배열된 한 쌍의 어드레스들을 지정하는 오퍼랜드들을 갖는 로드-저장 명령을 포함하는 명령 세트를 갖는다. 상기 3개의 메모리 어드레스는 2개의 로드 연산 각각에 대한 각각의 로드 어드레스 및 저장 연산에 대한 각각의 저장 어드레스이다. 상기 로드-저장 명령은 2개의 로드 어드레스 및 1개의 저장 어드레스 각각에 대한 각각의 스트라이드 값을 지정하는 3개의 즉치 스트라이드 오퍼랜드를 더 포함하고, 각각의 즉치 스트라이드 오퍼랜드의 적어도 일부 가능한 값은 하나 이상의 레지스터 파일들 중 하나의 스트라이드 레지스터 내의 복수의 필드들 중 하나를 지정함으로써 각 스트라이드 값을 지정하고, 각 필드는 상이한 스트라이드 값을 보유한다.

대표도 - 도7



(52) CPC특허분류
G06F 9/3455 (2013.01)

명세서

청구범위

청구항 1

실행 유닛, 메모리, 및 복수의 레지스터를 포함하는 하나 이상의 레지스터 파일을 포함하는 프로세서로서, 상기 실행 유닛은 각각이 오퍼코드 및 0 이상의 오퍼랜드로 구성된 기계 코드 명령의 타입들을 정의하는 명령 세트로부터 명령들의 인스턴스를 실행하도록 구성되고;

상기 실행 유닛은 로드-저장 유닛을 포함하고, 상기 명령 세트에 정의된 명령의 타입들은 하나 이상의 레지스터 파일 중 적어도 하나 내의 레지스터들 중에서, 2개의 로드 연산 각각의 개별 대상(respective destination), 저장 연산의 개별 소스 및 3개의 메모리 어드레스를 유지하도록 배열된 한 쌍의 어드레스 레지스터를 지정하는 오퍼랜드들을 갖는 로드-저장 명령을 포함하고, 상기 3개의 메모리 어드레스는 2개의 로드 연산 각각에 대한 개별 로드 어드레스 및 상기 저장 연산에 대한 각각의 저장 어드레스이고;

상기 로드-저장 명령은 2개의 로드 어드레스와 하나의 저장 어드레스 각각에 대한 개별 스트라이드(stride) 값을 각각 지정하는 3개의 즉시(immediate) 스트라이드 오퍼랜드를 더 포함하고, 상기 즉시 스트라이드 오퍼랜드 각각의 적어도 일부 가능한 값은 하나 이상의 레지스터 파일 중 하나의 스트라이드 레지스터 내의 복수의 필드 중 하나를 지정함으로써 개별 스트라이드 값을 지정하고, 상기 각 필드는 상이한 스트라이드 값을 유지하고; 그리고

상기 로드-저장 유닛은 로드-저장 명령의 오퍼코드에 응답하여, 상기 2개의 로드 어드레스 각각으로부터 메모리의 데이터의 개별 부분을 개별 로드 연산의 각각의 대상으로 로드하고, 상기 저장 연산의 소스로부터의 데이터의 개별 부분을 메모리의 저장 어드레스에 저장하고, 상기 각 로드 및 저장 연산 다음에 상기 개별 스트라이드 값만큼 각각의 어드레스를 증가시키도록 구성되는 것을 특징으로 하는 프로세서.

청구항 2

제1항에 있어서,

상기 로드-저장 명령은,

상기 하나 이상의 레지스터 파일 중 하나내의 복수의 가능한 레지스터 중에서 스트라이드 레지스터를 지정하기 위한 스트라이드 레지스터 오퍼랜드를 더 포함하는 것을 특징으로 하는 프로세서.

청구항 3

제1항 또는 제2항에 있어서,

상기 스트라이드 오퍼랜드의 하나의 가능한 값은 1 단위(unit)의 스트라이드 값을 지정하고, 상기 스트라이드 오퍼랜드의 복수의 다른 가능한 값은 스트라이드 레지스터 내의 상기 필드들의 상이한 것들을 지정하는 것을 특징으로 하는 프로세서.

청구항 4

제1항 또는 제2항에 있어서,

상기 레지스터 파일 또는 상기 어드레스 레지스터 및 스트라이드 레지스터의 파일들로부터 상기 로드-저장 유닛에 3개의 포트를 포함하고,

상기 로드-저장 유닛은 상기 스트라이드 레지스터에 액세스하기 위해 한 쌍의 어드레스 레지스터 각각 및 상기 포트들 중 하나에 대한 각각의 포트를 사용하도록 구성되는 것을 특징으로 하는 프로세서.

청구항 5

제4항에 있어서,

상기 3개의 각 포트는 액세스하는데 사용되는 각각의 어드레스 레지스터 또는 스트라이드 레지스터의 비트 폭과

동일한 비트 폭을 갖는 것을 특징으로 하는 프로세서.

청구항 6

제1항 또는 제2항에 있어서,

상기 한 쌍의 어드레스 레지스터 각각은 32비트 폭이고, 상기 로드 및 저장 어드레스 각각은 21비트 폭인 것을 특징으로 하는 프로세서.

청구항 7

제1항 또는 제2항에 있어서,

상기 명령 세트에 정의된 명령의 타입들은 하나 이상의 레지스터 파일 중 적어도 하나 내의 레지스터들 중에서, 제1 입력 및 제2 입력을 수신하는 소스들과 결과를 출력할 대상을 지정하는 오퍼랜드들을 취하는 산술 명령을 더 포함하고; 그리고

상기 프로세서는 로드-저장 명령의 인스턴스들과 산술 명령의 인스턴스들을 포함하는 일련의 명령들을 포함하는 프로그램을 실행하도록 프로그래밍되고, 상기 로드-저장 명령들의 적어도 일부의 소스는 산술 명령의 적어도 일부의 대상으로서 설정되고, 상기 로드-저장 명령들의 적어도 일부의 대상은 산술 명령의 적어도 일부의 소스로서 설정되는 것을 특징으로 하는 프로세서.

청구항 8

제7항에 있어서,

상기 일련은 각각의 명령 쌍이 로드-저장 명령의 인스턴스 및 산술 명령의 대응하는 인스턴스로 구성된 일련의 명령 쌍들을 포함하고; 그리고

각 명령 쌍에서, 상기 로드-저장 명령의 소스는 상기 쌍들 중 하나의 선행 쌍에서 산술 명령의 대상으로서 설정되고, 상기 로드-저장 명령의 대상은 상기 쌍들 중 하나의 현재 또는 후속 쌍에서 산술 명령의 소스로서 설정되는 것을 특징으로 하는 프로세서.

청구항 9

제8항에 있어서,

입력 및 결과 각각은 적어도 하나의 부동 소수점 값을 포함하고, 상기 실행 유닛은 산술 명령의 오퍼코드에 응답하여 산술 연산을 수행하도록 구성된 부동 소수점 산술 유닛을 포함하는 것을 특징으로 하는 프로세서.

청구항 10

제8항에 있어서,

상기 산술 명령은,

벡터 내적 명령, 누적 벡터 내적 명령, 행렬 곱 명령, 누적 행렬 곱 명령 또는 컨벌루션 명령인 것을 특징으로 하는 프로세서.

청구항 11

제9항에 있어서,

상기 명령 쌍들 각각은 동시에 실행될 명령 번들이고; 그리고

상기 프로세서는 첫 번째는 각 번들의 로드-저장 명령의 인스턴스를 실행하도록 배열된 로드-저장 유닛을 포함하고, 두 번째는 병렬로 산술 명령의 대응하는 인스턴스를 실행하도록 배열된 부동 소수점 산술 유닛을 포함하는 2개의 병렬 파이프 라인으로 분할되는 것을 특징으로 하는 프로세서.

청구항 12

제8항에 있어서,

상기 산술 명령은 N-엘리먼트 입력 벡터에 각각이 N-엘리먼트 벡터인 M 개의 커널의 $M \times N$ 행렬을 곱하는 누적 행렬 곱 명령이고, 상기 누적 행렬 곱 명령은 N1 개의 연속 페이즈(phase)에서 하나의 페이즈를 지정하는 즉치 오퍼랜드를 더 취하고;

상기 일련의 명령은 루프내에서 반복되는 시퀀스를 포함하고, 각 루프 내의 시퀀스는 상기 명령 쌍들의 N1 개의 시퀀스를 포함하고, 상기 시퀀스 내의 각각의 연속 쌍에서 누적 행렬 곱 명령의 인스턴스는 초기 페이즈에서 N1 번째 페이즈까지 상기 시퀀스의 상이한 연속 페이즈를 지정하는 페이즈 오퍼랜드의 상이한 값을 가지며;

각 페이즈에서 상기 제1 입력은 입력 벡터의 각각의 서브 벡터이고 상기 제2 입력은 하나 이상의 부분 합과 각각의 세트이고, 각 입력 서브 벡터내의 엘리먼트의 수(N_2)는 N/N_1 이고, 각 세트 내의 부분 합 수(N_p)는 M/N_1 이며;

각 루프의 각 페이즈에서의 상기 산술 연산은,

- 상기 누적 행렬 곱 명령의 각각의 인스턴스의 상기 제2 입력으로부터의 N_p 개의 부분 합과 각각의 세트를 다음 루프에서 사용하기 위해 임시 전파기 상태 (temporary propagator state)로 복사하는 단계와;
- 상기 M 개의 커널 각각에 대해, 상기 현재 페이즈의 상기 각각의 입력 서브 벡터와 상기 커널의 각각의 N_2 -엘리먼트 서브 벡터와의 내적을 수행하여, 상기 M 개의 커널 각각에 대한 대응하는 중간 결과를 생성하는 단계와;
- 상기 페이즈 오퍼랜드가 초기 페이즈를 지정하면, 상기 이전 루프로부터의 상기 대응하는 부분 곱을 상기 M 개의 중간 결과 각각에 가산하는 단계와;
- 상기 대응하는 중간 결과를 누산기 상태의 M 개의 엘리먼트 각각에 가산하는 단계와;
- 상기 누적 행렬 곱 명령의 각각의 인스턴스의 대상에 출력 결과로서 현재 또는 이전 루프로부터 상기 누산기 상태의 N_p 개의 엘리먼트의 각각의 서브셋을 출력하는 단계를 포함하는 것을 특징으로 하는 프로세서.

청구항 13

제12항에 있어서,

상기 프로그램은 유용한 출력 결과를 생성하기 전에 상기 루프들 중 적어도 2개의 워밍-업(warm-up) 기간을 포함하는 것을 특징으로 하는 프로세서.

청구항 14

제12항에 있어서,

상기 $M=8$, $N=16$, $N_1=4$, $N_2=4$ 및 $N_p=2$ 인 것을 특징으로 하는 프로세서.

청구항 15

제12항에 있어서,

컨볼루션을 수행하기 위해 루프된 시퀀스를 사용하도록 프로그래밍되고, 각 루프내의 입력 벡터는 입력 데이터의 일부의 상이한 샘플을 나타내고, 이전 루프들의 결과는 이후 루프들의 부분 합으로 사용되는 것을 특징으로 하는 프로세서.

청구항 16

제15항에 있어서,

상기 입력 데이터의 일부는 3D 데이터 볼륨을 포함하고, 상기 M 개의 커널 각각은 M 개의 더 큰 3D 커널들 중 대응하는 하나의 1D 구성 부분을 나타내고, 상기 이전 루프들의 결과는 각 3D 커널과 대응 구성 1D 커널의 입력 데이터와의 컨볼루션을 설정하도록 이후 루프들의 부분 합으로 사용되는 것을 특징으로 하는 프로세서.

청구항 17

제16항에 있어서,

상기 3D 커널들 각각은 신경망에서 노드의 가중치 세트를 나타내는 것을 특징으로 하는 프로세서.

청구항 18

제15항에 있어서,

상기 M 개의 각 값은 입력 데이터 및 상이한 특징의 컨볼루션을 나타내는 것을 특징으로 하는 프로세서.

청구항 19

제12항에 있어서,

상기 입력 벡터의 각 엘리먼트는 16 비트 부동 소수점 값이고, 각 커널의 각 엘리먼트는 16 비트 부동 소수점 값인 것을 특징으로 하는 프로세서.

청구항 20

제12항에 있어서,

상기 N_p 개의 출력 결과 각각은 32 비트 부동 소수점 값인 것을 특징으로 하는 프로세서.

청구항 21

제1항 또는 제2항에 있어서,

상기 레지스터 파일들은 제1 레지스터 파일 및 별개의 제2 레지스터 파일을 포함하고, 상기 어드레스 레지스터들은 제1 레지스터 파일의 레지스터이고, 상기 로드-저장 명령의 소스 및 대상은 제2 레지스터 파일의 레지스터인 것을 특징으로 하는 프로세서.

청구항 22

제21항에 있어서,

상기 스트라이드 레지스터는 제1 파일의 레지스터인 것을 특징으로 하는 프로세서.

청구항 23

제8항에 있어서,

상기 레지스터 파일들은 제1 레지스터 파일 및 별개의 제2 레지스터 파일을 포함하고, 상기 어드레스 레지스터들은 제1 레지스터 파일의 레지스터이고, 상기 로드-저장 명령의 소스 및 대상은 제2 레지스터 파일의 레지스터이고; 그리고

상기 산술 명령의 소스 및 대상은 제2 레지스터 파일의 레지스터인 것을 특징으로 하는 프로세서.

청구항 24

제12항에 있어서,

상기 레지스터 파일들은 제1 레지스터 파일 및 별개의 제2 레지스터 파일을 포함하고, 상기 어드레스 레지스터들은 제1 레지스터 파일의 레지스터이고, 상기 로드-저장 명령의 소스 및 대상은 제2 레지스터 파일의 레지스터들이고; 그리고

상기 레지스터 파일들은 상기 제1 및 제2 레지스터 파일과 별개인 제3 레지스터 파일을 더 포함하고, 상기 커널은 제3 파일에 유지되는 것을 특징으로 하는 프로세서.

청구항 25

임의의 선행하는 항들 중 어느 한 항의 프로세서상에서 실행되도록 구성된 코드를 포함하고, 상기 코드는 로드-저장 명령의 하나 이상의 인스턴스를 포함하는 것을 특징으로 하는 컴퓨터 판독 가능 매체에 구현된 컴퓨터 프로그램.

청구항 26

제1항 내지 제24항 중 어느 한 항에 따라 구성된 프로세서를 동작시키는 방법으로서, 상기 방법은 실행 유닛을

통해 프로세서상에서 로드-저장 명령의 하나 이상의 인스턴스를 포함하는 프로그램을 실행하는 단계를 포함하는 것을 특징으로 하는 방법.

발명의 설명

기술 분야

[0001] 본 발명은 높은 시맨틱 밀도를 갖는 로드-저장 명령의 형태에 관한 것이다.

배경 기술

[0002] 그래픽 처리 장치(GPU) 및 디지털 신호 프로세서(DSP)와 같은 특정 애플리케이션용으로 설계된 프로세서 개발에 대한 관심이 증가하고 있다. 최근 관심을 얻은 다른 타입의 애플리케이션-특정 프로세서는 신청자가 "IPU"(지능형 처리 장치)라고 부르는 기계 지능형 애플리케이션에 전용된 프로세서이다. 이들은 예를 들어, 신경망과 같은 지식 모델을 트레이닝하거나 트레이닝을 돕는 것과 같이, 호스트에 의해 할당된 작업을 수행하도록 또는 그러한 모델에 기초하여 예측 또는 추론을 수행하거나 수행을 돕도록 배열된 가속기 프로세서로서 사용될 수 있다.

[0003] 기계 지능 알고리즘은 다수의 상호 연결된 노드의 그래프로 나타낼 수 있는 "지식 모델"에 대한 반복 업데이트를 기반으로 한다. 각 노드는 입력의 함수를 나타낸다. 일부 노드는 그래프에 대한 입력을 수신하고 일부는 하나 이상의 다른 노드로부터 입력을 수신하는 반면, 일부 노드의 출력은 다른 노드의 입력을 형성하고 일부 노드의 출력은 그래프의 출력을 제공한다(경우에 따라 주어진 노드는 그래프의 입력, 그래프의 출력 및 다른 노드로의 연결 등 모든 것을 가질 수도 있다). 게다가, 각 노드에서의 함수는 하나 이상의 각각의 파라미터, 예를 들어, 가중치에 의해 파라미터화된다. 학습 단계에서 목표는, 경험적 입력 데이터 세트에 기초하여 그래프가 전체적으로 가능 입력 범위에 대해 원하는 출력을 생성하도록 다양한 파라미터에 대한 값을 찾는 것이다. 확률적 기울기 강하에 기초한 역전파 알고리즘과 같이 이것을 행하는 다양한 알고리즘이 당 업계에 공지되어 있다. 입력 데이터를 기반으로 여러 번 반복할 때 오류를 줄이기 위해 파라미터가 점진적으로 조정되고, 그에 따라 그래프는 해로 수렴된다. 후속 단계에서, 학습된 모델은 지정된 입력 세트가 주어진 경우 출력 예측을 하거나 지정된 출력 세트가 주어진 경우 입력(원인)을 추론할 수 있다.

[0004] 기계 지능 애플리케이션을 위해 설계된 프로세서는 기계 지능 애플리케이션에서 공통으로 사용되는 산술 연산을 수행하기 위한 명령 세트에 전용 명령을 포함할 수 있다(명령 세트는 프로세서의 실행 유닛이 인식하도록 구성된 기계 코드 명령 타입의 기본 세트이며, 각각의 타입은 각각의 오퍼코드 및 0이상의 오퍼랜드에 의해 정의된다. 예를 들어, 신경망과 같은 기계 지능 애플리케이션에 필요한 공통 연산은 입력 데이터 세트에 대한 커널의 컨볼루션이고, 여기서 커널은 신경망에서 노드의 가중치를 나타낸다. 데이터에 대해 커다란 크기인 커널의 컨볼루션을 수행하기 위해, 컨볼루션은 후속 곱(products)의 출력으로 누적될 부분 합을 각각 출력하는 다수의 벡터 및 행렬 곱으로 분해될 수 있다. 컨볼루션 수행에 사용하기 위한 벡터 및 행렬 곱셈 타입의 연산을 수행하기 위해 전용되는 산술 명령을 그들의 명령 세트에 포함하고 있는 프로세서가 이미 존재한다.

발명의 내용

[0005] 이러한 종류의 산술 명령은 가산, 승산 등과 같은 더 단순하고 보다 전통적인 타입의 산술 명령과 비교하여 단일 명령에서 많은 양의 시맨틱(의미론적) 콘텐츠를 포함한다(즉, 명령 당 많은 양의 논리를 수행한다). 그러나, 복잡한 명령 또는 연산을 동반하기 위해서는 명령 세트에서, 또한 단일 명령에 고밀도의 시맨틱 정보를 포함하는 로드-저장 명령을 제공하는 것이 바람직할 것이다.

[0006] 본 명세서에 개시된 일 양태에 따르면, 실행 유닛, 메모리, 및 복수의 레지스터를 포함하는 하나 이상의 레지스터 파일을 포함하는 프로세서가 제공되며, 상기 실행 유닛은 각각이 오퍼코드 및 0 이상의 오퍼랜드로 구성된 기계 코드 명령의 타입들을 정의하는 명령 세트로부터 명령들의 인스턴스를 실행하도록 구성된다. 상기 실행 유닛은 로드-저장 유닛을 포함하고, 상기 명령 세트에 정의된 명령의 타입들은 하나 이상의 레지스터 파일 중 적어도 하나 내의 레지스터들 중에서, 2개의 로드 연산 각각의 개별 대상(destination), 저장 연산의 개별 소스 및 3개의 메모리 어드레스를 유지하도록 배열된 한쌍의 어드레스 레지스터를 지정하는 오퍼랜드들을 갖는 로드-저장 명령을 포함하고, 상기 3개의 메모리 어드레스는 2개의 로드 연산 각각에 대한 개별 로드 어드레스 및 상기 저장 연산에 대한 개별 저장 어드레스이다. 상기 로드-저장 명령은 상기 2개의 로드 어드레스와 하나의 저장 어드레스 각각에 대한 개별 스트라이드(stride) 값을 각각 지정하는 3개의 즉시(immediate) 스트라이드 오퍼랜드를 더 포함하고, 상기 즉시 스트라이드 오퍼랜드 각각의 적어도 일부 가능한 값은 하나 이상의 레지스터 파일을

중 하나의 스트라이드 레지스터 내의 복수의 필드 중 하나를 지정함으로써 개별 스트라이드 값을 지정하고, 상기 각 필드는 상이한 스트라이드 값을 유지한다. 상기 로드-저장 유닛은 로드-저장 명령의 오퍼코드에 응답하여, 상기 2개의 로드 어드레스 각각으로부터 상기 메모리의 데이터의 개별 부분을 상기 개별 로드 연산의 각각의 대상으로 로드하고, 상기 저장 연산의 소스로부터 데이터의 각각의 부분을 상기 메모리의 저장 어드레스에 저장하고, 상기 각 로드 및 저장 연산 다음에 상기 개별 스트라이드 값만큼 상기 각각의 어드레스를 증가시키도록 구성된다(즉, 로드-저장 유닛은 각 즉치 스트라이드 오퍼랜드의 적어도 일부 가능한 값들에 대해, 스트라이드 레지스터 내의 복수의 필드 중 하나로부터 각각의 스트라이드 값을 취하도록 구성되고, 상기 필드는 즉치 스트라이드 오퍼랜드의 값에 의해 지정된다).

[0007] 이 로드-스토어 명령은 유리하게도 메모리로부터 입력 데이터를 스트리밍하고 결과를 낮은 코드 오버 헤드(높은 코드 밀도)로 고속으로 메모리로 다시 스트리밍할 수 있게 한다. 예를 들어, 이는 산술 연산에 의해 소비되고 출력되는 것과 동일한 속도로 산술 연산의 입력 및 출력을 메모리에 스트리밍하는데 사용되며, 이러한 목적을 위해 요구되는 명령의 오버 헤드는 상대적으로 낮다. 이것은 예를 들어 행렬 곱셈을 동반하는 기계 학습 애플리케이션 또는 입력 데이터와 가중치의 컨볼루션을 수행하는데 사용되는 행렬 곱셈 명령을 누적하는데 유용할 수 있다.

[0008] 그러나, 개시된 로드-저장 명령의 적용 가능성은 이러한 애플리케이션에 국한되지는 않는다. 이러한 시맨틱 컨텍스트를 갖는 로드-저장 타입 명령은 또한 복합 산술 명령의 다른 시퀀스에서도 같이 기계 학습, 컨볼루션 수행 또는 행렬 곱셈 이외의 애플리케이션을 탐색할 수 있다.

[0009] 실시예에서, 로드-저장 명령은 하나 이상의 레지스터 파일들 중 하나의 레지스터 내의 복수의 가능한 레지스터들 중에서 스트라이드 레지스터를 지정하기 위한 스트라이드 레지스터 오퍼랜드를 더 포함할 수 있다.

[0010] 실시예에서, 스트라이드 오퍼랜드의 하나의 가능한 값은 1 단위(unit)의 스트라이드 값을 지정하고, 상기 스트라이드 오퍼랜드의 복수의 다른 가능한 값은 스트라이드 레지스터 내의 상기 필드들의 상이한 것들을 지정할 수 있다.

[0011] 즉, 로드 저장 유닛은 각 즉치 스트라이드 오퍼랜드의 적어도 일부 가능한 값들에 대해, 스트라이드 레지스터의 필드로부터 스트라이드 값을 취하기보다는 각 스트라이드 값을 1 단위로 설정하도록 구성된다(여기서, 1 단위는 1 원자, 즉 반드시 1 바이트가 아닌 데이터 액세스의 크기, 예를 들어, 오퍼랜드가 16 비트 부동 소수점 값의 4-엘리먼트 벡터인 경우 증분은 1 원자/단위이고, 이는 8 바이트, 64 비트와 동일하다. 일부 또는 모든 다른 경우, 즉치 스트라이드 오퍼랜드의 값에 의해 지정된 대로 스트라이드 레지스터의 필드들 중 하나로부터 각각의 스트라이드 값을 취한다.

[0012] 실시예에서, 프로세서는 레지스터 파일 또는 상기 어드레스 레지스터 및 스트라이드 레지스터의 파일들로부터 로드-저장 유닛에 3개의 포트를 포함할 수 있고, 상기 로드-저장 유닛은 스트라이드 레지스터에 액세스하기 위해 상기 한 쌍의 어드레스 레지스터 각각 및 상기 포트들 중 하나에 대한 각각의 포트를 사용하도록 구성될 수 있다.

[0013] 실시예에서, 상기 3개의 각 포트는 액세스하는데 사용되는 각각의 어드레스 레지스터 또는 스트라이드 레지스터의 비트 폭과 동일한 비트 폭을 갖는다.

[0014] 실시예에서, 프로세서는 레지스터 파일 또는 문제의 파일(다중 스레드 프로세서의 경우에 컨텍스트간에 공유된)로부터 로드-저장 유닛에 이들 3개의 포트만을 포함한다. 실시예에서, 한 쌍의 어드레스 레지스터 및 스트라이드 레지스터는 동일한 레지스터 파일에 있을 수 있다. 실시예에서, 어드레스 및 스트라이드 레지스터는 모두 동일한 비트 폭을 가질 수 있다. 실시예에서, 이 폭은 32 비트일 수 있다.

[0015] 실시예에서, 한 쌍의 어드레스 레지스터 각각은 32비트 폭일 수 있고, 상기 로드 및 저장 어드레스 각각은 21비트 폭일 수 있다.

[0016] 실시예에서, 각각의 로드의 목적지는 한 쌍의 32 비트 폭 레지스터일 수 있고, 저장의 소스는 한 쌍의 32 비트 폭 레지스터일 수 있다.

[0017] 실시예에서, 상기 프로세서는 메모리로부터의 로딩을 수행하는 2 개의 포트 및 메모리로의 저장을 수행하기 위한 하나의 포트를 포함할 수 있다. 실시예에서, 프로세서는 메모리로부터 로딩하기 위한 이들 2 개의 포트 및 메모리(멀티 스레드 프로세서의 경우에서 컨텍스트들 사이에서 공유된)에 저장하기 위한 이 하나의 포트만을 포함할 수 있다. 실시예에서, 이들 포트 각각은 64 비트 폭이다. 실시예에서, 상기 로드들 각각에 의해 로딩된 데

이터의 부분은 64 비트 폭이고, 상기 저장에 의해 저장된 데이터의 부분은 64 비트 폭이다.

- [0018] 실시예에서, 상기 로드들 각각에 의해 로딩된 데이터의 부분은 한 쌍의 32 비트 부동 소수점 값들 또는 16 비트 부동 소수점 값들의 4-엘리먼트 벡터로 구성될 수 있고, 상기 저장에 의해 저장된 데이터의 부분은 한 쌍의 32 비트 부동 소수점 값으로 구성될 수 있다.
- [0019] 실시예에서, 상기 명령 세트에 정의된 명령의 타입들은 하나 이상의 레지스터 파일 중 적어도 하나 내의 레지스터들 중에서, 제1 입력 및 제2 입력을 수신하는 소스들과 결과를 출력할 대상을 지정하는 오퍼랜드들을 취하는 산술 명령을 더 포함할 수 있다. 상기 프로세서는 로드-저장 명령의 인스턴스들과 상기 산술 명령의 인스턴스들을 포함하는 일련의 명령들을 포함하는 프로그램을 실행하도록 프로그래밍되고, 상기 로드-저장 명령들의 적어도 일부의 소스는 산술 명령의 적어도 일부의 대상으로서 설정되고, 상기 로드-저장 명령들의 적어도 일부의 대상은 산술 명령의 적어도 일부의 소스로서 설정될 수 있다.
- [0020] 실시예에서, 상기 일련은 일련의 명령 쌍들을 포함할 수 있으며, 각 명령 쌍은 로드-저장 명령의 인스턴스 및 산술 명령의 대응하는 인스턴스로 구성된다. 각 명령 쌍에서, 로드-저장 명령의 소스는 상기 쌍들 중 하나의 선행 쌍에서 산술 명령의 대상으로서 설정되고, 로드-저장 명령의 대상은 상기 쌍들 중 하나의 현재 또는 후속 쌍에서 산술 명령의 소스로서 설정될 수 있다.
- [0021] 실시예에서, 상기 입력 및 상기 결과 각각은 적어도 하나의 부동 소수점 값을 포함할 수 있으며, 상기 실행 유닛은 산술 명령의 오퍼코드에 응답하여 상기 산술 연산을 수행하도록 구성된 부동 소수점 산술 유닛을 포함한다.
- [0022] 실시예에서, 상기 산술 명령은 벡터 내적 명령, 누적 벡터 내적 명령, 행렬 곱 명령, 누적 행렬 곱 명령 또는 컨벌루션 명령 중 하나일 수 있다.
- [0023] 실시예에서, 상기 명령 쌍들 각각은 동시에 실행될 명령 번들일 수 있으며, 상기 프로세서는 첫 번째는 각 번들의 로드-저장 명령의 인스턴스를 실행하도록 배열된 로드-저장 유닛을 포함하고, 두 번째는 병렬로 산술 명령의 대응하는 인스턴스를 실행하도록 배열된 부동 소수점 산술 유닛을 포함하는 2개의 병렬 파이프 라인으로 분할될 수 있다.
- [0024] 실시예에서, 상기 산술 명령은 N-엘리먼트 입력 벡터에 각각이 N-엘리먼트 벡터인 M 개의 커널의 M×N 행렬을 곱하는 누적 행렬 곱 명령일 수 있으며, 상기 누적 행렬 곱 명령은 N1 개의 연속 페이즈(phase)에서 하나의 페이즈를 지정하는 즉치 오퍼랜드를 더 취한다. 이 경우에, 상기 일련의 명령은 루프내에서 반복되는 시퀀스를 포함할 수 있으며, 각 루프 내의 시퀀스는 상기 명령 쌍들의 N1 개의 시퀀스를 포함하고, 상기 시퀀스 내의 각각의 연속 쌍에서 누적 행렬 곱 명령의 인스턴스는 초기 페이즈에서 N1 번째 페이즈까지 상기 시퀀스의 상이한 연속 페이즈를 지정하는 페이즈 오퍼랜드의 상이한 값을 가진다. 각 페이즈에서 제1 입력은 입력 벡터의 각각의 서브 벡터이고 제2 입력은 하나 이상의 부분 합의 각각의 세트이고, 각 입력 서브 벡터내의 엘리먼트의 수(N2)는 N/N1이고, 각 세트 내의 부분 합의 수(Np)는 M/N1이다. 이러한 실시예에서, 각 루프의 각 페이즈에서의 상기 산술 연산은,
- [0025] - 상기 누적 행렬 곱 명령의 각각의 인스턴스의 상기 제2 입력으로부터의 Np개의 부분 합의 각각의 세트를 다음 루프에서 사용하기 위해 임시 전파기 상태 (temporary propagator state)로 복사하는 단계와;
- [0026] - 상기 M 개의 커널 각각에 대해, 상기 현재 페이즈의 상기 각각의 입력 서브 벡터와 상기 커널의 각각의 N2-엘리먼트 서브 벡터와의 내적을 수행하여, 상기 M 개의 커널 각각에 대한 대응하는 중간 결과를 생성하는 단계와;
- [0027] - 상기 페이즈 오퍼랜드가 초기 페이즈를 지정하면, 상기 이전 루프로부터의 상기 대응하는 부분 곱을 상기 M 개의 중간 결과 각각에 가산하는 단계와;
- [0028] - 상기 대응하는 중간 결과를 누산기 상태의 M 개의 엘리먼트 각각에 가산하는 단계와;
- [0029] - 상기 누적 행렬 곱 명령의 각각의 인스턴스의 대상에 출력 결과로서 현재 또는 이전 루프로부터 상기 누산기 상태의 Np 개의 엘리먼트의 각각의 서브셋을 출력하는 단계를 포함할 수 있다.
- [0030] 따라서, 실시예들은 루프 당 M개의 명령들만을 사용하여 동일한 속도(rate)로 메모리로 입력 및 출력을 스트리밍하는 동안 루프 당 한번, N-엘리먼트 입력 벡터와 M×N 행렬의 곱셈을 가능하게 한다. 예를 들어, 실시예들에서 M = 8, N = 16, N1 = 4, N2 = 4 및 Np = 2이다.
- [0031] 상기 전파 상태 및 누산기 상태는 산술 유닛의 논리의 내부 상태일 수도 있고, 또는 레지스터들 파일 중 하나의

레지스터 또는 조합으로 구현될 수도 있다.

- [0032] 실시예에서, 상기 프로그램은 유용한 출력 결과를 생성하기 전에 상기 루프들 중 적어도 2개의 워밍-업(warm-up) 기간을 포함할 수 있다.
- [0033] 실시예에서, 상기 프로세서는 컨볼루션을 수행하기 위해 루프된 시퀀스를 사용하도록 프로그래밍될 수 있다. 이 경우, 각 루프내의 입력 벡터는 입력 데이터의 일부의 상이한 샘플을 나타내고, 이전 루프들의 결과는 이후 루프들의 부분 합으로 사용될 수 있다.
- [0034] 실시예에서, 상기 입력 데이터의 일부는 3D 데이터 볼륨을 포함하고, 상기 M 개의 커널 각각은 M 개의 더 큰 3D 커널들 중 대응하는 하나의 1D 구성 부분을 나타낼 수 있다. 이 경우, 이전 루프들의 결과는 각 3D 커널과 대응하는 구성 1D 커널의 입력 데이터와의 컨볼루션을 설정하도록 이후 루프들의 부분 합으로 사용될 수 있다.
- [0035] 실시예에서, 입력 데이터의 일부는 3D 데이터 볼륨을 포함할 수 있고, 상기 M 개의 커널 각각은 M 개의 더 큰 3D 커널들 중 대응하는 하나의 1D 구성 부분을 나타낼 수 있다. 이 경우, 이전 루프들의 결과는 이후 루프들의 부분 합으로 사용될 수 있다.
- [0036] 실시예에서, 상기 3D 커널들 각각은 신경망에서 노드의 가중치 세트를 나타낼 수 있다.
- [0037] 실시예에서, 상기 M 개의 각 값은 입력 데이터 및 상이한 특징의 컨볼루션을 나타낼 수 있다.
- [0038] 실시예에서, 상기 입력 벡터의 각 엘리먼트는 16 비트 부동 소수점 값이고, 각 커널의 각 엘리먼트는 16 비트 부동 소수점 값이다.
- [0039] 대안적인 실시예에서, 상기 입력 벡터의 각 엘리먼트는 예를 들어 32 비트 부동 소수점 값일 수 있고, 각 커널의 각 엘리먼트는 32 비트 부동 소수점 값일 수 있다.
- [0040] 실시예에서, 상기 N_p 개의 출력 결과 각각은 32 비트 부동 소수점 값일 수 있다.
- [0041] 대안적인 실시예에서, 상기 출력 결과는 예를 들어 16 비트 부동 소수점 값일 수 있다.
- [0042] 실시예에서, 상기 레지스터 파일들은 제1 레지스터 파일 및 별개의 제2 레지스터 파일을 포함할 수 있고, 상기 어드레스 레지스터들은 제1 레지스터 파일의 레지스터이고, 상기 로드-저장 명령의 소스 및 대상은 제2 레지스터 파일의 레지스터이다.
- [0043] 실시예에서, 상기 스트라이드 레지스터는 제1 파일의 레지스터일 수 있다.
- [0044] 실시예에서, 상기 산술 명령어의 소스 및 대상은 제 2 레지스터 파일의 레지스터일 수 있다.
- [0045] 실시예에서, 상기 프로세서는 제1 레지스터 파일로부터 로드-저장 유닛으로의 3개의 포트 및 제2 레지스터 파일로부터 로드-저장 유닛으로의 하나의 포트를 포함할 수 있다. 실시예에서, 제1 레지스터 파일로부터 로드-저장 유닛으로의 이들 3 개의 포트 각각은 32 비트 폭일 수 있다. 실시예에서, 제2 레지스터 파일로부터 로드-저장 유닛으로의 하나의 포트는 64 비트 폭일 수 있다. 실시예에서, 상기 프로세서는 제1 레지스터 파일로부터 로드-저장 유닛으로의 이들 3 개의 포트들만을 포함할 수 있고, 제2 레지스터 파일로부터 로드-저장 유닛으로의 이 하나의 포트만을 포함할 수 있다.
- [0046] 실시예에서, 상기 프로세서는 제2 레지스터 파일로부터 산술 유닛으로의 2 개의 포트를 포함할 수 있다. 실시예에서, 이들 2 개의 포트 각각은 64 비트 폭일 수 있다. 실시예에서, 상기 프로세서는 제2 레지스터 파일로부터 산술 유닛으로의 이들 2 개의 포트들만을 포함할 수 있다(다중 스레드 프로세서의 경우 모든 컨텍스트들 사이에서 공유되는 2 개의 포트만을 의미함).
- [0047] 위에서 설명한 파라미터들을 사용하면, 각 로드-저장 명령은 2 개의 32 비트 부분 합과 커플(couplet) 당 입력 벡터의 4 개의 16 비트 엘리먼트를 로드할 수 있으므로, 4개 커플의 시퀀스의 루프 당 입력 벡터의 8 개의 32 비트 부분 합과 16 개의 16 비트 엘리먼트를 로드할 수 있다(루프 당 4 단). 각 로드-저장 명령은 또한 커플 당 두 개의 32 비트 부분 합을 저장할 수 있으며, 따라서 4 개 커플의 시퀀스의 루프 당 8 개의 32 비트 부분 합 출력을 저장할 수 있다. 이렇게 하면 4 개의 로드-저장 및 4 개의 산술 명령 인스턴스의 코드 오버 헤드만으로, 상기 대응하는 속도로 메모리에서 입력 및 출력을 스트리밍하면서 루프 당 한 번씩 8×16 행렬의 16-엘리먼트 입력 벡터를 수행할 수 있다.
- [0048] 실시예에서, 상기 레지스터 파일들은 제1 및 제2 레지스터 파일과 별개인 제3 레지스터 파일을 더 포함할 수

있고, 상기 커널들은 제3파일에 유지될 수 있다.

[0049] 상기 제1 레지스터 파일은 스트라이드와 같은 메모리 어드레스 및 어드레스 오프셋을 보유하는 메모리 어드레스 레지스터 파일일 수 있다. 상기 제2 레지스터 파일은 산술 연산의 입력 및 출력 오퍼랜드를 보유하는 산술 레지스터 파일일 수 있다. 상기 제3 레지스터 파일은 가중치, 예를 들어 신경망을 유지하는 가중치 레지스터 파일일 수 있다.

[0050] 본 발명의 다른 양태에 따르면, 상기 실시예 또는 본 명세서의 임의의 실시예의 프로세서상에서 실행되도록 구성된 코드를 포함하는 컴퓨터 판독 가능 스토리지 상에 구현된 컴퓨터 프로그램이 제공되며, 상기 코드는 상기 로드-저장 명령의 하나 이상의 인스턴스를 포함한다.

[0051] 실시예에서, 상기 프로그램은 본 명세서에 개시된 임의의 구성들에 따라 동작되도록 더 구성될 수 있다. 예를 들어, 실시예들에서, 상기 프로그램은 산술 명령의 하나 이상의 인스턴스들을 포함할 수 있다. 상기 프로그램은 로드-저장 명령의 인스턴스들에 산재된 산술 명령의 인스턴스들을 포함할 수 있다. 상기 프로그램은 산술 한 일련의 명령, 예를 들어 루핑 시퀀스를 포함할 수 있다.

[0052] 본 명세서에 개시된 또 다른 양태에 따르면, 상기 또는 본 명세서의 임의의 실시예에 따라 구성된 프로세서를 동작시키는 방법이 제공되며, 상기 방법은 실행 유닛을 통해 상기 프로세서상에서 로드-저장 명령의 하나 이상의 인스턴스를 포함하는 프로그램을 실행하는 단계를 포함한다.

도면의 간단한 설명

[0053] 본 발명의 실시예의 이해를 돕고 그러한 실시예의 실시 방법을 보여주기 위해, 단지 예로서 첨부된 도면을 참조한다.

도 1은 예시적인 멀티-스레드 프로세서의 개략적인 블록도이다.

도 2는 인터리빙된 타임 슬롯의 방식을 개략적으로 도시한다.

도 3은 슈퍼바이저 스레드 및 복수의 인터리빙된 타임 슬롯에서 동작하는 복수의 워커 스레드를 개략적으로 도시한다.

도 4는 예시적인 프로세서의 논리 블록 구조를 개략적으로 도시한다.

도 5는 구성 프로세서들의 어레이를 포함하는 프로세서의 개략적인 블록도이다.

도 6은 기계 지능 알고리즘에 사용된 그래프의 개략도이다.

도 7은 하나의 타입의 로드-저장 명령을 구현하는데 사용하기 위한 어드레스 패킹 방식을 개략적으로 도시한다.

도 8은 한 세트의 스트라이드 레지스터들 내의 사전 결정된 스트라이드 값들의 배열을 개략적으로 도시한다.

도 9는 데이터의 입력 볼륨을 갖는 3D 커널(K)의 컨볼루션을 개략적으로 도시한다.

도 10은 누적 행렬 곱셈 명령의 페이즈(페이즈)들의 시퀀스에 의해 수행되는 행렬 곱셈을 개략적으로 도시한다.

도 11은 누적 행렬 곱셈 명령의 연산을 더 도시한다.

도 12는 컨볼루션을 수행하도록 배열된 누산 행렬 곱셈 명령의 시퀀스의 일련의 루프의 예를 제공한다.

도 13은 컨볼루션 명령의 연산을 개략적으로 도시한다.

도 14는 일련의 컨볼루션 명령의 예를 보여 준다.

발명을 실시하기 위한 구체적인 내용

[0054] 도 1은 본 발명의 실시예에 따른 프로세서(4)의 예를 도시한다. 프로세서(4)는 배럴-스레드 처리 유닛의 형태인 멀티-스레드 처리 유닛(10) 및 로컬 메모리(11)(즉, 멀티-타일 어레이의 경우에는 동일한 타일상에 또는 단일-프로세서 칩의 경우 동일한 칩상)을 포함한다. 배럴-스레드 처리 유닛은 파이프 라인의 실행 시간이 인터리빙된 타임 슬롯의 반복 시퀀스로 분할되는 일 타입의 멀티-스레드 처리 유닛으로, 각 스레드는 주어진 스레드에 의해 점유될 수 있다. 이는 또한 곧 더 자세히 논의되는 동시 실행이라고 지칭될 수 있다. 메모리(11)는 (상기한 어드레스 가능 메모리 유닛들 또는 동일한 어드레스 가능 메모리 유닛의 상이한 영역들에 구현될 수 있는)

명령 메모리(12) 및 데이터 메모리(22)를 포함한다. 명령 메모리 (12)는 처리 유닛(10)에 의해 실행되는 기계 코드를 저장하고, 데이터 메모리 (22)는 실행된 코드에 의해 연산될 데이터 및 실행된 코드에 의해 출력된 데이터(예를 들어, 그러한 연산의 결과로서) 모두를 저장한다.

[0055] 메모리(12)는 프로그램의 복수의 상이한 스레드를 저장하고, 각 스레드는 특정 태스크(작업) 또는 태스크들을 수행하기 위한 명령들의 각각의 시퀀스를 포함한다. 본 명세서에서 참조되는 명령들은 기계 코드 명령, 즉 단일 오퍼코드 및 0 이상의 오퍼랜드(피연산자)로 이루어진 프로세서 명령 세트의 기본 명령들 중 하나의 인스턴스를 의미한다. 실시예에서, 프로그램은 복수의 워커(worker) 스레드 및 하나 이상의 수퍼바이저(supervisor) 스레드로서 구조화될 수 있는 수퍼바이저 서브 프로그램을 포함한다. 이것들은 곧 더 자세히 논의될 것이다.

[0056] 멀티-스레드 프로세서는 일반적으로 동시 방식으로 서로 나란히 다수의 프로그램 스레드를 실행할 수 있는 프로세서이다. 동시 실행은 스레드들이 공통 실행 파이프 라인(또는 적어도 파이프 라인의 공통 부분)을 공유하고, 반복 사이클에서 상이한 인터리빙된 타임 슬롯에서 이 동일한 공유 실행 파이프 라인을 통해 상이한 스레드들이 인터리빙되는 것을 의미한다. 이는 파이프 라인 지연 시간을 숨길 기회를 증가시켜 성능이 향상된다. 프로세서는 다수의 상이한 스레드(예를 들어, 공통 명령 메모리, 데이터 메모리 및/또는 실행 유닛)에 공통인 일부 하드웨어를 포함하지만, 멀티-스레딩을 제공하기 위해 프로세서는 또한 각 스레드에 특정한 전용 하드웨어를 포함한다.

[0057] 전용 하드웨어는 동시적으로 실행될 수 있는 적어도 스레드 각각, 즉 사이클에서 슬롯 당 하나의 세트에 대한 컨텍스트 레지스터(26)들의 개별 세트를 포함한다. "컨텍스트"는, 멀티-스레드 프로세서에 대해 이야기할 때, 서로 나란히 실행되는 스레드들 각각의 프로그램 상태(예를 들어, 프로그램 카운터 값, 상태 및 현재 오퍼랜드 값을 지칭한다. 컨텍스트 레지스터는 각각의 스레드의 이 프로그램 상태를 나타내는 각각의 레지스터를 지칭한다. 레지스터 파일의 레지스터들은 레지스터 어드레스는 명령 워드들의 비트로 고정되는 반면 메모리 어드레스는 명령을 실행함으로써 계산될 수 있다는 점에서 일반 메모리와 구별된다.

[0058] 처리 유닛(10) 내에서, 명령 메모리(12)로부터의 스레드들 중 다수의 상이한 스레드는 (일반적으로, 명령 메모리에 저장된 전체 스레드들의 서브 세트만이 전체 프로그램의 특정 지점에서 인터리빙될 수 있지만) 단일 실행 파이프 라인(13)을 통해 인터리빙될 수 있다. 멀티 스레드 처리 유닛(10)은 컨텍스트 레지스터(26)의 복수의 세트를 포함하며, 각 세트(26)는 동시에 실행될 수 있는 상이한 스레드 각각의 상태(컨텍스트)를 나타내기 위해 배치된다. 멀티 스레드 처리 유닛(10)은 또한 동시 실행된 스레드에 공통인 공유(된) 실행 파이프 라인(13) 및 인터리빙 방식(예를 들어, 라운드 로빈 방식)으로 공유 파이프 라인을 통한 실행을 위해 동시 스레드들을 스케줄링하는 스케줄러(24)를 포함한다. 처리 유닛(10)은 복수의 스레드에 공통인 공유 명령 메모리(12)에 연결되고, 복수의 스레드에 다시 공통인 공유 데이터 메모리(22)에 연결된다.

[0059] 실행 파이프 라인(13)은 페치 단계(stage)(14), 디코드 단계(16) 및 상기 명령 세트 아키텍처에 의해 정의된 바와같이 산술 및 논리 연산, 어드레스 계산, 로드 및 저장 연산 및 다른 연산을 수행할 수 있는 실행 유닛을 포함하는 실행 단계(18)를 포함한다.

[0060] 컨텍스트 레지스터들(26)의 각 세트는 (스레드가 현재 실행중인 명령 어드레스를 추적하기 위해) 각각의 스레드에 대한 적어도 하나의 프로그램 카운터(PC) 및 실시예들에서 각각의 스레드의 현재 상태(현재 실행 중인지 또는 일시 중지 중인 여부와 같은)를 기록하는 하나 이상의 제어 상태 레지스터(CSR)의 세트를 포함하는 각각의 하나 이상의 제어 레지스터를 포함한다. 컨텍스트 레지스터 파일들(26)의 각 세트는 또한 각각의 스레드에 의해 실행되는 명령들의 오퍼랜드들, 즉 실행될 때 각 스레드의 명령의 오퍼코드에 의해 정의된 연산에서 연산되거나 연산 결과인 값들을 일시적으로 유지하기 위한 오퍼랜드 레지스터들의 각각의 세트를 포함한다. 레지스터들(26)의 각 세트는 하나 이상의 레지스터 파일로 구현될 수 있다.

[0061] 페치 단계(14)는 각각의 컨텍스트의 프로그램 카운터(PC)에 액세스한다. 각 개별 스레드에 대해, 페치 단계(14)는 프로그램 카운터에 의해 지시된 바와 같이 프로그램 메모리(12)의 다음 어드레스로부터 그 스레드의 다음 명령을 페치한다. 프로그램 카운터는 분기 명령에 의해 분기하지 않는 한 각 실행 사이클을 자동으로 증가시킨다. 그런 다음, 페치 단계(14)는 페치된 명령을 디코드 단계(16)에 전달하여 디코딩되도록 하고, 디코드 단계(16)는 디코딩된 명령의 표시를, 상기 명령이 실행되도록 하기 위해 상기 명령에 특정된 임의의 오퍼랜드 레지스터들의 디코딩된 어드레스와 함께 실행 유닛(18)으로 전달한다. 상기 실행 유닛(18)은 오퍼랜드 레지스터들 및 제어 상태 레지스터들에 대한 액세스를 가지는데, 이는 산술 명령의 경우와 같이(예를 들어, 2개의 오퍼랜드 레지스터의 값을 가산, 승산, 감산 및 제산하고 그 결과를 각각의 스레드의 다른 오퍼랜드 레지스터에 출력함으로써), 상기 디코딩된 레지스터 어드레스들에 기초하여 명령을 실행하는데 사용할 수 있다. 또는 상기 명령이

메모리 액세스(로드 또는 저장)를 정의하는 경우, 실행 유닛(18)의 로드/저장 로직은 데이터 메모리로부터의 값을 각 스레드의 오퍼랜드 레지스터에 로딩하거나 또는 상기 명령에 따라 각 스레드의 오퍼랜드 레지스터로부터의 값을 데이터 메모리(22)에 저장한다.

[0062] 페치 단계(14)는 스케줄러(24)의 제어하에, 명령 메모리(12)로부터 실행될 명령을 페치하도록 접속된다. 스케줄러(24)는 차례로 타임 슬롯의 반분 시퀀스에서 동시에 실행되는 스레드 세트 각각으로부터 명령을 페치하도록 페치 단계(14)를 제어하여, 파이프 라인(13)의 자원을 이하 더 자세히 논의되는 바와같이, 복수의 시간적으로 인터리빙된 타임 슬롯으로 분할한다. 예를 들어, 스케줄링 방식은 라운드 로빈 또는 가중 라운드 로빈(weighted round-robin)일 수 있다. 이와 같은 방식으로 동작하는 프로세서의 다른 용어는 배럴 스레드 프로세서이다.

[0063] 스케줄러(24)에 의해 구현된 인터리빙 방식의 일 예가 도 2에 도시되어 있다. 여기서, 동시 스레드들은 라운드-로빈 방식에 따라 인터리빙되어, 상기 방식의 각 라운드 내에서, 상기 라운드는 타임 슬롯(S0, S1, S2...SJ-1)(예를 들어, J=4 또는 J=6)의 시퀀스로 분할되고, 각각의 슬롯은 각각의 스레드를 실행한다. 통상적으로, 각 슬롯은 하나의 실행 사이클 길이이고, 상이한 슬롯은 모든 가능한 실시예에서 반드시 그런 것은 아니지만 균일하게 크기화되고, 예를 들어 가중 라운드 로빈 방식도 가능하므로 일부 스레드는 실행 라운드마다 다른 스레드보다 더 많은 사이클을 얻는다. 일반적으로, 배럴-스레딩은 짝수 라운드 로빈 또는 가중 라운드 로빈 스케줄을 채택할 수 있으며, 후자의 경우, 상기 가중은 고정적이거나 적응적일 수 있다.

[0064] 실행 라운드마다의 시퀀스가 무엇이든, 이 패턴은 반복되고, 각 라운드는 각각의 타임 슬롯의 각각의 인스턴스를 포함한다. 따라서, 본 명세서에서 언급된 타임 슬롯은 시퀀스의 주어진 반복에서의 타임 슬롯의 특정 인스턴스가 아니라, 시퀀스내의 반복 할당된 장소(place)를 의미한다. 달리 말하면, 스케줄러(24)는 파이프 라인(13)의 실행 사이클을, 각각이 타임 슬롯의 반복 시퀀스에서 각각의 타임 슬롯의 반복을 포함하는 복수의 시간 인터리빙된(시분할 다중화된) 실행 채널로 배분한다. 도시된 실시예에서, 4개의 타임 슬롯이 있지만, 이것은 단지 설명의 목적을 위한 것이며, 다른 숫자도 가능하다. 예: 하나의 바람직한 실시예에서는 실제로 6개의 타임 슬롯이 존재한다.

[0065] 실시예에서, 컨텍스트 레지스터들(26)은 동시에 실행될 수 있는 스레드의 수(J)(도시된 예에서는 J=3이지만 제한적이지는 않음) 각각에 대한 워커 컨텍스트 레지스터(CX0...CX(J-1))의 각각의 세트 및 하나의 추가 슈퍼바이저 컨텍스트 레지스터 파일(CXS)을 포함한다. 워커 컨텍스트 레지스터 파일들은 워커 스레드의 컨텍스트를 보유하고, 슈퍼바이저 컨텍스트 레지스터 파일은 슈퍼바이저 스레드의 컨텍스트를 보유한다. 실시예에서, 슈퍼바이저 컨텍스트는 각각의 워커와 상이한 수의 레지스터를 갖는다. 따라서, 처리 유닛(10)은 타임 슬롯보다 하나 이상의 컨텍스트 레지스터 파일(26)을 포함한다. 즉, 배럴-스레딩이 가능한 인터리빙된 타임 슬롯의 수보다 하나 더 많은 컨텍스트를 지원한다.

[0066] 워커 컨텍스트들(CX0...CXJ) 각각은, 프로그래머에 의해 요구되는 애플리케이션-특정 계산 작업을 수행하기 위해 4개의 실행 타임 슬롯(S0...SJ) 중 하나에 현재 할당된 복수의 워커 스레드 각각의 상태를 나타내는데 사용된다(다시 말하면 이것은 명령 메모리(12)에 저장된 바와같이 프로그램의 총수의 워커 스레드의 서브 세트일 수 있다). 추가 컨텍스트(CXS)는 워커 스레드의 실행을 조정하는 역할을 하는 "슈퍼바이저 스레드"(SV)의 상태를 나타내는데 사용되며, 적어도 워커 스레드들(W) 중 어느 것이 전체 프로그램의 어느 시점에서 상기 타임 슬롯들(S0, S1, S2,...)중 어느 타임 슬롯에 실행될지를 할당하는 의미로 사용된다. 선택적으로 슈퍼바이저 스레드는 외부 교환이나 배리어 동기화 수행과 같은 다른 "관리자(overseer)" 또는 조정 책임을 가질 수 있다. 물론 예시된 J=4의 경우는 설명의 목적을 위한 하나의 예시적인 구현일 뿐이라는 점에 유의해야 한다. 예: 다른 구현예에서, J=6(6개의 타임 슬롯, 6개의 워커 컨텍스트 및 하나의 슈퍼바이저 컨텍스트).

[0067] 도 3을 참조하면, 실시예들에서, 슈퍼바이저 스레드(SV)는 인터리빙된 실행 타임 슬롯의 방식에서 자신의 타임 슬롯 자체를 가지지 않는다. 또한 워커 스레드에 슬롯을 할당하는 워커도 유연하게 정의되지 않는다. 오히려, 각 타임 슬롯은 워커 컨텍스트를 저장하기 위한 컨텍스트 레지스터 세트를 가지며, 이는 슬롯이 워커에게 할당될 때 워커에 의해 사용되지만 슬롯이 슈퍼바이저에게 할당될 때는 사용되지 않는다. 주어진 슬롯이 슈퍼바이저에게 할당되면, 대신에 해당 슬롯은 슈퍼바이저의 컨텍스트 레지스터 파일(CXS)을 사용한다. 슈퍼바이저는 항상 자신의 컨텍스트에 액세스할 수 있으며 워커들은 슈퍼바이저 컨텍스트 레지스터 파일(CXS)를 점유할 수 없다.

[0068] 슈퍼바이저 스레드(SV)는 임의의 및 모든 타임 슬롯(S0...S3)(또는 보다 일반적으로 S0...SJ-1)에서 실행할 수 있는 성능을 갖는다. 스케줄러(24)는 전체 프로그램이 시작될 때, 슈퍼바이저 스레드를 모든 타임 슬롯에 할당함으로써 시작되도록, 즉 슈퍼바이저(SV)가 S0...SJ-1 모두에서 실행을 시작하도록 구성된다. 그러나, 슈퍼바이저 스레드는 어떤 후속 포인트에서(즉각적으로 또는 하나 이상의 슈퍼바이저 태스크를 수행한 후에), 그것이 실행

행중인 각각의 슬롯을 워커 스레드들, 예를 들어, 도 3에 도시된 초기 워커(W0...W3)(또는 보다 일반적으로 W0...WJ-1) 각각에 일시적으로 양도하는 메커니즘을 제공받는다. 이것은, 명령 메모리(12) 내의 워커 스레드의 적어도 하나의 어드레스를 오퍼랜드로서 취하는 실행 명령(run instruction)을 실행하는 슈퍼바이저 스레드에 의해 달성된다. 워커 스레드들은 서로 동시에 실행될 수 있는 코드의 일부이며, 각각은 수행될 하나 이상의 각각의 계산 작업을 나타낸다.

[0069] 실행 명령은 이 명령 자체가 실행되는 현재의 타임 슬롯을 오퍼랜드에 의해 지정된 워커 스레드에게 양도하도록 스케줄러(24)에 작용한다. 이것은 실행 명령에서 이 명령이 실행되는 타임 슬롯이 양도되고 있다는 암시임에 유의한다(기계 코드 명령의 컨텍스트에서 이것을 지정하기 위해 오퍼랜드가 필요하지 않음을 의미하는 암시이다-이는 오퍼코드 자체에서 암시적으로 이해된다). 따라서, 주어진 타임 슬롯은 슈퍼바이저가 실행 명령을 실행하는 타임 슬롯이다.

[0070] 슈퍼바이저 스레드(SV)는 하나 이상의 다른 타임 슬롯 각각에서 유사한 연산을 수행하여, 그 타임 슬롯들 중 일부 또는 전부를 (명령 메모리(12) 내의 가능한 워커 스레드들의 더 큰 세트로부터 선택된) 워커 스레드(W0...WJ-1)의 상이한 각각의 스레드에 제공한다. 일단 마지막 슬롯에 대해 이렇게 하면, 슈퍼바이저는 일시 중지된다(그런 다음 나중에 슬롯들 중 하나가 워커(W)에 의해 되돌려질 때 중지된 위치에서 다시 시작된다). 따라서, 슈퍼바이저 스레드(SV)는 인터리빙된 실행 타임 슬롯(S0...SJ-1)(예를 들어, 예시된 바와같이 J=4 또는 J=6) 중 상이한 하나들에게, 각각이 하나 이상의 작업을 수행하는 상이한 워커 스레드들을 할당할 수 있다. 슈퍼바이저 스레드는 워커 스레드를 실행할 시간이라고 결정되면, 상기 실행 명령을 사용하여 실행 명령이 실행된 타임 슬롯에 이 워커를 할당한다.

[0071] 일부 실시예에서, 명령 세트는 또한 실행 명령의 변형인 "전체 실행(run-all)"을 포함한다. 이 명령은 모두 동일한 코드를 실행하는 하나 이상의 워커 세트를 함께 론칭(launch, 시작)하는데 사용된다. 실시예에서, 이것은 처리 유닛의 슬롯(S0...S3)(또는 보다 일반적으로 S0...S(J-1)) 모두에서 워커를 론칭한다.

[0072] 일단 론칭되면, 현재 할당된 워커 스레드(W0...WJ-1) 각각은 각각의 실행 명령에 의해 지정된 코드로 정의된 하나 이상의 계산 작업을 수행하게 된다. 이 작업이 끝나면, 각각의 워커 스레드는 실행중인 타임 슬롯을 슈퍼바이저 스레드로 다시 넘긴다. 이것은 각각의 워커 스레드에서 종료 명령을 실행하여 수행된다. 종료 명령은 스케줄러(24)에 작용하여, 이 명령 자체가 실행되는 현재의 타임 슬롯을 슈퍼바이저 스레드로 다시 넘긴다. 이에 응답하여 스케줄러(24)는 해당 슬롯에서 슈퍼바이저 실행을 계속한다.

[0073] 도 4는 실행 유닛(18) 및 컨텍스트 레지스터(26)의 세부 사항을 포함하는 프로세서(4)의 추가 예시적인 세부 사항을 도시한다. 프로세서는 동시에 실행될 수 있는 M개의 스레드 각각에 대해 각각의 명령 버퍼(53)를 포함한다. 컨텍스트 레지스터(26)는 M개의 워커 컨텍스트 및 슈퍼바이저 컨텍스트 각각에 대한 각각의 메인 레지스터 파일(MRF)(26M)을 포함한다. 컨텍스트 레지스터는 적어도 각각의 워커 컨텍스트에 대한 각각의 보조 레지스터 파일(ARF)(26A)을 더 포함한다. 컨텍스트 레지스터(26)는 현재 실행중인 모든 워커 스레드가 관독하기 위해 액세스할 수 있는 공통 가중치 레지스터 파일(WRW)(26W)을 더 포함한다. WRW는 슈퍼바이저 스레드가 WRW에 기록할 수 있는 유일한 스레드라는 점에서 슈퍼바이저 컨텍스트와 연관될 수 있다. 컨텍스트 레지스터(26)는 또한 각각의 슈퍼바이저 및 워커 컨텍스트에 대한 각각의 제어 상태 레지스터(26CSR) 그룹을 포함할 수 있다. 실행 유닛(18)은 메인 실행 유닛(18M) 및 보조 실행 유닛(18A)을 포함한다. 메인 실행 유닛(18M)은 로드-저장 유닛(LSU)(55) 및 정수 연산 논리 유닛(IALU)(56)을 포함한다. 보조 실행 유닛(18A)은 적어도 부동 소수점 연산 유닛(FPU)을 포함한다.

[0074] J개의 인터리빙된 타임 슬롯(S0...SJ-1) 각각에서, 스케줄러(24)는 명령 메모리(11)로부터 현재의 타임 슬롯에 대응하는 J개의 명령 버퍼 각각으로 각각의 스레드의 적어도 하나의 명령을 페치하도록 페치 단계(14)를 제어한다. 실시예에서, 각 타임 슬롯은 프로세서의 하나의 실행 사이클이지만, 다른 방식이 배제되지는 않는다(예를 들어, 가중 라운드 로빈). 프로세서(4)의 각 실행 사이클(즉, 프로그램 카운터를 클럭킹하는 프로세서 클럭의 각 사이클)에서, 페치 단계(14)는 구현에 따라 단일 명령 또는 작은 "명령 번들(bundle)"(예를 들어, 2-명령 번들 또는 4-명령 번들)을 페치한다. 그런 다음 각 명령은 디코드 단계(16)를 통해, (코드 명령에 따라) 명령이 각각 메모리 액세스 명령인지, 정수 산술 명령인지 또는 부동 소수점 산술 명령인지 여부에 따라, 메인 실행 유닛(18M)의 LSU(55) 또는 IALU(56)중의 하나 또는 보조 실행 유닛(18A)의 FPU로 발행된다. 메인 실행 유닛(18M)의 LSU(55) 및 IALU(56)는 MRF(26M)로부터의 레지스터를 사용하여 명령들을 실행하며, MRF(26M) 내의 특정 레지스터는 명령의 오퍼랜드에 의해 지정된다. 보조 실행 유닛(18A)의 FPU는 ARF(26A) 및 WRW(26W) 내의 레지스터들을 사용하여 연산을 수행하는데, 여기서 ARF 내의 특정 레지스터들은 명령의 오퍼랜드에 의해 지정된다. 실행

시에에서, WRF 내의 레지스터들은 명령 타입(즉, 해당 명령 타입에 대해 사정-결정된)에서 암시적일 수 있다. 보조 실행 유닛(18A)은 또한 하나 이상의 타입의 부동 소수점 산술 명령의 연산을 수행하는데 사용하기 위한 일부 내부 상태(57)를 유지하는 보조 실행 유닛(18A) 내부에 논리적 래치 형태의 회로를 포함할 수 있다.

[0075] 번들로 명령을 폐치하고 실행하는 실시예에서, 주어진 명령 번들 내의 개별 명령은 (도 4에 도시된) 독립적인 파이프 라인(18M, 18A)(도 4에 도시됨)에서 병렬로 동시에 실행된다. 2개의 명령의 번들을 실행하는 실시예에서, 2개의 명령은 각각의 보조 및 메인 파이프 라인을 따라 동시에 실행될 수 있다. 이 경우, 메인 파이프 라인은 MRF를 사용하는 타입의 명령을 실행하도록 배열되고, 보조 파이프 라인은 ARF를 사용하는 타입의 명령을 실행하는데 사용된다. 적절한 상보적 번들로의 명령들의 페어링은 컴파일러에 의해 처리될 수 있다.

[0076] 각각의 워커 스레드 컨텍스트는 메인 레지스터 파일(MRF)(26M) 및 보조 레지스터 파일(ARF)(26A)(즉, 배럴-스레드 슬롯 각각에 대해 하나의 MRF 및 하나의 ARF)의 자체 인스턴스를 갖는다. MRF 또는 ARF와 관련하여 본 명세서에 기술된 기능은 컨텍스트에 기초하여 동작하는 것으로 이해되어야 한다. 그러나 스레드 간에 공유되는 단일의 공유 가중치 레지스터 파일(WRF)이 있다. 각 스레드는 자체 컨텍스트(26)만의 MRF 및 ARF에 액세스할 수 있다. 그러나, 현재 실행중인 모든 워커 스레드는 공통 WRF에 액세스할 수 있다. 따라서 WRF는 모든 워커 스레드가 사용할 공통 가중치 세트를 제공한다. 실시예들에서, 슈퍼바이저 만이 WRF에 기록할 수 있고, 워커는 WRF에서만 판독할 수 있다.

[0077] 프로세서(4)의 명령 세트는 오피코드가 실행될 때, LSU(55)로 하여금 데이터 메모리로부터 상기 로드 명령이 실행된 스레드의 각각의 ARF(26A)로 데이터를 로드하게 하는 적어도 하나의 타입의 로드 명령을 포함한다. ARF에서 대상 (destination, 목적지)의 위치는 로드 명령의 오퍼랜드에 의해 지정된다. 로드 명령의 다른 오퍼랜드는 데이터를 로드하는 데이터 메모리(22)의 어드레스에 대한 포인터를 보유하는 각각의 MRF(26M)내의 어드레스 레지스터를 특정한다. 프로세서(4)의 명령 세트는 오피코드가 실행될 때, LSU(55)로 하여금 상기 로드 명령이 실행된 스레드의 각각의 ARF로부터 데이터를 데이터 메모리(22)에 저장하게 하는 적어도 하나의 타입의 저장 명령을 또한 포함한다. ARF 내의 상기 저장 소스의 위치는 오퍼랜드에 의해 지정된다. 저장 명령의 다른 오퍼랜드는 데이터를 저장할 데이터 메모리(22)의 어드레스에 대한 포인터를 보유하는 MRF 내의 어드레스 레지스터를 특정한다. 일반적으로, 명령 세트는 개별 로드 및 저장 명령 타입, 및/또는 단일 명령으로 로드 및 저장 연산을 결합하는 적어도 하나의 로드-저장 명령 타입을 포함할 수 있다. 간략하게 보다 상세히 논의되는 바와 같이, 명령 세트는 단일 명령으로 모두 2개의 로드 및 하나의 저장 연산을 수행하는 특정 타입의 로드-저장 명령을 포함할 수 있다. 주어진 프로세서(4)의 명령 세트는 다수의 상이한 종류의 로드, 저장 및/또는 로드-저장 명령 타입을 포함할 수 있음을 주목한다.

[0078] 프로세서의 명령 세트는 또한 산술 연산을 수행하기 위한 하나 이상의 타입의 산술 명령을 포함한다. 본 명세서에 개시된 실시예들에 따르면, 이들은 공통 가중치 레지스터 파일(WRF, 26W)을 사용하는 적어도 하나의 타입의 산술 명령을 포함할 수 있다. 이러한 타입의 명령은 산술 명령이 실행된 스레드의 각각의 ARF(26A)에서 대응하는 산술 연산의 적어도 하나의 소스를 지정하는 적어도 하나의 오퍼랜드를 취한다. 그러나, 산술 명령의 적어도 하나의 다른 소스는 모든 워커 스레드에 공통적인 공통 WRF에 있다. 실시예에서, 이 소스는 당해 산술 명령에 암시되어 있다(즉, 이러한 타입의 산술 명령에 대해 암묵적이다). 기계 코드 명령이라는 견지에서 암시는 오퍼랜드가 지정하지 않아도 됨을 의미한다. 즉, 이 경우, WRF 내의 소스 위치는 (특정 오피코드에 대해 사전 결정된) 오피코드에서 고유하다. 대안적으로 다른 실시예에서, 산술 명령은 WRF의 몇몇 상이한 세트들 중에서 어떤 가중치 레지스터 세트가 가중치를 취하는지를 지정하는 오퍼랜드를 취할 수 있다. 그러나, 가중치의 소스가 WRF에서 발견된다는 사실(예컨대, 범용 MRF 또는 ARF와는 달리)은 여전히 암묵적이다.

[0079] 관련 타입의 산술 명령의 오피코드에 응답하여, 보조 실행 유닛(18A) 내의 산술 유닛(예를 들어, FPU)은 오피코드에 의해 지정된 바와같이, 스레드의 각각의 ARF내의 지정된 소스 레지스터(들) 및 WRF내의 소스 레지스터(들)의 값들에 대한 연산하는 것을 포함하는 산술 연산을 수행한다. 또한 산술 명령의 대상 오퍼랜드(destination operand)에 의해 명시적으로 지정된 스레드의 각각의 ARF에 있는 대상 레지스터로 산술 연산의 결과를 출력한다.

[0080] 공통 WRF(26W)에서 소스를 사용할 수 있는 산술 명령의 예시적인 타입은 하나 이상의 벡터 승산 명령 타입, 하나 이상의 행렬 승산 명령 타입, 하나 이상의 누적 벡터 승산 명령 타입 및/또는 누적 행렬 승산 명령 타입(명령의 하나의 인스턴스에서 다음 인스턴스로의 승산 누적하는), 및/또는 하나 이상의 컨볼루션 명령 타입을 포함할 수 있다. 예를 들어, 벡터 승산 명령 타입은 ARF(26A)로부터의 명시적인 입력 벡터에 WRF로부터의 사전 결정된 가중치 벡터를 곱할 수 있거나 또는 행렬 승산 명령 타입은 ARF로부터의 명시적인 입력 벡터에 WRF로부터의

사전 결정된 가중치 행렬을 제공할 수 있다. 다른 예로서, 컨볼루션 명령 타입은 ARF로부터의 입력 행렬을 WRF로부터의 사전 결정된 행렬과 컨볼루션할 수 있다. 복수의 스레드에 공통인 공유 가중치 레지스터 파일(WRF)을 가짐으로써, 각 스레드는 공통 커널을 자신의 각각의 데이터로 승산 또는 컨볼루션할 수 있다. 이는 기계 학습 애플리케이션에서 많이 사용되는 시나리오이므로 유용하며, 여기서 각 스레드는 신경망에서 상이한 노드를 나타내고, 공통 커널은 검색되거나 트레이닝되는 특징(예를 들어, 그래픽 데이터의 영역 또는 볼륨에서 에지 또는 특정 모양)를 나타낸다.

[0081] 실시예에서, WRF(26W)의 값들은 슈퍼바이저 스레드에 의해 기록될 수 있다. 슈퍼바이저(실시예에서 모든 슬롯(S0...SM)에서 실행됨으로써 시작함)는 먼저 일련의 풋(put) 명령을 실행하여 WRF 내의 사전 결정된 위치에 일부 공통 가중치의 값을 기록한다. 그런 다음 실행 명령(또는 전체 실행 명령)을 실행하여 일부 또는 모든 슬롯(S0...SJ-1)에서 개별 워커를 론칭한다. 그런 다음 각각의 워커는 각각의 ARF(26A)로 로딩될 때, 자신의 각각의 입력 데이터에 대응하는 산술 연산(들)을 수행하지만 슈퍼바이저가 WRF(26W)에 기록한 공통 가중치를 사용하기 위해, 전술한 타입(들)의 하나 이상의 산술 명령의 하나 이상의 인스턴스를 포함한다. 각 스레드가 해당 작업을 완료하면, 종료 명령을 실행하여 슬롯을 슈퍼바이저에게 다시 넘긴다. 론칭된 모든 스레드가 각각의 작업을 마치면, 슈퍼바이저는 새로운 값들을 WRF에 기록하고 새로운 스레드 세트를 론칭할 수 있다(또는 WRF에서 기존의 값들을 계속 사용하기 위해 새로운 세트를 론칭함).

[0082] 상기 "메인", "보조" 및 "가중치"라는 라벨이 반드시 제한적이지 않음을 알 수 있을 것이다. 실시예에서, 이들은 임의의 제1 레지스터 파일(워커 컨텍스트 당), 제2 레지스터 파일(워커 컨텍스트 당) 및 공유 제3 레지스터 파일(예를 들어, 슈퍼바이저 컨텍스트의 일부이지만 모든 워커가 액세스 가능한)일 수 있다. ARF(26A) 및 보조 실행 유닛(18)은 산술 명령(또는 적어도 부동 소수점 연산)을 위해 사용되기 때문에 산술 레지스터 파일 및 산술 실행 유닛으로 지칭될 수도 있다. MRF(26M) 및 보조 실행 유닛(18)은 이들 중 하나가 메모리를 액세스하기 위한 것이므로 메모리 어드레스 레지스터 파일 및 산술 실행 유닛으로도 지칭될 수 있다. 가중치 레지스터 파일(WRF)(26W)은 특정 타입 또는 타입들의 연산 명령에서 사용된 곱셈 가중치를 유지하는데 사용되므로 소위 더 자세히 논의될 것이다. 예: 이들은 신경망에서 노드들의 가중치를 나타내기 위해 사용될 수 있다. 다른 방법으로 볼 때, MRF는 정수 오퍼랜드를 저장하는데 사용되는 정수 레지스터 파일이라고 할 수 있지만, ARF는 부동 소수점 오퍼랜드를 저장하는데 사용되므로 부동 소수점 레지스터 파일이라고 할 수 있다. 2개의 번들로 명령을 실행하는 실시예에서, MRF는 메인 파이프 라인에 의해 사용되는 레지스터 파일이고, ARF는 보조 파이프 라인에 의해 사용되는 레지스터이다.

[0083] 그러나, 대안적인 실시예에서, 레지스터 공간(26)은 이러한 상이한 목적을 위해 이들 개별 레지스터 파일로 반드시 분할되지는 않는다는 것을 주목해야 한다. 그 대신, 메인 실행 유닛 및 보조 실행 유닛을 통해 실행되는 명령들은 동일한 공유 레지스터 파일(다중 스레드 프로세서의 경우 컨텍스트 당 하나의 레지스터 파일) 중에서 레지스터를 지정할 수 있다. 또한, 파이프 라인(13)은 반드시 명령들의 번들을 동시에 실행하기 위한 병렬 구성 파이프 라인(예를 들어, 보조 및 메인 파이프 라인)을 포함할 필요는 없다.

[0084] 또한, 프로세서(4)는 메모리(11)와 하나 이상의 다른 자원, 예를 들어 프로세서의 다른 인스턴스들 및/또는 네트워크 인터페이스 또는 네트워크 첨부 저장(twork Attached Storage:NAS)) 디바이스와 같은 외부 디바이스사이에서 데이터를 교환하기 위한 교환 인터페이스(51)를 포함할 수 있다. 도 5에 도시된 바와 같이, 실시예에서 프로세서(4)는 상호 연결된 프로세서 타일들의 어레이(6) 중 하나를 형성할 수 있으며, 각 타일은 더 넓은(wider) 프로그램의 일부를 실행한다. 따라서, 개별 프로세서(4)(타일들)는 더 넓은 프로세서 또는 처리 시스템(6)의 일부를 형성한다. 타일들(4)은 상호 접속 서브 시스템(34)을 통해 함께 접속(연결)될 수 있고, 상호 접속 서브 시스템(34)은 그들 각각의 교환 인터페이스(51)를 통해 접속된다. 타일들(4)은 동일한 칩(즉, 다이) 상에 또는 상이한 칩 상에, 또는 조합(즉, 어레이는 다중 타일(4)을 각각 포함하는 다중 칩으로 형성될 수 있음)으로 구현될 수 있다. 따라서, 상호 접속 시스템(34) 및 교환 인터페이스(51)는 내부(온-칩) 상호 접속 메커니즘 및/또는 외부(인터-칩) 교환 메커니즘을 포함할 수 있다.

[0085] 다중-스레드 및/또는 다중-타일 프로세서 또는 시스템의 하나의 예시적인 애플리케이션에서, 다수의 스레드 및/또는 타일들(4)에 대한 프로그램 실행은 신경망을 트레이닝하고 및/또는 신경망에 기초하여 추론을 수행하도록 구성된 알고리즘과 같은 기계 지능 알고리즘을 포함한다. 이러한 실시예에서, 각 워커 스레드 또는 각 타일상에서 실행되는 프로그램의 일부 또는 각 타일상의 각 워커 스레드는 신경망(일종의 그래프)의 상이한 노드(102)를 나타내는데 사용되고, 스레드들 및/또는 타일들 간의 통신은 그래프에서 노드들(102) 사이의 에지(104)를 나타낸다. 이것은 도 6에 개시되어 있다.

- [0086] 기계 지능은 기계 지능 알고리즘이 지식 모델을 학습하는 학습 단계부터 시작된다. 상기 모델은 상호 연결된 노드들(즉, 꼭지점)(102) 및 에지들(즉, 링크)(104)의 그래프를 포함한다. 그래프의 각 노드(102)는 하나 이상의 입력 에지 및 하나 이상의 출력 에지를 갖는다. 일부 노드(102)의 입력 에지들 중 일부는 노드의 다른 일부의 출력 에지이며, 이에 따라 노드를 함께 연결하여 그래프를 형성한다. 게다가, 하나 이상의 노드(102)의 하나 이상의 입력 에지는 전체적으로 그래프에 대한 입력을 형성하고, 하나 이상의 노드(102)의 하나 이상의 출력 에지는 다음과 같이 전체적으로 그래프의 출력을 형성한다. 때로는 주어진 노드가 그래프의 입력, 그래프의 출력 및 다른 노드와의 연결 등 이들 모두를 가질 수 있다. 각각의 에지(104)는 값 또는 더 자주 텐서(n차원 행렬)를 전달하며, 이들은 각각 그들의 입력 및 출력 에지상에서 노드들(102)에 제공되는 입력 및 출력을 형성한다.
- [0087] 각각의 노드(102)는 그 입력 에지 또는 에지들 상에서 수신되는 하나 이상의 입력의 함수를 나타내며, 이 함수의 결과는 출력 에지 또는 에지들 상에 제공된 출력(들)이다. 각 함수는 하나 이상의 파라미터(때로는 곱셈 가중치일 필요는 없지만 가중치라고도 함)로 파라미터화된다. 일반적으로, 상이한 노드들(102)에 의해 표현되는 함수들은 상이한 형태의 함수일 수 있고 및/또는 상이한 파라미터에 의해 파라미터화될 수 있다.
- [0088] 게다가, 각 노드의 함수의 하나 이상의 파라미터 각각은 각각의 에러 값에 의해 특징화된다. 또한, 각각의 조건은 각 노드(102)의 파라미터(들) 내의 에러(들)와 관련될 수 있다. 단일 파라미터에 의해 파라미터화된 함수를 나타내는 노드(102)에 대해, 조건은 단순 임계값일 수 있다. 즉, 에러가 특정 임계값 내에 있으면 조건이 충족되지만 에러가 임계값을 초과하면 조건이 충족되지 않는다. 하나 이상의 파라미터에 의해 파라미터화된 노드(102)에 대해, 허용 가능한 에러 레벨에 도달한 노드(102)에 대한 조건은 더 복잡할 수 있다. 예를 들어, 조건은 해당 노드(102)의 각 파라미터가 각각의 임계값 내에 있는 경우에만 충족될 수 있다. 다른 예로서, 결합 메트릭(combined metric)이 동일 노드(102)에 대한 상이한 파라미터의 에러를 결합하여 정의될 수 있고, 조건은 결합 메트릭의 값이 지정된 임계값 내에 있는 조건에서 만족될 수 있지만, 그렇지 않은 경우 상기 조건은 결합 메트릭의 값이 임계값을 초과하면 조건이 만족되지 않는다(또는 메트릭의 정의에 따라 그 반대일 수도 있음). 조건이 무엇이든, 이것은 노드의 파라미터(들)의 에러가 일정 레벨 또는 수용도 이하로 떨어지는지의 척도를 제공한다. 일반적으로 임의의 적합한 메트릭이 사용될 수 있다. 조건 또는 메트릭은 모든 노드에 대해 동일할 수도 있고, 각각의 노드에 대해 다를 수도 있다.
- [0089] 학습 단계에서, 알고리즘은 경험 데이터, 즉 그래프에 대한 상이한 입력 조합을 나타내는 다수의 데이터 포인트를 수신한다. 점점 더 많은 경험 데이터가 수신됨에 따라, 알고리즘은 경험 데이터에 기초하여 그래프의 다양한 노드(102)의 파라미터를 점진적으로 조정(tune)하여 파라미터들의 에러를 최소화하려고 시도한다. 목표는 그래프의 출력이 주어진 입력에 대해 원하는 출력에 가능한 가깝도록 파라미터들의 값을 찾는 것이다. 그래프가 전체적으로 이러한 상태에 가까워지면 그래프가 수렴한다고 한다. 적절한 수렴도 이후에 그래프는 예측 또는 추론의 수행, 즉 일부 주어진 입력에 대한 결과를 예측하거나 일부 주어진 출력에 대한 원인을 추론하는데 사용될 수 있다.
- [0090] 학습 단계는 다수의 상이한 가능 형태를 취할 수 있다. 예를 들어, 슈퍼바이저식 접근법에서, 입력 경험 데이터는 트레이닝 데이터, 즉 알려진 출력에 대응하는 입력의 형태를 취한다. 각 데이터 포인트에서 알고리즘은 출력이 주어진 입력에 대한 알려진 출력과 더 가깝게 매칭하도록 파라미터들을 조정할 수 있다. 후속 예측 단계에서, 그래프는 입력 쿼리를 근사적으로 예측된 결과에 매핑하는데 사용될 수 있다(또는 추론하는 경우는 그 반대임). 다른 접근법도 가능하다. 예를 들어 비슈퍼바이저식 접근법에서, 입력 데이터마다 참조 결과에 대한 개념이 없는 대신 기계 지능 알고리즘이 출력 데이터에서 자체 구조를 식별하도록 남겨 둔다. 또는 강화 접근법에서, 알고리즘은 입력 경험 데이터의 각 데이터 포인트에 대해 적어도 하나의 가능한 출력을 시도하고, 이 출력이 양 또는 음(및 잠재적으로 양 또는 음인 정도)인지 여부, 예를 들어, 승패(win or lose), 또는 보상 또는 비용 등을 알려준다. 많은 시도에서 알고리즘은 긍정적인 결과를 생성할 입력을 예측할 수 있도록 그래프의 파라미터들을 점차적으로 조정할 수 있다. 그래프를 학습하기 위한 다양한 접근법 및 알고리즘은 기계 학습 분야의 당업자에게 공지될 것이다.
- [0091] 본 명세서에 개시된 기술들의 예시적인 애플리케이션에 따르면, 각 워커 스레드는 신경망과 같은 기계 지능 그래프에서 노드들(102) 중 개개의 하나와 관련된 계산을 수행하도록 프로그래밍된다. 이 경우, 노드들(102) 간의 에지들(104) 중 적어도 일부는 스레드들 간의 데이터 교환에 대응하고, 일부는 타일들 간의 교환을 수반할 수 있다. 타일(4) 당 다수의 스레드를 갖는 다중-타일 배열(6)의 경우, 각 타일(4)은 그래프의 서브 그래프를 실행한다. 각 서브 그래프는 하나 이상의 슈퍼바이저 스레드를 포함하는 슈퍼바이저 서브 프로그램, 및 각각의 서브 그래프의 노드들(102)을 나타내는 워커 스레드 세트를 포함한다.

- [0092] 기계 지능과 같은 애플리케이션에서, 프로세서(4)(예를 들어, 타일)의 데이터 메모리(22)로 및 그로부터 데이터를 효율적으로 스트리밍할 수 있는 것이 바람직할 것이다. 예를 들어, 이것은 벡터 내적(dot product) 명령, 행렬 곱(products), 누적 벡터 내적, 누적 행렬 곱 또는 전용 컨볼루션 명령과 같은 복잡한 산술 명령의 시퀀스를 동반하여, 고도의 산술 복잡성을 단일 산술 명령으로 패키징(pack)하는 것이 유용할 것이다(배타적이지 않음). 이러한 문제점을 해결하기 위해, 본 발명은 프로세서(4)의 명령 세트에서 2개의 로드 연산과 하나의 저장 연산을 수행한 다음 명령의 오퍼코드의 단일 인스턴스에 모두 응답하여, 로드 및 저장 어드레스 각각에 독립적인 스트라이드(stride)를 자동으로 적용하는 로드-저장 명령 타입을 제공한다. 게다가, 명령은 적절한 레지스터 파일(예를 들어, MRF(26M))에서 어드레스의 효율적인 패키징을 사용하여, 3개의 어드레스(2개의 로드 어드레스 및 하나의 저장 어드레스)가 2개의 레지스터의 공간으로 패키징된다. 이것은 MRF(26M)로부터 LSU(3개의 레지스터 각각을 액세스하기 위한 하나의 개별 포트)까지의 3개의 포트만을 이용하여 단일 명령으로 3개의 메모리 어드레스와 스트라이드 레지스터에 액세스할 수 있게 한다.
- [0093] 이 명령은 두개의 64 비트값을 로드하고 "페이스(pace)"가 스트라이드의 동의어인 하나의 64 비트값을 저장한다는 사실을 참조하여 "ld2xst64pace"로 본 명세서에서 지칭될 수 있다. 그러나 이 레이블 또는 특정 비트 폭은 반드시 제한적인 것으로 간주되어서는 안된다. 명령은 더 일반적으로 로드-저장 명령(이것은 반드시 프로세서의 명령 세트 내의 유일한 로드-저장 명령의 종류를 의미하지는 않음) 또는 "ldx2st"(두 번 로드하고 저장)라고도 지칭된다. 그 구문은 다음과 같습니다.
- [0094] ld2xst \$aDst, \$aSrc, \$mAddr, \$mStride, Strimm
- [0095] \$aDst는 보조(또는 산술) 레지스터 파일(ARF)(26A)에서 대상(destination)을 식별하는 하나 이상의 오퍼랜드를 지칭하며, 상기 대상은 ARF의 하나 이상의 레지스터를 포함한다. 실시예에서, ARF내의 레지스터 각각은 32 비트 폭이고, 대상(\$ aDst)은 4개의 32 비트 레지스터: \$aDst0:Dst0+3로 구성될 수 있다. ld2xst64pace의 대상 오퍼랜드는 ARF내의 이러한 대상 레지스터들 중 단지 하나의 위치(예컨대, 최하위 \$aDst0)를 지정할 수 있고, 다른 레지스터(들)의 위치는 이 위치에 대해 암시적일 수 있다(예를 들어, ARF내의 다음 3개의 연속 레지스터). 대안적으로 다른 구현예에서, (비록 증가된 명령폭을 요구할지라도) 다수의 대상 오퍼랜드가 개별 오퍼랜드에 의해 명시적으로 및 독립적으로 각각 식별될 수 있다는 것을 배제하지 않는다.
- [0096] \$aSrc는 ARF(26A)의 소스를 식별하는 하나 이상의 오퍼랜드를 지칭하며, 소스는 ARF의 레지스터들 중 하나 이상을 포함한다. 실시예에서, ARF의 레지스터 각각은 32 비트 폭이고, 소스(\$aSrc)는 \$aSrc0:Src+1과 같은 2개의 32 비트 레지스터로 구성될 수 있다. ld2xst64pace의 소스 오퍼랜드는 ARF내의 이러한 소스 레지스터들 중 단지 하나의 위치(예컨대, 최하위 \$aSrc0)를 지정할 수 있고, 다른 레지스터(들)의 위치는 이 위치에 대해 암시적일 수 있다(예를 들어, ARF내의 다음 연속 레지스터). 대안적으로, 다른 구현예에서, (비록 증가된 명령폭을 요구할지라도) 다수의 소스 오퍼랜드가 개별 오퍼랜드에 의해 명시적으로 및 독립적으로 각각 식별될 수 있다는 것을 배제하지 않는다.
- [0097] \$mAddr은 메인 레지스터 파일(MRF)(25M)에 있는 두개의 레지스터(\$mAddr0: Addr0+1)의 위치를 지정하는 하나 이상의 오퍼랜드를 지칭하고, 두 레지스터 사이에는 3개의 어드레스 즉, 2개의 로드 어드레스와 하나의 저장소 어드레스가 있다. 실시예에서, MRF 내의 레지스터 각각은 32 비트 폭이다. 메모리 어드레스 오퍼랜드는 MRF 내의 이들 레지스터들 중 단지 하나의 위치(예를 들어, 최하위 \$mAddr0)를 지정할 수 있고, 다른 레지스터의 위치는 이것에 대해 암시적일 수 있다(예를 들어, MRF 내의 다음 연속 레지스터). 대안적으로, 다른 구현예에서, (비록 증가된 명령폭을 요구할지라도) 메모리 어드레스 레지스터들이 개별 오퍼랜드에 의해 명시적으로 및 독립적으로 각각 식별될 수 있다는 것을 배제하지 않는다.
- [0098] Strimm은 즉치(immediate) 오퍼랜드 세트이며, 2개의 로드 및 1개의 저장 어드레스 각각에 대한 각각의 오퍼랜드를 포함한다. MRF(26M)에는, 복수의 필드를 포함하는 스트라이드 레지스터가 제공된다. 각 필드는 사전 결정된 상이한 가능 스트라이드 값 세트 중 상이한 하나를 보유한다. 스트라이드 값은 전형적으로 일련의 단계에서 사용하기 위해 메모리 어드레스, 즉 메모리 어드레스 단계를 증가시키는 값이다. 2개의 로드 및 1개의 저장 각각에 대해, 각각의 즉치 스트라이드 오퍼랜드는 명령의 현재 인스턴스의 각각의 로드 또는 저장 연산을 수행한 후에, 스트라이드 레지스터의 어느 필드로부터 스트라이드 값을 취하여 각각의 로드 또는 저장 어드레스에 적용할지를 지정한다. 이것은 로드-저장 명령의 후속 인스턴스의 이익을 위해 MRF(26M)의 어드레스를 따라 이동한다.
- [0099] 실시예에서, 로드-저장 유닛(55)은 즉치 스트라이드 오퍼랜드의 하나의 특정 값이 레지스터 필드를 가리키는 것이 아니라 1의 스트라이드 값(즉, 사용된 어드레스 공간에서 한 어드레스 단위의 증가)을 직접 지정할 수 있는

특징을 지원한다. 즉, 즉치 스트라이드 오퍼랜드가 취할 수 있는 값 범위에서, 값들 중 하나는 스트라이드를 지정하고, 다른 일부 또는 전체는 스트라이드 레지스터에 프로그램 가능한 스트라이드 값들을 유지하는 상이한 가능 필드를 지정한다. 여기서 1 유닛(단위)은 데이터 액세스의 원자 사이즈를 의미한다. 예를 들어, 오퍼랜드가 16비트 부동 소수점값의 4-엘리먼트 벡터인 경우, 증분은 1원자/단위이며, 이는 8 바이트(64 비트)와 동일하다.

[0100] 스트라이드 레지스터는 MRF(26M)의 레지스터이다. 실시예에서, 위커는 (자신의 MRF에 있는) 스트라이드 레지스터의 필드에 스트라이드 값을 기록하는 책임이 있다. 대안적으로 다른 구현예에서, 스트라이드 레지스터의 필드 내의 스트라이드 값들은 슈퍼바이저 스레드(SV)에 의해 기록될 수 있거나 접근법들의 조합이 사용될 수 있음을 배제하지 않는다.

[0101] \$mStride를 지정하는 오퍼랜드에 대한 요구 사항은 구현에 따른 프로세서(4)의 선택적 특징이다. 실시예에서, 로드-저장 명령은 MRF(26M)에서 스트라이드 레지스터(\$mStride)의 위치를 지정하는 오퍼랜드를 취한다. 따라서, 프로그램은 복수의 가능 스트라이드 레지스터들 중에서 스트라이드 레지스터를 선택할 수 있어서, 스트라이드의 선택에 훨씬 더 많은 융통성을 제공한다. 그러나, 다른 구현에서는 \$mStride의 위치가 고정되거나 암시적일 수 있고 로드-저장 명령에서 오퍼랜드가 필요하지 않을 수도 있다.

[0102] 도 7은 MRF(26M) 내의 2개의 32 비트 레지스터에 각각이 21 비트인 3개의 어드레스를 패킹하는 예를 도시한다. 도 7은 또한 MRF내의 하나의 32 비트 레지스터에 각각의 스트라이드 값이 10비트(이 경우에서 레지스터의 2비트는 사용되지 않음)인 3개의 스트라이드 값을 패킹하는 예를 도시한다. 따라서, 로드-저장 명령은 단일 명령으로 MRF의 단지 2개의 32 비트 폭 포트를 통해 3개의 메모리 어드레스에 액세스할 수 있다. 세번째 32 비트 폭 포트는 스트라이드 레지스터에 액세스하기 위해 동일한 명령에 의해 사용될 수 있다. 예를 들어, MRF내의 32 비트 폭 스트라이드 레지스터(해당 레지스터의 2비트는 사용하지 않고 남겨둠)에 패킹된 3개의 10 비트 폭 필드에 3개의 10 비트 스트라이드 값(어드레스 델타)을 포함한다. 도 7에 표시된 특정 패킹은 단지 예일뿐이다. 예를 들어, 다른 구현예에서 addr2는 addr1이 아니라 두 레지스터에 걸쳐 있을 수 있다.

[0103] 도 8은 MRF(26M) 내의 다수의 스트라이드 레지스터(\$mStrideA, \$mStrideB, \$mStrideC...)의 배열을 도시한다. 이들 중 하나는 로드-저장 명령의 대응하는 오퍼랜드에 의해 스트라이드 레지스터(\$mStride)로 지정될 수 있다. 각각의 가능한 스트라이드 레지스터는 복수의 필드, 예를 들어, 실시예에서는 3개를 포함한다(구현에 따라 로드 및 저장 어드레스와 동일한 수의 필드 또는 스트라이드 레지스터가 반드시 존재할 필요는 없음). 2개의 로드 및 1개의 저장 어드레스 각각에 대해, 즉치 오퍼랜드 세트(Strimm)내의 각각의 즉치 오퍼랜드는 스트라이드 레지스터(\$mStride)의 가능한 필드들 중 하나를 지정하여 각각의 스트라이드 값(각각의 로드 또는 저장 다음에 적용될 어드레스 델타)을 취할 수 있다. 예를 들어, 실시예에서, 스트라이드 오퍼랜드가 2개의 로드 및 1개의 저장 각각에 대해 2-비트인 경우, 가능한 값들 중 3개는 3개의 스트라이드 필드 중 서로 다른 값을 지정하고, 다른 가능한 값은 단순히 레지스터에 유지된 값을 참조하지 않고 하나의 스트라이드를 지정한다. 예를 들어, 00은 스트라이드 1(원자)을 지정하고, 01은 스트라이드 레지스터의 첫 번째 필드를 지정하고, 10은 스트라이드 레지스터의 두 번째 필드를 지정하고, 11은 스트라이드 레지스터의 세 번째 필드를 지정한다.

[0104] 실시예에서, 스트라이드 즉치의 가능한 값들 중 하나는 스트라이드 레지스터(\$mStride)의 필드가 아니라 1의 디폴트 증분을 지정한다는 점에 주목한다.

[0105] 동작시, 실행 유닛(18)에 의해 실행될 때, (디코드 단계(16)에 의한 디코딩 후에) 로드-저장 명령의 오퍼코드는 LSU(55)를 트리거하여 다음 연산들을 수행한다. MRF(26M)의 \$mAddr에 저장된 어드레스에 의해 지정된 대로 상기 메모리의 2개의 로드 어드레스로부터의 값을 \$aDsc에 의해 지정된 ARF(26A)의 대상(목적지)으로 로딩한다. 또한 MRF의 \$mAddr에 저장된 어드레스에 의해 지정된 대로 ARF의 \$aSrc로부터의 값(들)을 메모리(22)의 저장소 어드레스에 저장한다. 그런 다음 LSU(55)는 3개의 즉치 오퍼랜드(Strimm) 중의 각각의 하나에 의해 지정된 대로 MRF의 각 스트라이드 레지스터(\$mStride)의 필드로부터의 각각의 스트라이드 값으로 2개의 로드 및 1개의 저장 어드레스 각각을 독립적으로 사후-증가(post-increments)시킨다.

[0106] 주(note): 로드-저장 명령의 현재의 인스턴스의 2개의 로드 및 1개의 저장 어드레스 각각에 대해, 현재 명령의 스트라이드 연산은 현재 로드-저장 연산의 각각의 로드 및 저장 연산에 후속하는 각각의 스트라이드로 각각의 어드레스를 증가시킨다. 구현에 따라, 이것은 모든 증가가 로드 및 저장 후에 함께 적용된다는 것, 예를 들어 로드, 로드, 저장, 증가, 증가, 증가된다는 것을 의미할 수 있다. 대안적으로, 스트라이드 연산은 예를 들어 로드, 증가, 로드, 증가, 저장, 증가와 같이 각 로드 및 저장 바로 후속하는 각각의 어드레스를 증가시킬 수 있다. 실제로 저장은 로드들 중 하나 또는 둘 모두의 앞에 올 수 있다. 중요한 것은 각 로드 어드레스의 증가가 각각의 로드 이후에 이루어지고, 저장 어드레스의 증가는 저장 이후에 완료된다는 것이다. 요점은 MRF의 로드

및 저장 어드레스를 따라 이동하여 로드-저장 명령의 후속 인스턴스를 준비하는 것이다.

- [0107] 실시예에서, 메모리(11)(또는 적어도 데이터 메모리(22))는 메모리로부터 데이터를 로딩하기 위한 단지 2개의 64 비트 포트 및 메모리로 데이터를 저장하기 위한 하나의 64 비트 폭 포트만을 갖는다. 실시예에서, (주어진 컨텍스트의) MRF(26M)는 로드 저장 유닛(55)에 단지 3개의 32 비트 폭 포트를 가지며, (주어진 컨텍스트)의 ARF(26A)는 로드 저장 유닛(55)에 단 하나의 64 비트 폭 포트를 갖는다. (주: 도시된 예에서, IALU(56)는 MRF(26M)로부터 포인터를 검색하고 이들로부터 LSF(55)로 전달하기 위한 어드레스들을 계산하기 위해 사용되고 있으므로 사실상 IALU(56)는 LSU(55)의 일부로서 동작한다. 따라서, 이 예에서 MRF(26M)로부터 LSU(55)로의 3개의 포트는 MRF(26M)로부터 IALU(56)까지의 3개의 포트를 포함한다. 또한, 다른 구현들에서, LSU(55)는 MRF(26M)로부터 포인터들을 직접 검색하고 그것에 기초하여 자신의 어드레스들을 계산할 수 있다는 것을 배제하지 않는다.)
- [0108] 실시예에서, ARF(총 128 비트) 내의 4개의 32-비트 대상 레지스터(\$aDst0 : Dst0+3)은 메모리(22)로부터 예를 들어, 16 비트 부동 소수점 (f16) 값들의 4-엘리먼트 벡터와 32 비트 부동 소수점(f32) 값들의 2-엘리먼트 벡터를 로딩하는데 사용될 수 있다. ARF(총 64 비트) 내의 2개의 32 비트 소스 레지스터(\$aSrc0:Src+1)는 메모리에 예를 들어, f32 값들의 32 비트의 2-엘리먼트 벡터를 저장하는데 사용될 수 있다.
- [0109] 프로그램에서 사용하기 위해, 로드-저장 명령은 로드 명령의 대상으로부터 입력을 취하고, 이에 기초하여 연산을 수행하고, 결과를 로드-저장 명령의 소스로 출력하는 산술 명령과 같은 다른 타입의 명령들 사이에 산재되어 있다. 즉, 프로그램은 이전에 언급한 로드-저장 명령의 인스턴스들과, 벡터 내적 명령, 행렬 곱셈 명령, 누적 벡터 내적 명령, 누적 행렬 곱셈 명령 또는 컨볼루션 명령과 같은 적어도 하나의 산술 명령의 인스턴스들을 포함하고, 상기 로드-저장 명령들의 적어도 일부의 대상은 상기 산술 명령들 중 적어도 일부의 소스이고, 상기 산술 명령들 중 적어도 일부의 대상은 로드-저장 명령들의 적어도 일부의 소스이다. 로드-저장 명령의 시맨틱 밀도가 높기 때문에, 두 로드 및 스트라이드 기능을 통해, 프로그램은 로드 및 저장 연산에 소비되는 작은 코드 오버 헤드로 데이터를 효율적으로 처리할 수 있다.
- [0110] 예를 들어, 프로그램은 각각 명령 쌍이로드-저장 명령의 인스턴스 다음에 산술 명령의 대응하는 인스턴스로 구성되거나 동일한 번들에서 동시에 동반되는 일련의 명령 쌍을 포함할 수 있다. 각 명령 쌍에서, 로드-저장 명령의 소스는 선행 쌍으로부터의 산술 명령의 대상으로서 설정되고, 로드-저장 명령의 대상은 현재 또는 후속 쌍에서 산술 명령의 산술 명령의 소스로서 설정된다. 예를 들어 다음과 같은 구문을 갖는 산술 명령 "arith"를 고려한다.
- [0111] arith \$aDst, \$aSrcA, \$aSrcB
- [0112] 여기서 \$aDst는 ARF(26A)의 대상을 지정하는 오퍼랜드이고, \$aSrcA, \$aSrcB는 ARF내의 2의 소스를 지정하는 오퍼랜드를 지칭한다("arith"는 적어도 이 구문을 갖는 임의의 산술 명령에 대해 본 명세서에 사용된 일반 이름이다). 프로그램은 일련의 명령 쌍으로 프로그램될 수 있다. 예를 들면 다음과 같다.
- [0113] ...
- [0114] ldx2st Xin-Pin,Pout,Tripacked, Strides;arith Pout,Xin, Pin;
- [0115] ldx2st Xin-Pin,Pout,Tripacked, Strides;arith Pout,Xin, Pin;
- [0116] ldx2st Xin-Pin,Pout,Tripacked, Strides;arith Pout,Xin, Pin;
- [0117] ldx2st Xin-Pin,Pout,Tripacked, Strides;arith Pout,Xin, Pin;
- [0118] ...
- [0119] 또는:
- [0120] ...
- [0121] ldx2st Xin-Pin,Pout_A,Tripacked, Strides;arith Pout_A,Xin, Pin;
- [0122] ldx2st Xin-Pin,Pout_B,Tripacked, Strides;arith Pout_B,Xin, Pin;
- [0123] ldx2st Xin-Pin,Pout_A,Tripacked, Strides;arith Pout_A,Xin, Pin;
- [0124] ldx2st Xin-Pin,Pout_B,Tripacked, Strides;arith Pout_B,Xin, Pin;

- [0125] ...
- [0126] 기타
- [0127] 실시예에서, 각각의 쌍은 명령 번들, 즉 각각의 파이프 라인을 따라 동시에 실행된다. 예를 들어, 실시예들에서, 로드-저장 명령은 MRF(26M)를 사용하여 메인 파이프 라인에 의해 실행되고, 산술 명령은 ARF(26A)를 사용하여 보조 파이프 라인에 의해 병렬로 실행된다. 그러나, 대안 구현예에서 로드 저장 및 산술 명령이 단일 파이프 라인 처리 유닛을 통해 하나씩 실행될 수 있다는 점을 배제하지 않는다.
- [0128] 로드 저장 명령(ldx2st)의 인스턴스에서, Xin-Pin은 ARF(26A)에서 2개의 로드 연산의 대상(\$dsc)(예를 들어, 4개의 32 비트 레지스터(\$dsc0:dsc0+3)에 로딩된 4x16 입력 벡터(Xin) 및 4x32 입력 벡터(Pin))을 나타내고, Pout은 ARF 내의 저장(소)의 소스(예를 들어, 2개의 32 비트 레지스터(\$aSrc0:Src0+1)로부터 저장된 2x32 값)를 나타낸다. "Tripacked"는 MRF(26M)에서 (예를 들어, 2개의 32 비트 레지스터내에 3개의 21 비트 어드레스 포인터를 보유하는) 3중 패킹된 어드레스 레지스터 쌍(\$mAddr0:Addr0+1)을 나타낸다. "스트라이드(Strides)"는 3개의 즉치 값(Strimm)에 의해 지정된 MRF에서 스트라이드 레지스터 (\$mStride)내의 필드를 참조하여(도 8 참조), 2개의 로드 및 저장 연산 각각에 대한 스트라이드를 지정하는 스트라이드 오퍼랜드들(\$mStride and Strimm)을 나타낸다. 산술 명령의 인스턴스에서, Xin, Pin은 현재 또는 이전 명령 쌍에서 로드-저장 명령에 의해 수행된 로드의 대상(목적지)과 동일하게 설정되는 산술 명령의 입력들이 취해지는 소스 레지스터들을 나타낸다. Pout은 후속 명령 쌍에서 로드-저장 명령의 저장 연산 소스와 동일하게 설정되는 ARF의 산술 명령의 출력 대상을 가리킨다. 각 쌍을 2개의 동시 명령의 번들로서 실행하는 실시예에서, Xin, Pin은 이전 번들의 로드-저장 명령에 의해 수행된 로드의 대상과 동일하게 설정되고, Pout은 다음 번들에 있는 ldx2st 명령의 저장 연산 소스와 동일하게 설정된다. 이것은 동일한 레지스터들을 재사용하지만, 두 명령이 병렬로 실행되기 때문에, 현재의 산술 명령에 대한 입력 데이터는 이전의 로드 명령에 의해 읽혀진 것이다. 그 다음 현재의 ldx2st는 동일한 레지스터를 다시 사용하여 다음 산술 명령에 대한 입력 데이터를 준비한다.
- [0129] 따라서 각 쌍에서, 로드-저장(ldx2st)은 이전 명령 이전 쌍(예를 들어, 번들)에서 이전의 산술 명령으로부터 레지스터(Pout)에서 현재 발견된 출력을 저장하고, 또한 메모리로부터의 값들을 레지스터(Xin 및 Pin)에 로딩한다. 동일한 쌍(번들)에서 다음의 산술 명령은 이전에 로딩된 값들에 기초하여 산술 연산을 수행하여, 후속 쌍의 후속 로드-저장 명령의 저장 연산에 의해 저장하기 위해 레지스터(Pout)로 출력한다.
- [0130] 실시예들에서, 산술 명령은 공유된 WRF(26W)에서 적어도 제3의 암시적 소스를 갖는다. 이것들은 WRF에 기록하는 슈퍼바이저 쓰레드에 의해 설정될 수 있다. 따라서, 슈퍼바이저는 모든 실행중인 워커 쓰레드의 산술 연산들에 의해 암시적으로 사용될 공통 가중치를 설정한다.
- [0131] MRF(26M)의 \$mAddr0:Addr0+1을 지정하는 메모리 어드레스 "Tripacked"의 값들은 또한 각각의 워커에 의해 일부 초기 값으로 설정된다. MRF의 각 가능한 스트라이드 레지스터(\$mStride)의 필드에 있는 스트라이드 값들은 또한 각각의 워커에 의해 설정된다. 로드 저장 명령이 실행될 때마다, 메모리 어드레스 레지스터들(\$mAddr0:Addr0+1)내의 3개의 어드레스(2개의 로드 어드레스와 1개의 저장 어드레스) 각각은, 각각의 스트라이드 값만큼 사후 증가된다(도 7 및 도 8 참조). 예를 들어, 로드 저장 명령이 \$mStride = \$mStrideA 및 Strimm = 011000(제1 즉치 스트라이드 오퍼랜드 = 01, 제2 = 10, 제3 = 00)을 지정한다고 가정해 보자. 이것은 제1 로드를 수행한 후 \$mStrideA의 필드 0에 있는 스트라이드 값만큼 제1 로드 어드레스를 증가시키고, 제2 로드를 수행한 후 제2 로드 어드레스를 \$mStrideA의 필드 1에 있는 스트라이드 값만큼 증가시키며, 저장 후 저장 어드레스를 원자 증가(사용중인 어드레스 공간에서 1의 메모리 어드레스 단계만큼 증가)만큼 증가시키는 것을 의미한다. 로드-저장 명령의 후속 인스턴스는 \$mStride와 동일하거나 다른 값을 지정하고 3개의 즉치 오퍼랜드 각각과 동일하거나 다른 값을 지정하도록 설정할 수 있다. 따라서, 일련의 각 로드-저장 명령으로, 상기 로드들 및 저장의 위치가 제어 가능한 방식으로 함께 이동될 수 있다. 따라서, 상기 코드는 새로운 어드레스를 계산하기 위해 추가 정수 산술 명령을 필요로하지 않고 유연한 방식으로 입력 데이터 공간을 효율적으로 스캔할 수 있다.
- [0132] 도 9는 이것이 유용할 수 있는 예시적인 애플리케이션, 즉 입력 데이터(Xin)의 다차원 부분을 갖는 커널(K)의 컨볼루션을 수행하는 예를 도시한다. 예를 들어, 입력 데이터는 3D 볼륨의 입력 데이터일 수 있고, 커널(K)은 3D 커널일 수 있다. 입력 데이터는 또한 호출된 입력 채널이라고도 하며, 출력 데이터는 출력 특징 채널이라고 할 수 있다. 데이터가 다차원이라고 하면, 이것은 데이터가 2 차원 이상의 좌표계에서 좌표의 복수의 조합 각각에서 값을 취하며, 각 차원(즉, 각 축)은 상이한 독립 변수(및 주어진 좌표 조합에서의 데이터 값은 종속 변수)임을 의미한다.

[0133] 이 예는 각 데이터 값이 픽셀을 나타내는 이미지 처리에 있다. 예를 들어, 일반적으로 8 비트는 적(Red), 녹(Green) 및 청(Blue) 각각에 사용되고, 그런 다음 8 비트 정수값은 컨볼루션 연산에 앞서 f16 값으로 변환될 것이다. 각 R, G, B 값은 개별 입력 평면(즉, 입력 채널에 개별적으로)으로서 취급된다. 따라서 각 2D 입력 이미지는 x, y, z 차원을 가지며 R, G, B 평면은 z 축에 채워진다. 데이터의 볼륨은 예를 들어 프레임들의 시퀀스를 포함할 수 있는데, 여기서 볼륨의 2 차원은 프레임 영역의 공간적 x 및 y 축을 나타내고 3 차원은 시퀀스내의 상이한 프레임들에 대응하는 시간축을 나타낸다. 다른 예에서, 볼륨의 3 차원은 공간 또는 오브젝트의 3D 모델 또는 이미지의 3개의 공간적 x, y 및 z 차원을 나타낼 수 있다. 다른 예에서, 데이터는 정지 이미지의 공간적 x 및 y 축을 나타내는 단지 2 차원일 수 있고, z 축은 동시에 처리될 입력 이미지의 수를 나타낸다. 개시된 기술들의 적용 가능성은 이미지 또는 그래픽 처리에 제한되지 않으며, 더 높은 차원을 갖는 데이터 또한 배제되지 않는다는 것을 유의해야 한다. 예를 들어, 2차원, 3차원 또는 그 이상의 차원은 사용자 행동을 학습하는 애플리케이션에서 사용자 또는 사용자의 상황의 상이한 측면을 나타낼 수 있다.

[0134] 커널(K)은 사전 결정된 가중치들의 행렬이다. 이는 신경망이 인식되도록 트레이닝되거나 후속 추론에서 탐색되고 있는 소정 특징을 나타낼 수 있다. 예를 들어, 이미지 데이터의 경우, 커널은 에지 또는 일부 다른 특정 종류의 모양을 나타낼 수 있다. 기계 학습 애플리케이션에서, 신경망의 각 노드는 각각의 커널에 대응할 수 있다(나중에 더 논의되겠지만, 많은 신경망에서 동일한 커널 값을 사용하지만 단지 다른 연결(104)을 갖는 다중 노드(102)가 있다). 실시예에서, 각 노드(102)는 각각의 워커 스레드에 의해 표현되므로, 각 스레드는 해당 스레드의 각각의 입력 데이터로 커널(K)의 각각의 컨볼루션을 수행한다. 워커 스레드들 중 적어도 일부는 그 값이 공유 가중치 레지스터 파일(WRF)(26W)에 저장된 동일한 커널(K)을 사용할 수 있지만, 이러한 각각의 스레드는 자신의 각각의 보조(또는 산술) 레지스터 파일(ARF)(26A)을 통해 메모리(22)로부터 및 메모리(22)로 스트리밍되는 스레드 자신의 각각의 입력 데이터로 해당 커널을 컨볼루션한다.

[0135] 3 차원의 이산 컨볼루션(*)은 다음과 같이 표현될 수 있다.

$$Xin * K[x, y, z] = \sum_{x', y', z'} (K[x', y', z'] Xin[x-x', y-y', z-z'])$$

[0137] 따라서, 컨볼루션은 커널이 입력 데이터로 중첩될 수 있는 가능 위치들의 일부 또는 전부에 걸쳐 체계적으로 커널(K)을 스캐닝하는 것을 포함한다. 각각의 이러한 위치에서, 커널과 중첩되는 입력 데이터 부분의 각 픽셀(또는 데이터 포인트)은 대응하는 중첩 포인트에서의 커널 값과 곱해지며, 이들 각각의 승산 결과는 해당 특정 중첩 위치에서 입력 데이터로 커널의 컨볼루션을 나타내는 스칼라 출력을 제공하도록 합산된다. 그런 다음 이것은 출력 데이터의 해당 볼륨(커널 위치당 하나의 스칼라 값)을 제공하기 위해 입력 데이터(Xin)에 대해 커널의 다른 스캔 위치에서 반복된다. 예를 들어, 도 9에 도시된 예에서, 커널(K)은 입력 데이터(Xin)의 볼륨 내의 한 코너(corner)에 위치하기 시작할 수 있고, 각 커널 값은 그 커널이 해당 위치에 있을 때 중첩하는 데이터 포인트로 곱해진다. 그런 다음 커널은 볼륨의 길이를 따라 한 단계씩 이동하고, 이번에는 커널을 새로운 위치에 놓고 포인트 승산 및 합계가 다시 수행된다. 커널이 볼륨의 전체 길이를 따라 스캔될 때, 전체 볼륨이 스캔될 때까지 새로운 좌표 위치로 시프트되고 길이 스캔이 다시 수행된다. 각 주어진 위치에서 커널은 출력 데이터의 한 픽셀(하나의 스칼라 값)을 생성한다. 따라서, 출력 데이터(Xout)는 컨볼루션 스캔의 단계 크기에 따라 입력 데이터와 동일하거나 유사한 크기를 가질 것이다.

[0138] 실제로 이것은 더 작은 연산들로 나눌 필요가 있다. 이것은 주어진 위치에 있는 커널로 데이터의 컨볼루션을 다수의 개별적인 벡터 내적과 부분 합으로 나눔으로써 가능하다. 이것은 도 9의 중앙과 우측에 개시되어 있다. 예를 들어, 커널 크기는 $3 \times 3 \times 16$ 픽셀이다("픽셀"은 현재의 논의를 위해 반드시 이미지를 언급하지않음). 즉, $3 \times 3 \times 16$ 개의 사전 결정된 가중치로 구성된다. 이것은 9개의 16-엘리먼트 벡터(CW)(본 명세서에서는 커널(K)의 구성 커널이라고도 함)로 나눌 수 있다. 먼저, 이들 9개의 벡터들 중 하나 및 대응 위치에 있는 입력 데이터 사이에서 내적(즉, 스칼라 곱)을 취하여 부분 합(P)(스칼라)을 얻는다. 그런 다음, 내적은 9개의 벡터 중 다른 벡터 사이에서 취해지고 주어진 제1 부분 합과 누적되어 제2 부분 합(또한 스칼라)을 제공한다. 그런 다음, 상기 내적은 9개의 16-엘리먼트 벡터 중 다른 엘리먼트 벡터들 사이에서 취해지고, 이 결과는 제2 부분 합과 누적되어 제3 부분 합을 제공한다. 일단 이것이 모든 9개의 벡터에 대해 누적되도록 수행되면, 전체 누적 결과는 출력 데이터(Xout)에서 단일 포인트 또는 픽셀의 값을 제공한다. 이것은 커널(K)이 입력 데이터(Xin)의 볼륨에 대해 상이한 위치에 걸쳐 스캐닝에 따라 각 커널 위치에서 반복되어, 출력 데이터(Xout)(커널 위치 당 하나의 출력 데이터 포인트)의 대응하는 볼륨을 제공한다. 실시예에서, 효율적인 방식으로 컨볼루션과 같은 계산을 수행하는 것을 돕기 위해, 프로세서(4)의 명령 세트는 누적 곱("amp") 명령의 형태로 산술 명령을 포함한다. 구문은 다음과 같다.

- [0139] `amp $aDst, $aSrcA, $aSrcB, Phase`
- [0140] 여기서 `$aDst`는 ARF(26A)에서 대상을 지정하는 오퍼랜드를 다시 나타내고, `$aSrcA`, `$aSrcB`는 ARF내의 2개의 소스를 지정하는 오퍼랜드를 나타낸다. "페이즈(Phase)"는 누적 페이즈를 지정하는 즉치 오퍼랜드이다. 예를 들어, 실시예들에서, 페이즈 오퍼랜드는 4개의 가능 페이즈(0...3) 중 하나를 지정하는 2 비트로 형성된다. 실시예에서, 상기 누적 곱(`amp`) 명령은 4개의 반(`half`) 정밀도(16 비트) 부동 소수점 값 중 하나의 벡터를 제1 입력으로 취하고 단(`single`) 정밀도(32 비트) 부동 소수점 값을 제2 입력으로 취하여, 단 정밀도(32 비트) 부동 소수점 값(단 정밀도 입력, 단 정밀도 출력을 나타내는 "siso")을 출력한다는 사실을 참조하여, `f16v4sisoamp`라고 할 수 있다. 실제로 실시예에서, 제2 입력은 출력과 마찬가지로 2개의 단 정밀도 부동 소수점 값들의 쌍(2×32 비트)이다. 그러나, 이들 특정 값의 정밀도 및 엘리먼트의 수는 모든 가능한 실시예를 제한하는 것이 아님을 이해할 것이다.
- [0141] 누적 곱(`amp`)과 같은 하나 이상의 타입의 명령에 대해, 실시예에서 WRF에서의 가중치의 위치는 완전히 암시적이다. 대안적으로 또는 부가적으로, 하나 이상의 다른 타입의 명령은 WRF의 몇몇 상이한 세트 중에서 어느 세트의 가중치를 사용할 것인지를 지정하는 추가 오퍼랜드(미도시))를 취할 수 있다. 예를 들어, 일 예는 후술하는 슬림(`slim`) 컨볼루션("slic") 명령일 수 있다.
- [0142] 앞서 설명한 로드-저장 명령(`ld2xst`)와 결합하여, `amp` 명령과 같은 명령은 메모리로부터의 데이터를 효율적으로 스트리밍하여, 내적 및 부분 합을 수행하고, 부분 합을 다시 메모리로 스트리밍하는데 사용될 수 있다. 예를 들어, 다음과 같이 4개의 명령 쌍의 루프된 시퀀스를 포함하는 프로그램을 고려해 보자.
- [0143] 루프 {
- [0144] `ldx2st Xin-Pin,Pout,Tripacked,Strides; amp Pout,Xin,Pin,Phase=0;`
- [0145] `ldx2st Xin-Pin,Pout,Tripacked,Strides; amp Pout,Xin,Pin,Phase=1;`
- [0146] `ldx2st Xin-Pin,Pout,Tripacked,Strides; amp Pout,Xin,Pin,Phase=2;`
- [0147] `ldx2st Xin-Pin,Pout,Tripacked,Strides; amp Pout,Xin,Pin,Phase=3;`
- [0148] }
- [0149] 다른 예시적인 버전은 다음과 같다.
- [0150] 루프 {
- [0151] `ldx2st Xin-Pin,Pout_A,Tripacked,Strides; amp Pout_A,Xin,Pin,Phase=0;`
- [0152] `ldx2st Xin-Pin,Pout_B,Tripacked,Strides; amp Pout_B,Xin,Pin,Phase=1;`
- [0153] `ldx2st Xin-Pin,Pout_A,Tripacked,Strides; amp Pout_A,Xin,Pin,Phase=2;`
- [0154] `ldx2st Xin-Pin,Pout_B,Tripacked,Strides; amp Pout_B,Xin,Pin,Phase=3;`
- [0155] }
- [0156] 기타
- [0157] 또한, 실시예들에서, 각각의 쌍은 명령 번들, 즉 각각의 파이프 라인들, 예를 들어 메인 및 보조 파이프 라인을 동시에 실행하는 명령 번들이다.
- [0158] 각 명령 쌍은 누적 곱(`amp`) 명령의 인스턴스가 뒤따르는 로드-저장 명령의 인스턴스를 포함한다. `amp` 명령의 소스들은 동일한 또는 선행 명령 쌍에서 로드-저장 명령에 의해 수행된 2개의 로드의 대상(목적지)이다. 2개의 동시 명령의 번들로서 각 쌍을 실행하는 실시예에서, `amp(Xin, Pin)`의 소스들은 이전 번들에서 수행된 로드의 대상들(목적지)과 동일하게 설정된다. `amp` 명령의 대상은 후속 명령 쌍(예를 들어, 다음 번들)에서 로드-저장 명령에 의해 수행되는 하나의 저장(소)의 소스이다. `amp` 명령의 소스들(`$aSrcA`, `$aSrcB`) 중 하나는 입력 데이터(`Xin`)로부터 입력 벡터(`x`)를 취하는데 사용된다. 다른 하나는 부분 합을 취하는데 사용된다. `amp` 명령은 실행될 때, 입력 벡터(`x`)와 가중치 레지스터 파일(WRF, 26W)로부터의 대응하는 가중치 벡터(CW)(이 벡터(CW)는 전체 커널 또는 3D 커널(K)의 구성 커널임)와의 내적을 수행한다. 페이즈 오퍼랜드는 내적 결과를 누적하기 위한 페이즈를 지정한다. 주어진 루프 인스턴스에서 시퀀스의 각 명령 쌍에서, 페이즈 오퍼랜드는 시퀀스의 상이한 연속 페이즈를 지정하는 서로 다른 값으로 설정된다. 페이즈 오퍼랜드에 의해 지정된 시퀀스의 이러한 연속 페이즈에

대해, amp 명령의 효과는 연속적인 내적 각각의 결과를 누적하는 것이다. 누적은 입력 부분 합과 시퀀스의 첫 번째 페이지에서 시작된다.

[0159] 로드-저장 명령의 스트라이드 기능은 프로그램이 MRF(26M)에 포함할 새 메모리 어드레스들을 계산하기 위해 별도의 정수 산술 명령을 사용하지 않고도 각 명령 쌍을 사용하여 메모리 어드레스들을 따라 다음 데이터 부분(piece)으로 자동 시프트하도록 한다. 스트라이드 레지스터(\$mStride)로부터 사전 프로그래밍된 여러 스트라이드들 중 하나를 선택하는 상기 기술된 로드-저장 명령의 기능은 도 9의 예에 의해 도시된 바와 같은 다차원 데이터를 처리하는데 특히 유용하다. 메모리 공간은 1 차원이지만 데이터는 2차원, 3차원 또는 더 많은 차원을 가질 수 있다. 다차원 데이터 공간에 대한 1D 메모리 공간의 맵핑에 따라, 커널(K)이 입력 데이터(Xin)를 통해 스캐닝될 때, 종종 메모리 공간 내의 대응하는 메모리 어드레스를 통해 상이한 크기의 단계들을 만들어야 할 수도 있다. 예를 들어, 커널(K)이 입력 데이터(Xin)의 한 차원을 따라 길이 방향으로 스캔될 때, 각 스캔 위치에 있는 데이터의 메모리 어드레스는 원자 단계(atomic steps)로 증가할 수 있지만, 스캔이 수직 평면에서 스캔될 필요가 있을 때, 입력 데이터(Xin)의 다음 부분 또는 서브 세트(및 제2 입력 및 출력 어드레스에 대해 독립적으로 동일한 프로세스가 수행됨)에 도달하기 위해서는 메모리 어드레스의 상이한 크기 단계(step)가 필요할 수 있다. 스트라이드 레지스터(\$mStride) 또는 스트라이드 레지스터들(예컨대, \$mStrideA, \$mStrideB,...)의 상이한 필드는 유리하게 메모리 공간의 상이한 크기 점프(size jumps)로 사전 프로그래밍될 수 있다. 그런 다음 프로그래머 또는 컴파일러는 스트림(Strimm)을 설정하여, (다음 어드레스 또는 다음 단계 크기를 재계산하기 위해 별도의 정수 산술 명령을 사용하지 않고도) 로드-저장 명령별로 사용할 점프 크기를 선택할 수 있다.

[0160] 실시예에서, amp 명령은 누적 행렬 곱(셈) 명령이다. 이 명령의 연산은 도 10 내지 도 12를 참조하여 설명한다. 도 10에 도시된 바와 같이, amp 명령은 FPU에 의해 실행될 때, (WRF(26W)로부터의) 가중치의 $M \times N$ 행렬과 입력 데이터(Xin)로부터의 N-엘리먼트 입력 벡터와의 승산을 수행한다. 실시예들에서 M=8 및 N=16이다. 행렬의 M개의 행들 각각은 상이한 각각의 커널(K0, K1,...,KM-1)로부터의 구성 벡터에 대응한다. 따라서, amp 명령은 입력 데이터와 병렬로 M개의 상이한 커널(K)의 컨볼루션 성분을 수행하는데 사용된다. 예를 들어, 이들 커널 각각은 입력 데이터와 상이한 각각의 특징(예를 들어, 상이한 에지 또는 모양)의 컨볼루션에 대응할 수 있다. 컨볼루션 신경망의 경우에서 이들은 서로 다른 특징 필터이며, M개의 커널 각각은 입력 계층에서 상이한 특징의 존재를 검출하려고 시도한다. 주어진 커널(K)에 대해, amp 명령은 내적 및 누적만 수행한다. 그러나, 상기 행렬은 다수의 커널이 병렬로 데이터와 효율적으로 컨볼루션될 수 있게 한다.

[0161] N-엘리먼트 입력 벡터는 N1개의 세그먼트로 분할되고, 각 세그먼트는 N2개의 엘리먼트의 서브 벡터를 포함한다(그래서 $N=N1 \times N2$). 실시예들에서 N1=4 및 N2=4이다. amp 명령의 페이지 오퍼랜드는 N1개의 상이한 세그먼트들에 대응하는 N1개의 상이한 가능한 값(0...N1-1) 중 하나를 취한다. 루프된(looped) 시퀀스의 각 인스턴스에서, 각 연속 쌍의 amp 명령은 페이지(0...N1-1) 중 상이한 연속적인 하나를 지정하는 페이지 오퍼랜드의 다른 값을 사용한다. 각 페이지는 N-엘리먼트 입력 벡터의 N1개의 서브 벡터들 중 대응하는 하나와 상기 행렬의 M개의 행 각각의 내적을 수행한다(그래서 M개의 상이한 커널(K) 각각에 대한 계산의 일부). 실행 유닛(18) 내의 FPU(보조 또는 산술 실행 유닛(18A)의 실시예에서)는 M개의 누산기 상태(\$AACC)(스레드 당)를 갖는다. 실시예에서, 이들은 FPU의 내부 상태(57)로서 구현된다. 그러나, 대안적인 구현예에서는 레지스터 파일(26) 중 하나(예를 들어, 각각의 스레드의 ARF(26A))내의 레지스터들로서 구현될 수 있음을 배제하지 않는다. 도시된 예에서 이들은 짝수로 넘버링된 상태(\$AACC[0], \$AACC[2]...\$AACC[14])이고; 홀수로 넘버링된 상태(\$AACC[1], \$AACC[3]...\$AACC[15])는 시퀀스의 서로 다른 루프 사이에서 값들을 전파하는데 사용될 수 있지만, 이는 구현 세부 사항이며 이에 한정되지는 않는다.

[0162] 도 11은 amp 명령의 연산을 개략적으로 나타낸다. 도 12는 일련의 amp 명령의 예를 도시한다. 각각은 전술한 바와같이, 명령 쌍의 대응하는 로드-저장 명령이 선행되는 것으로 이해된다. 도 12에서 Pn은 단-정밀도 입력 부분 합이고, xn은 f16v4 입력 벡터이고, CWm,n은 공통 가중치 상태(\$CWEI_m_n)이며, Rn은 Pn으로 시작하는 연속적인 내적 누적의 최종의 단-정밀도 결과이다.

[0163] 동작시, amp 명령의 (디코딩된) 오퍼코드에 응답하여, 실행 유닛(18)의 FPU(실시예들에서 보조 실행 유닛(18A))는 이하의 연산들을 수행한다.

[0164] ● amp 명령의 소스 오퍼랜드들 중 하나에 의해 지정된 ARF(26A)의 레지스터(들)로부터, M/N1개의 부분 합을 취하여 이들을 일시적으로 다음 루프의 전파기 상태(propagator state)로 놓는다. 이러한 전파기 상태는 FPU의 내부 상태(57)에서 구현될 수 있거나, 대안적인 구현예에서는 레지스터 파일들(26) 중 하나의 다른 레지스터(예를 들어, ARF)일 수 있다. 각 페이지는 M/N1개의 부분 합들의 상이한 서브 세트를 취한다. N1개의 페이지에 걸쳐,

M개의 이러한 부분 합이 전파기 상태로 취해진다. 따라서, $M = 8$ 및 $N1 = 4$ 인 경우, 각각의 amp 명령은 2개의 부분 합(예를 들어, 2개의 f32 값)을 입력으로 취하여 이들을 전파기 상태로 유지하고, 4개의 모든 페이지에 걸쳐 8개의 부분 합이 수신된다.

- [0165] ● amp 명령의 소스 오퍼랜드들 중 하나에 의해 지정된 ARF(26A)의 레지스터(들)로부터, 입력 벡터(xin)의 N2-엘리먼트 세그먼트(서브 벡터)를 취한다. 행렬의 M개의 행들(즉, M개의 커널(K0...KM-1) 각각에 대해, 이것을 WRF로부터의 가중치(CW)의 대응하는 서브 벡터와 내적을 수행한다. 가중치의 서브 벡터가 취해지는 열은 페이지 오퍼랜드의 값에 따라 다르다. 또한, 페이지가 시퀀스의 제1(즉, 초기) 페이지이면, 상기 부분 합들 중 대응하는 하나가 또한 M개의 내적들 각각에 가산된다. 이전 루프의 각 amp 명령이 각각 M/N1개의 부분 합을 취했기 때문에(N1은 페이지의 수이다), 이전 루프는 모든 M개의 부분 합을 현재의 루프에 대해 준비된 전파기 상태로 놓는다.
- [0166] ● M개의 행들 각각에 대해, 위 계산 결과는 M개의 누산기 상태(\$AACC) 중 대응하는 하나에 이미 임의의 값으로 누적(합계)되어, M 결과들(R0...RM-1) 중 대응하는 하나를 생성한다. N1개의 페이지 동안, amp 명령은 전체 N-엘리먼트 입력 벡터와 WRF로부터의 가중치의 대응하는 N-엘리먼트 벡터와의 내적을 수행할 것이다.
- [0167] ● 현재 루프에서, 이전 루프로부터의 결과(R)의 수(M/N1)는 amp 명령의 대상 오퍼랜드로 지정된 ARF의 대상 레지스터(들)로 출력된다. 각 페이지는 M/N1개의 부분 합의 상이한 서브 세트를 출력하고, N1개의 페이지에 걸쳐, M개의 이러한 부분 합은 전파기 상태로 취해진다. 따라서, $M = 8$ 및 $N1 = 4$ 인 실시예에서, 2개의 결과가 amp 명령(예를 들어, 2개의 f32) 당 출력되고, 8개의 결과(R0...R7) 모두가 4개의 모든 페이지에 걸쳐 출력된다.
- [0168] ● 현재 루프의 마지막 페이지 이후(또는 임의의 결과를 오버라이트(덮어쓰기)하기 전에 다음 루프의 제1 페이지 시작시), 현재 루프의 결과(R)은 다음 루프의 출력을 위해 준비된 전파기 상태로 일시적으로 놓인다. 이러한 전파기 상태는 FPU의 내부 상태(57)에서 구현될 수 있거나, 다른 구현 예에서는 레지스터 파일들(26) 중 하나의 다른 레지스터(예를 들어, ARF)일 수 있다.
- [0169] 요약하면, 단일 부분 합은 (1) 제1 내적 결과에 가산되어 누산기 상태로 저장(이전에 보유한 값을 오버라이트함)하고, (2) 3개의 내적 결과들과 누적되어, (3) 임시 전파기 상태로 복사된다(이 경우 해당 값은 향후 amp 명령의 대상 오퍼랜드에 기록된다). 단계(3)에 대한 대안은 결과 부분 합을 누산기에서 곧바로 amp 명령의 대상 오퍼랜드에 기록하는 것이다. 어느 쪽이든, 단계(1)과 (3)은 하드웨어에 중첩되어 4 사이클 루프를 제공할 수 있다.
- [0170] 실시예에서, 누산기 상태는 명시적으로 리셋될 필요가 없다. 각 행렬-벡터 곱셈의 시작점은 메모리에서 읽은 입력 부분 합이므로, 출력 볼륨은 누산기 상태를 재설정하는 대신 컨볼루션 시작시 모든 엘리먼트를 0으로 설정한다. 즉, 실시예들에서 타일(4)은 실제로 1을 0으로 하여 누산기 상태를 모두 초기화하는 것을 허용한다.
- [0171] 현재의 amp 명령의 입력 전파기 상태는 다음 루프에서 사용하기 위해 유지된다. 또한 ARF의 대상 레지스터에 대한 amp 명령에 의한 결과 출력은 현재의 amp의 누적 연산이 적용되기 전의 기존 누산기 상태이며, 이는 현재의 입력을 포함하지 않는 이전의 짧은 누적 시퀀스의 결과이다.
- [0172] 프로그램은 결과(R)가 후속 루프의 부분 합 입력(P)가 되도록 (프로그래머 또는 컴파일러에 의해) 구성된다. 정확한 관계는 프로그래머나 컴파일러에게 달려 있고 데이터 및 커널의 구조에 달려 있다. 일반적으로 부분 합은 0의 초기 값(메모리에)을 가진다.
- [0173] 루프들 간의 지연(lag)으로 인해 시퀀스는 워밍-업 기간(warm-up period)이 필요하다. 제시되는 특정 입력 벡터가 제시되고 4-페이지 계산의 결과가 amp 명령에 의해 반환되는 것 사이에는 대기 시간(latency) 있다. 대기 시간은 고정되어 있으며 첫 번째 M개의 결과가 유효하지 않거나 중요하지 않거나 의미가 없어 워밍-업 코드에서 폐기된다는 것을 의미한다. 유용한 결과가 생성되지 않는 워밍-업 기간 후에, 코드는 반복 루프를 시작하고 의미 있는 결과를 만들어 필요한 메모리 위치에 저장한다. 또한 새로운 입력이 제공되지 않지만 최종 출력 값이 저장되는 내부 루프 이후에는 냉각(cool-down) 기간이 있다. 도 12는 f16v4sisoamp의 워밍-업 기간과 "결과 대기 시간"을 도시한다.
- [0174] 실시예에서, 커널의 수는 $M = 8$ 이고, 입력 벡터의 각 엘리먼트 수는 $N = 16$ 이고 그리고 페이지의 수는 $N1 = 4$ 이다. 입력 벡터의 각 엘리먼트는 f16 값이고, 각 입력 부분 합 및 각 결과는 f32 값이다. 이러한 실시예에서, 위의 프로세스는 명령 쌍 당 2개의 f32 부분 합 및 입력 벡터의 4개의 f16 엘리먼트의 비율로 데이터를 로딩 및 처리하고, 명령 쌍당 2개의 f32 값의 비율로 데이터를 출력 및 저장할 수 있다. 이는 명령 쌍 당 128 비트의 입력과 64 비트의 출력을 제공한다. 이는 2개의 64 비트 폭 로드 포트와 LSU(55)와 데이터 메모리(22) 사이의 하

나의 64 비트 폭 저장 포트에 해당한다(다시 도 4 참조). 또한 어드레스의 3중 패킹으로 인해, 이 프로세스는 MRF(26M)에서 LSU까지의 2 개의 32 비트 폭 포트를 통해 처리되며, 슬라이드를 위해 또 다른 포트가 추가된다. 2개의 64 비트 폭 포트는 ARF(26A)에서 보조 실행 유닛(18A)의 FPU까지 사용된다.

[0175] 가중치(CW)는 공유된 WRF(26W)로부터 취해지며 각 스레드에 대해 동일하다. 신경망과 같은 애플리케이션에서는 (컨볼루션의 일부와 같은) 상이한 데이터로 동일한 가중치 세트를 곱해야할 필요가 있는 시나리오가 많이 있다. 예를 들어, 신경망의 일부 노드는 정확히 동일한 가중치를 가지지만 상이한 연결을 포함할 수 있다. 예를 들어, 각각의 커널(K)($m = 0 \dots M$)은 입력 데이터(예를 들어, 그래픽 픽셀의 영역 또는 볼륨)와 특징(예를 들어, 에지 또는 형상)의 컨볼루션을 나타낼 수 있다. 따라서 이들 커널에 대응하는 워커 스레드는 별도의 가중치가 필요하지 않다. 대신 수퍼바이저가 소유한 가중치 상태의 하나의 복사본만 제공된다. 공유 가중치 레지스터 파일로부터의 공유 스레드 오퍼랜드의 사용은 유리하게도 가중치를 위한 레지스터 공간을 덜 필요로 한다. 도 12는 16-엘리먼트 입력 벡터 각각에 대해 루프가 8개의 출력 값(커널 당 1개의 출력 값)을 생성함을 보여준다. 이 특정 시나리오에서, 출력 값은 각각 32 비트이다. 상기 루프는 4 사이클마다 16개의 입력 값을 처리한다(명령 당 $4 \times f16$ 입력 값). 프로세서(4)는 이러한 입력 속도를 유지하기 위해 4 사이클 루프마다 8×32 비트 값을 생성하는데, 이는 루프 당 4×64 비트 저장 또는 사이클 당 1×64 비트 저장에 해당한다.

[0176] 실시예에서, 홀수 누산기들(\$AACC[1], \$AACC[3], \dots, \$AACC[15])은 결과(Rx)를 스테거(stagger)하고 부분 합 입력(Px)과 동일하게 하는데 사용된다. 부분 입력은 누산기 레지스터(\$AACC[13]와 \$AACC[15])에 2개씩 입력된다. 각 사이클에서 이 값들은 \$AACC[11]와 \$AACC[9]로 포워드되고, 이전의 \$AACC[11]와 \$AACC[9]의 값은 \$AACC[7]와 \$AACC[5]로 이동된다. 대상(목적지)에 도달하면 이것들은 누적 연산을 위한 준비가 된 짝수 번호의 누산기들에 전파된다.

[0177] 도 13 및 도 14는 본 명세서에 기술된 타입의 로드-저장 명령, 즉 각각의 스레드의 ARF(26A)를 통해 스트리밍된 입력 데이터를 공통 가중치 레지스터 파일(WRF)(26W)로부터의 2D 가중치 세트로 컨볼루션하는 컨볼루션 명령으로부터 이익을 얻을 수 있는 다른 타입의 산술 명령을 도시한다. 실시예에서, 이 명령은 반 정밀 부동 소수점 벡터 슬라이스 컨볼루션(slic 명령), 예를 들어 f16 값의 4-엘리먼트 서브 벡터상에서 작동한다는 사실을 나타내는 "f16v4slic"의 형태를 취할 수 있다. 그것은 하나의 반 정밀도와 하나의 단 정밀도 입력 및 단 정밀도 출력을 가지고 있다. slic은 유사하며 amp와 동일한 하드웨어를 사용한다. 그러나, 이 명령은 8개의 커널 각각에 대해 $1 \times 1 \times 16$ 누적 내적이 아닌 2개의 커널 각각에 대해 (진성(true) 컨볼루션인) $1 \times 4 \times 4$ 컨볼루션을 수행한다. 실제로, 실시예에서, N이 2, 3 또는 4인 $1 \times N \times 4$ 컨볼루션을 수행하도록 구성될 수 있다.

[0178] 누적 행렬 곱("amp") 및 슬라이스 컨볼루션("slic") 명령은 워커 컨텍스트 전체에서 공유하는 가중치가 적절한 시나리오에서 고성능 곱셈 누적 시퀀스를 용이하게 한다. 실시예에서, amp 및 slic 명령은 단-정밀도 및 반-정밀도 숫자 포맷 모두에서 지원될 수 있으며, 동일한 기본 방식으로 작동한다. (WRF(26W)내의 공유 가중치를 포함하는) 공통 계산 구성 상태는 수퍼바이저 컨텍스트에 의해 먼저 초기화된다. 그런 다음 입력 데이터의 두 스트림, 즉 입력 활성화(픽셀) 데이터 및 후속 곱셈-누적 시퀀스의 시작 값을 지정하는 부분-합 값(컨볼루션의 경우 부분적으로 계산된 사전 활성화)이 처리된다. 출력 데이터의 단일 스트림, 즉 결과인 누적된 부분 합 값이 생성된다. 최종 부분-합 결과가 출력으로 제시되기 전에, 각 부분-합 입력 값은 곱셈-누적 연산의 고정 길이 시퀀스에 적용된다. 계산 엔진에 의해 수행된 많은 내적 및 누적 연산은 병렬로 발생한다. 첫 번째 피승수는 입력 데이터 스트림에 의해 제공되고 두 번째는 공통 계산 구성 상태에 의해 제공된다.

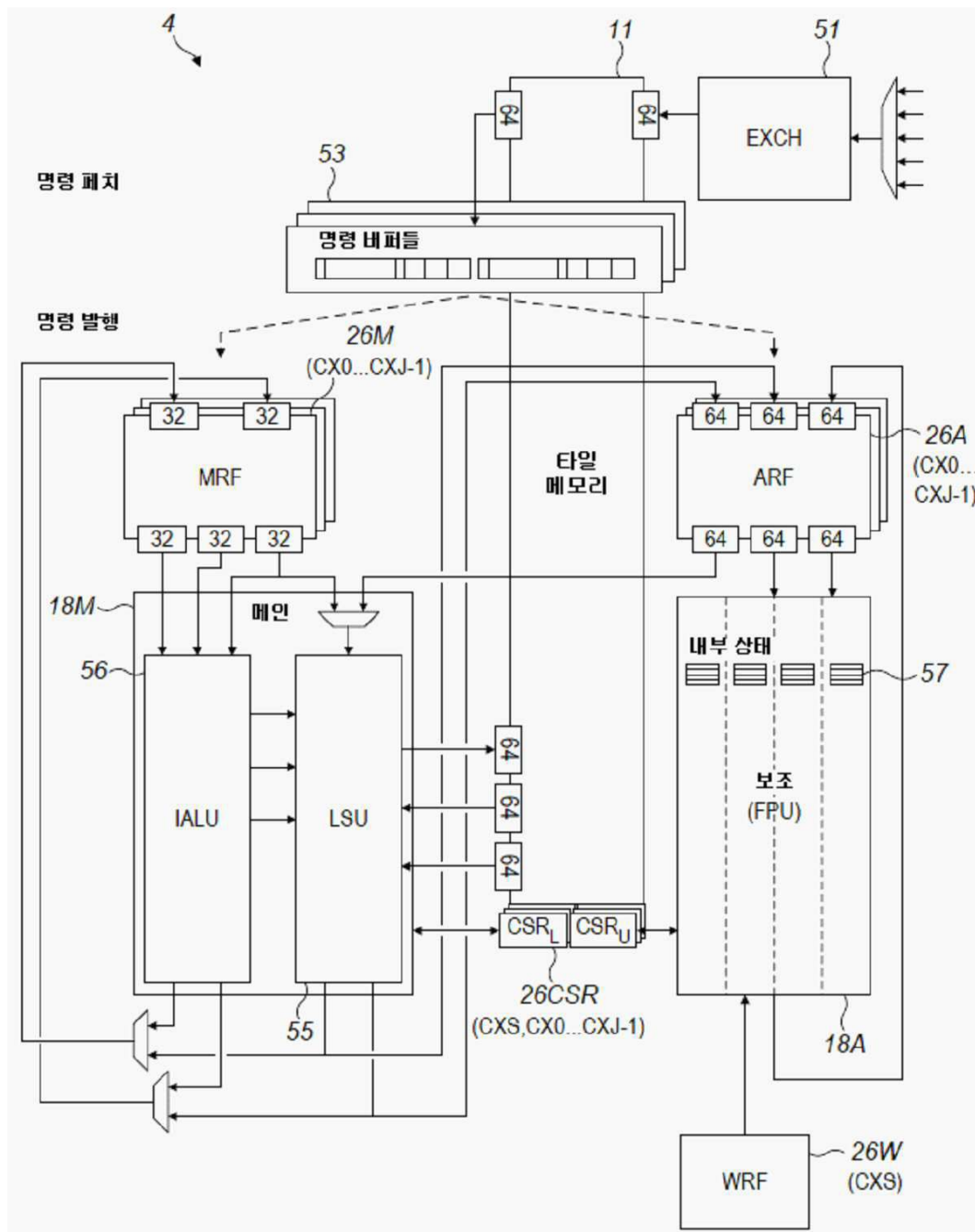
[0179] 다음 표는 일부 실시예에서 프로세서의 명령 세트에 포함될 수 있는 임의의 및 전부의 예시적인 amp 및 slic 명령 변수를 나열한다.

명령	입력 데이터 포맷	입력 부분-합 포맷	출력 부분-합 포맷
f16v4sisoamp	반-정밀도(v4)	단-정밀도(v2)	단-정밀도(v2)
f16v4hihoamp	반-정밀도(v4)	반-정밀도(v2)	반-정밀도(v2)
f16v4sihoamp	반-정밀도(v4)	단-정밀도(v2)	반-정밀도(v2)
f32sisoamp	단-정밀도(스칼라)	단-정밀도(v2)	단-정밀도(v2)
f16v4sisoslic	반-정밀도(v4)	단-정밀도(v2)	단-정밀도(v2)
f16v4hihoslic	반-정밀도(v4)	반-정밀도(v2)	반-정밀도(v2)
f16v4sihoslic	반-정밀도(v4)	단-정밀도(v2)	반-정밀도(v2)
f32sisoslic	단-정밀도(스칼라)	단-정밀도(v2)	단-정밀도(v2)

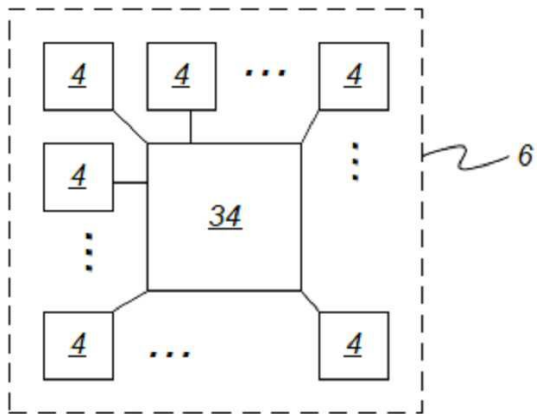
[0180]

- [0182] 주: 일부 타입의 명령에 대해, 가중치들의 위치는 완전히 암시적이지만, 다른 타입의 명령에 대해 가중치들은 WRF에서 사용할 여러 상이한 세트 중 하나를 선택하는지 오퍼랜드를 취한다. 예를 들어 f16v4sisoslic 명령은 f16v4sisoamp의 가중치 상태의 4분의 1만 필요하다. 결과적으로 소프트웨어는 최대 4 세트의 가중치를 사전로드할 수 있다. 가중치 세트 선택은 즉치 오퍼랜드의 2 비트로 지정된다. 반면에 f16v4sisoamp의 경우 이것은 전체 가중치 상태를 취하므로 그 안에 선택은 없다
- [0183] 상기 실시예는 단지 예로서 기술되었다는 것을 알 수 있을 것이다.
- [0184] 예를 들어, 본 발명의 범위는 슈퍼바이저 스레드에 대해 별도의 컨텍스트가 제공되거나 슈퍼바이저 스레드가 슬롯에서 실행된 다음 해당 슬롯을 워커에게 양도하는 전술한 아키텍처이다. 슈퍼바이저는 대신 범용 컨텍스트를 사용할 수 있다. 또는 예를 들어 다른 배치에서 슈퍼바이저는 전용 슬롯에서 실행될 수 있다. 또한 구현은 슈퍼바이저 역할을 하는 특정 스레드 또는 실제로는 멀티-스레딩에 국한되지 않는다. 본 명세서에 개시된 기술은 비-멀티 스레드 실행을 사용하지 않는 프로세서상에서도 사용될 수 있다. 또한, 본 발명의 범위는 프로세서(4)가 타일 어레이내의 타일인 것으로 제한되지 않는다. 대안적인 실시예에서, 프로세서(4)는 예를 들어 독립형 프로세서 또는 단일-칩 프로세서 일 수 있다.
- [0185] 실행 유닛(18)의 구현은 별개의 메인 실행 유닛(18M) 및 보조 실행 유닛(18A) 즉 별도의 MRF와 ARF로 제한되지 않는다. 또한 별도의 WRF를 사용하는 것으로 제한되지 않는다. 일반적으로 레지스터들은 하나 이상의 레지스터 파일에 있을 수 있으며, 상이한 메모리 액세스 및 산술 연산간에 공유되거나 분리될 수 있다.
- [0186] 개시된 로드-저장 명령이 사용될 수 있는 산술 명령은 행렬 곱 또는 컨볼루션 명령으로 제한되지 않는다. 보다 일반적으로, 적어도 2개의 입력 오퍼랜드를 취하여 적어도 하나의 결과를 출력하는 임의의 타입의 산술 명령 또는 산술 명령 조합에 대해, 로드-저장 명령의 인스턴스는 또한 메모리에서 데이터를 스트리밍하고 결과를 메모리로 다시 스트리밍하는데 사용될 수 있다. 예를 들어, 로드-저장 명령 타입은 누산이 개별 명령 또는 범용 코드에서 수행되는 비-누산 행렬 곱 명령 또는 컨볼루션 이외의 다른 행렬 곱셈의 시퀀스와 함께 사용될 수 있는데, 여기서 이전 곱의 결과는 반드시 후속 곱의 입력은 아니다. 또한 로드 저장은 벡터 내적 곱 또는 누적 벡터 내적 곱 명령과 함께 사용될 수 있다. 즉, 산술 명령은 M개의 다른 커널을 병렬로 적용하지 않아도 된다.
- [0187] 또한, 본 발명의 범위는 예로서 전술한 특정 레지스터 크기, 포트 비트 폭, 포트 수, 값 정밀도, 벡터 크기 또는 행렬 크기에 제한되지 않는다. 다른 레지스터 및 포트 비트 폭, 값의 다른 정밀도 및 다른 벡터 또는 행렬 크기(예컨대, 엘리먼트 수의 관점에서)가 가능하다. 또한, 다른 구현예에서, 2개의 레지스터로의 2개의 로드 및 하나의 저장 어드레스의 다른 패키징이 가능한데, 예를 들어 2개의 16 비트 레지스터에 3개의 10 비트 어드레스 또는 2개의 64 비트 레지스터에 3개의 42 비트 어드레스가 있다. 또한 로드 및 저장 어드레스는 반드시 동일한 길이일 필요는 없다. 또한, 스트라이드 레지스터로의 스트라이드 값의 다른 패키징이 다른 구현예에서, 예를 들어 32 비트 레지스터내의 4개의 8 비트 스트라이드 필드가 가능하다.
- [0188] 또 다른 변형예에서, 본 발명의 적용 가능성은 이미지 처리에 한정되지 않는다. 본 명세서에서 사용되는 용어 "커널"은 벡터 승산, 행렬 승산 또는 컨볼루션, 또는 다른 것(예를 들어, 상관)과 같은 연산들을 포함하는 임의의 프로세스의 일부로서 적용되는 임의의 가중치 행렬을 의미할 수 있다. 또한 애플리케이션 기능은 3D 데이터 볼륨에 제한되지 않는다. 입력 데이터(및 커널)는 선형 또는 2D이거나 더 큰 다차원(>3 독립 변수 또는 자유도)을 가질 수 있다. 또한, 본 발명의 범위는 기계 학습 애플리케이션에 한정되지 않는다. 벡터 승산, 행렬 승산 및/또는 컨볼루션 등을 포함하는 프로세스들을 수행하는 것이 바람직할 수 있는 많은 다른 애플리케이션이 있다.
- [0189] 개시된 기술의 다른 변형 예 또는 사용 예는 본 명세서의 개시 내용이 일단 부여되면 당업자에게 명백해질 수 있다. 발명의 범위는 개시된 실시예들에 의해 제한되지 않고 첨부된 청구항들에 의해서만 제한된다.

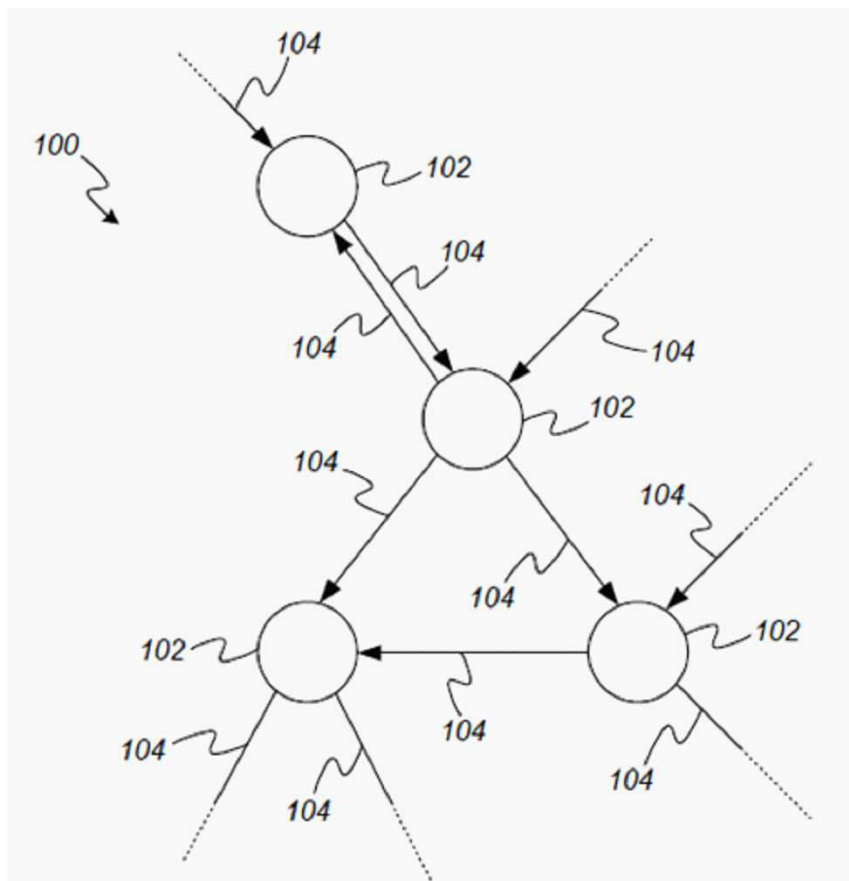
도면4



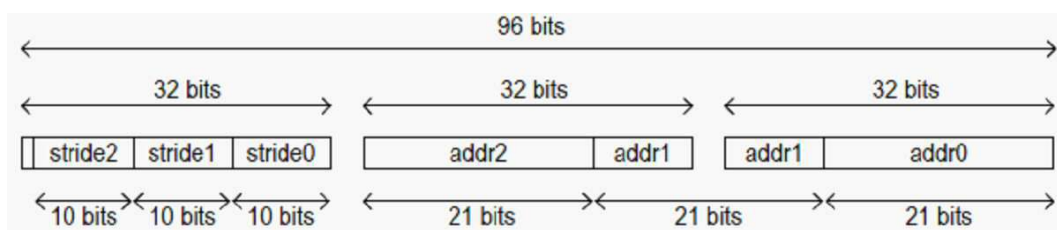
도면5



도면6



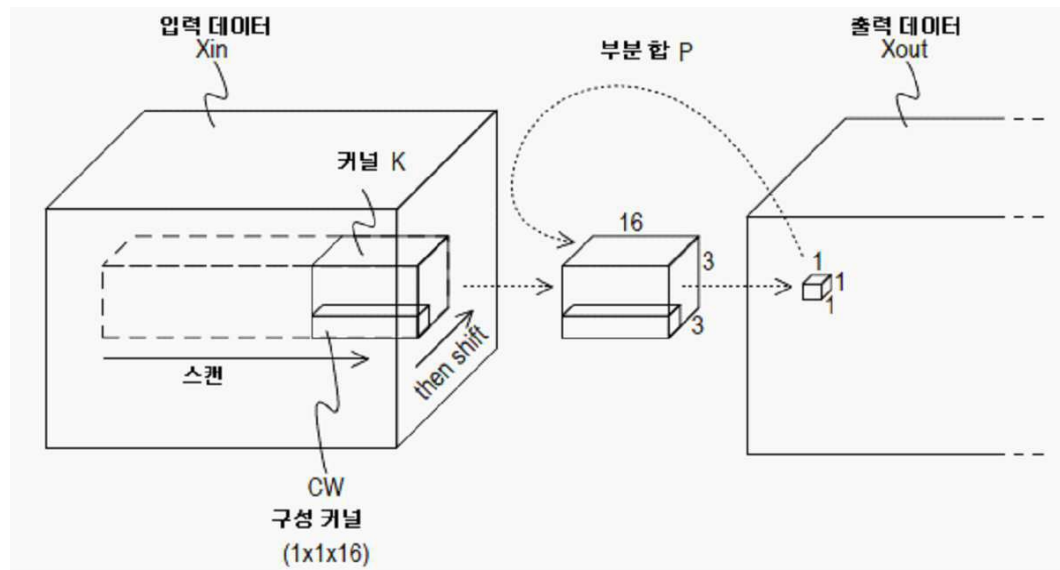
도면7



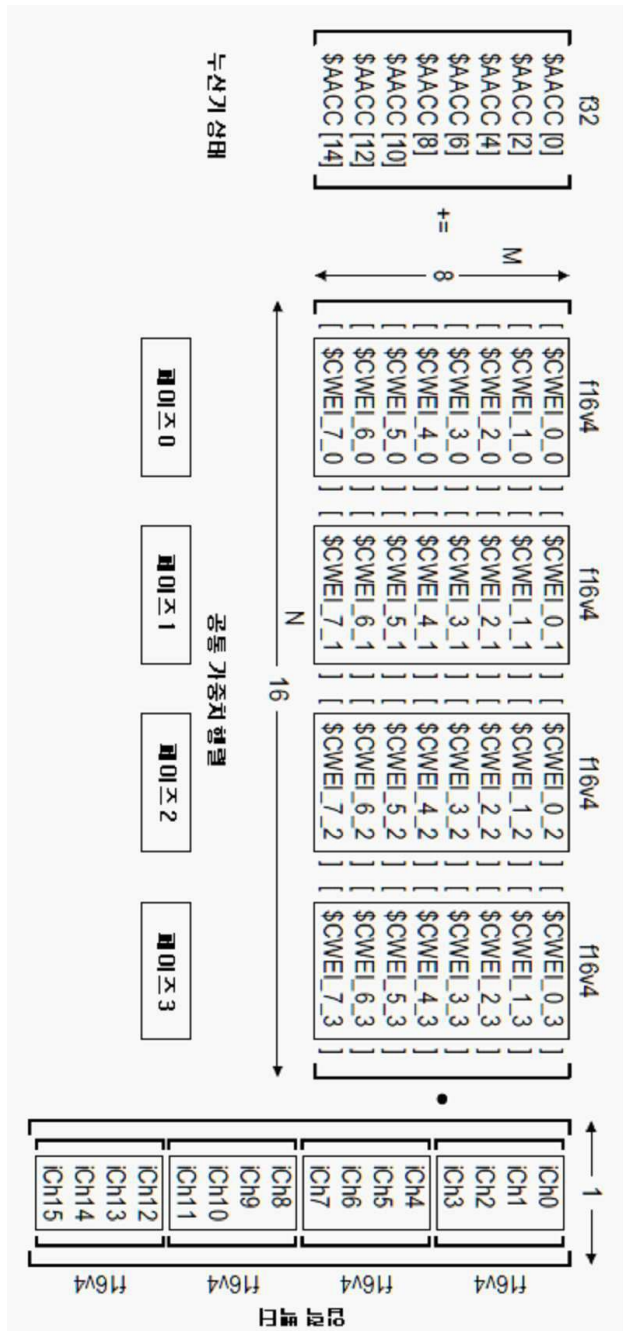
도면8

$\$mStrideA$	strideA2	strideA1	strideA0
$\$mStrideB$	strideB2	strideB1	strideB0
$\$mStrideC$	strideC2	strideC1	strideC0
$\vdots$	$\vdots$		

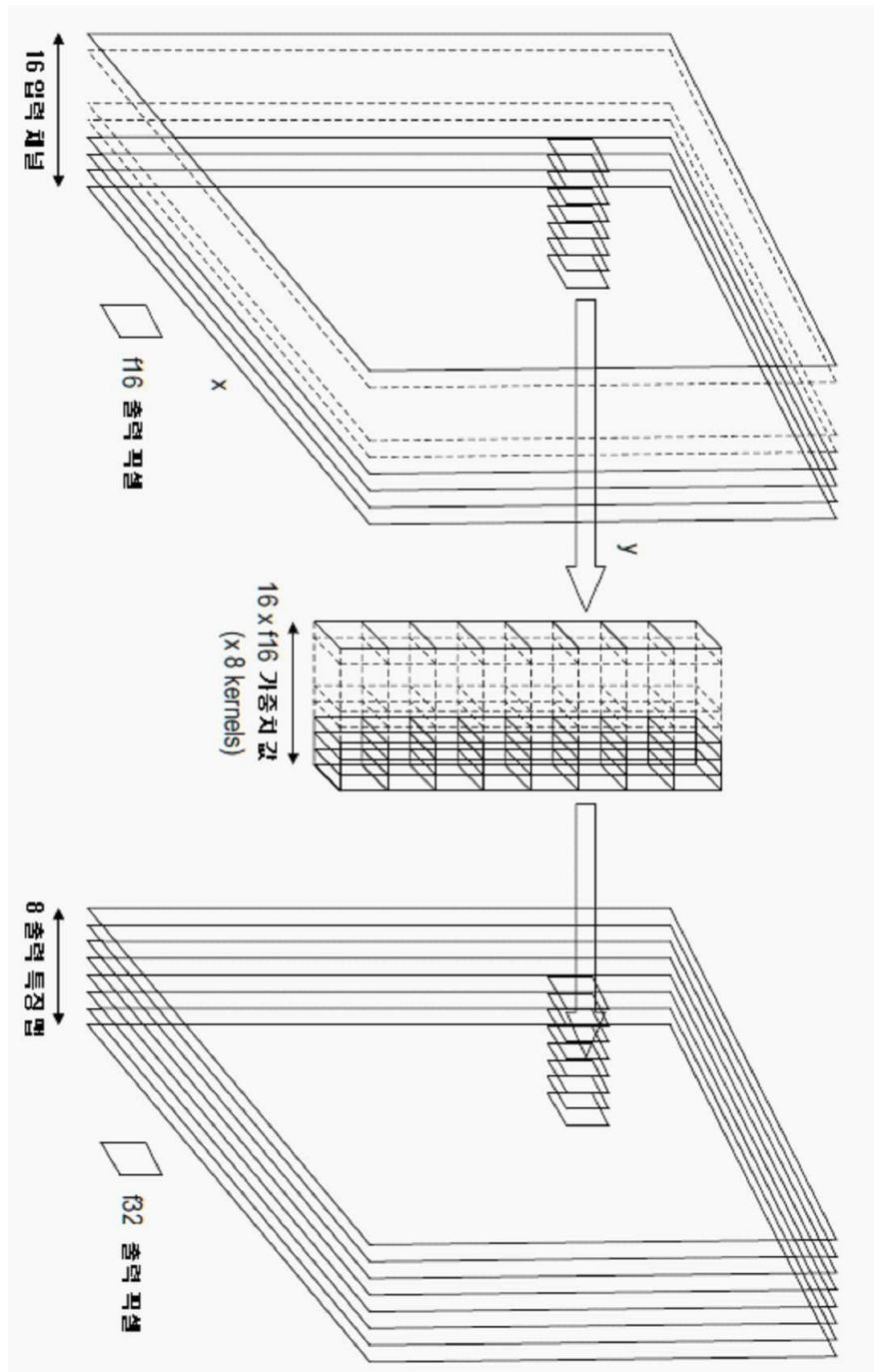
도면9



도면10



도면11



도면12

\$aSrc0[1]	\$AACc[14]	\$AACc[12]	\$AACc[10]	\$AACc[8]
$0^3P_0, P_1$	-	-	-	-
$0P_2, P_3$	-	-	-	[WARM UP]
$0P_4, P_5$	-	-	-	-
$0P_6, P_7$	-	-	-	-
X_0P_8, P_9	$R_7=X_0 \cdot CW_{7,0} + P_7$	$R_8=X_0 \cdot CW_{6,0} + P_6$	$R_5=X_0 \cdot CW_{5,0} + P_5$	$R_4=X_0 \cdot CW_{4,0} + P_4$
X_1P_{10}, P_{11}	$R_7=X_1 \cdot CW_{7,1}$	$R_6=X_1 \cdot CW_{6,1}$	$R_5=X_1 \cdot CW_{5,1}$	$R_4=X_1 \cdot CW_{4,1}$
X_2P_{12}, P_{13}	$R_7=X_2 \cdot CW_{7,2}$	$R_6=X_2 \cdot CW_{6,2}$	$R_5=X_2 \cdot CW_{5,2}$	$R_4=X_2 \cdot CW_{4,2}$
X_3P_{14}, P_{15}	$R_7=X_3 \cdot CW_{7,3}$	$R_6=X_3 \cdot CW_{6,3}$	$R_5=X_3 \cdot CW_{5,3}$	$R_4=X_3 \cdot CW_{4,3}$
X_4P_{16}, P_{17}	$R_{15}=X_4 \cdot CW_{7,0} + P_{15}$	$R_{14}=X_4 \cdot CW_{6,0} + P_{14}$	$R_{13}=X_4 \cdot CW_{5,0} + P_{13}$	$R_{12}=X_4 \cdot CW_{4,0} + P_{12}$
X_5P_{18}, P_{19}	$R_{15}=X_5 \cdot CW_{7,1}$	$R_{14}=X_5 \cdot CW_{6,1}$	$R_{13}=X_5 \cdot CW_{5,1}$	$R_{12}=X_5 \cdot CW_{4,1}$
X_6P_{20}, P_{21}	$R_{15}=X_6 \cdot CW_{7,2}$	$R_{14}=X_6 \cdot CW_{6,2}$	$R_{13}=X_6 \cdot CW_{5,2}$	$R_{12}=X_6 \cdot CW_{4,2}$
X_7P_{22}, P_{23}	$R_{15}=X_7 \cdot CW_{7,3}$	$R_{14}=X_7 \cdot CW_{6,3}$	$R_{13}=X_7 \cdot CW_{5,3}$	$R_{12}=X_7 \cdot CW_{4,3}$
X_8P_{24}, P_{25}	$R_{31}=X_8 \cdot CW_{7,0} + P_{23}$	$R_{22}=X_8 \cdot CW_{6,0} + P_{22}$	$R_{21}=X_8 \cdot CW_{5,0} + P_{21}$	$R_{20}=X_8 \cdot CW_{4,0} + P_{20}$
X_9P_{26}, P_{27}	$R_{23}=X_9 \cdot CW_{7,1}$	$R_{22}=X_9 \cdot CW_{6,1}$	$R_{21}=X_9 \cdot CW_{5,1}$	$R_{20}=X_9 \cdot CW_{4,1}$
$X_{10}P_{28}, P_{29}$	$R_{23}=X_{10} \cdot CW_{7,2}$	$R_{22}=X_{10} \cdot CW_{6,2}$	$R_{21}=X_{10} \cdot CW_{5,2}$	$R_{20}=X_{10} \cdot CW_{4,2}$
$X_{11}P_{30}, P_{31}$	$R_{23}=X_{11} \cdot CW_{7,3}$	$R_{22}=X_{11} \cdot CW_{6,3}$	$R_{21}=X_{11} \cdot CW_{5,3}$	$R_{20}=X_{11} \cdot CW_{4,3}$
$X_{12}P_{32}, P_{33}$	$R_{31}=X_{12} \cdot CW_{7,0} + P_{31}$	$R_{30}=X_{12} \cdot CW_{6,0} + P_{30}$	$R_{29}=X_{12} \cdot CW_{5,0} + P_{29}$	$R_{28}=X_{12} \cdot CW_{4,0} + P_{28}$

P_n 은 단-정밀도 입력 부분-원 n

X_n 은 116v4 입력 벡터

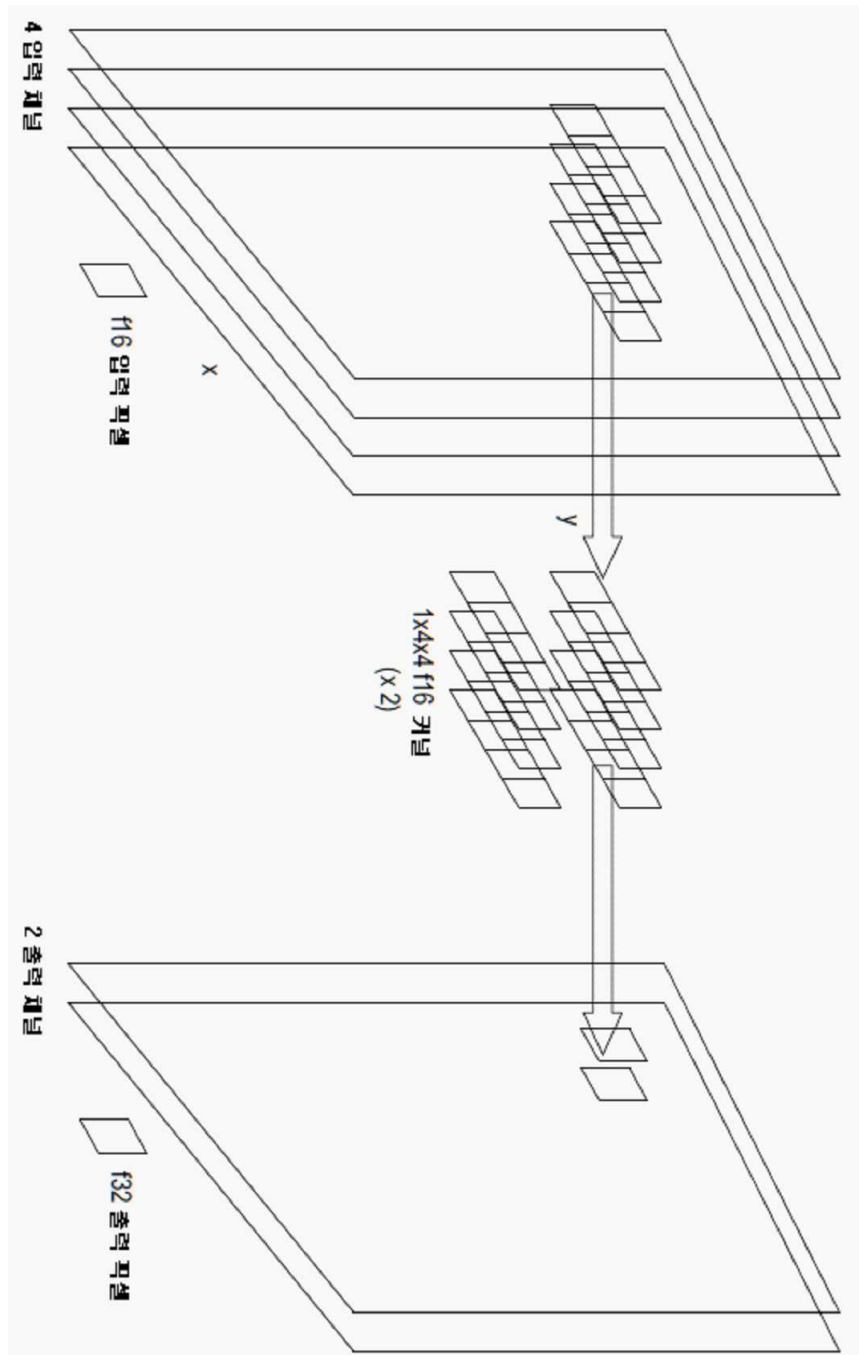
$CW_{m,n}$ 은 공통 가중치 상배 \$CWELm,n

R_n 은 P_n 으로 시작하는 연속적인내적 누적의 최종 단-정밀도 결과

도면12a

\$AACC[6]	\$AACC[4]	\$AACC[2]	\$AACC[0]	\$aDst0
-	-	-	-	-
PERIOD]	-	-	-	-
-	-	-	-	-
-	-	-	-	-
$R_3 = x_0 \cdot CW_{3,0} + P_3$	$R_2 = x_0 \cdot CW_{2,0} + P_2$	$R_1 = x_0 \cdot CW_{1,0} + P_1$	$R_0 = x_0 \cdot CW_{0,0} + P_0$	-
$R_3 = x_1 \cdot CW_{3,1}$	$R_2 = x_1 \cdot CW_{2,1}$	$R_1 = x_1 \cdot CW_{1,1}$	$R_0 = x_1 \cdot CW_{0,1}$	-
$R_3 = x_2 \cdot CW_{3,2}$	$R_2 = x_2 \cdot CW_{2,2}$	$R_1 = x_2 \cdot CW_{1,2}$	$R_0 = x_2 \cdot CW_{0,2}$	-
$R_3 = x_3 \cdot CW_{3,3}$	$R_2 = x_3 \cdot CW_{2,3}$	$R_1 = x_3 \cdot CW_{1,3}$	$R_0 = x_3 \cdot CW_{0,3}$	-
$R_{11} = x_4 \cdot CW_{3,0} + P_{11}$	$R_{10} = x_4 \cdot CW_{2,0} + P_{10}$	$R_9 = x_4 \cdot CW_{1,0} + P_9$	$R_8 = x_4 \cdot CW_{0,0} + P_8$	R_0, R_1
$R_{11} = x_5 \cdot CW_{3,1}$	$R_{10} = x_5 \cdot CW_{2,1}$	$R_9 = x_5 \cdot CW_{1,1}$	$R_8 = x_5 \cdot CW_{0,1}$	R_2, R_3
$R_{11} = x_6 \cdot CW_{3,2}$	$R_{10} = x_6 \cdot CW_{2,2}$	$R_9 = x_6 \cdot CW_{1,2}$	$R_8 = x_6 \cdot CW_{0,2}$	R_4, R_5
$R_{11} = x_7 \cdot CW_{3,3}$	$R_{10} = x_7 \cdot CW_{2,3}$	$R_9 = x_7 \cdot CW_{1,3}$	$R_8 = x_7 \cdot CW_{0,3}$	R_6, R_7
$R_{19} = x_8 \cdot CW_{3,0} + P_{19}$	$R_{18} = x_8 \cdot CW_{2,0} + P_{18}$	$R_{17} = x_8 \cdot CW_{1,0} + P_{17}$	$R_{16} = x_8 \cdot CW_{0,0} + P_{16}$	R_8, R_9
$R_{19} = x_9 \cdot CW_{3,1}$	$R_{18} = x_9 \cdot CW_{2,1}$	$R_{17} = x_9 \cdot CW_{1,1}$	$R_{16} = x_9 \cdot CW_{0,1}$	R_{10}, R_{11}
$R_{19} = x_{10} \cdot CW_{3,2}$	$R_{18} = x_{10} \cdot CW_{2,2}$	$R_{17} = x_{10} \cdot CW_{1,2}$	$R_{16} = x_{10} \cdot CW_{0,2}$	R_{12}, R_{13}
$R_{19} = x_{11} \cdot CW_{3,3}$	$R_{18} = x_{11} \cdot CW_{2,3}$	$R_{17} = x_{11} \cdot CW_{1,3}$	$R_{16} = x_{11} \cdot CW_{0,3}$	R_{14}, R_{15}
$R_{27} = x_{12} \cdot CW_{3,0} + P_{27}$	$R_{26} = x_{12} \cdot CW_{2,0} + P_{26}$	$R_{25} = x_{12} \cdot CW_{1,0} + P_{25}$	$R_{24} = x_{12} \cdot CW_{0,0} + P_{24}$	R_{16}, R_{17}

도면13



도면14

\$aSrc[0]1	\$AAC[14]	\$AAC[10]	\$AAC[6]	\$AAC[2]
$x_0 P_0P_1$	-	$R_1=x_0.CW_{5,0}+P_1$	-	-
$x_1 P_2P_3$	-	$R_3=x_1.CW_{5,0}+P_3$	$R_1=x_1.CW_{3,0}$	-
$x_2 P_4P_5$	-	$R_5=x_2.CW_{5,0}+P_5$	$R_3=x_2.CW_{3,0}$	$R_1=x_2.CW_{1,0}$
$x_3 P_6P_7$	-	$R_7=x_3.CW_{5,0}+P_7$	$R_5=x_3.CW_{3,0}$	$R_3=x_3.CW_{1,0}$
$x_4 P_8P_9$	-	$R_9=x_4.CW_{5,0}+P_9$	$R_7=x_4.CW_{3,0}$	$R_5=x_4.CW_{1,0}$
$x_5 P_{10}P_{11}$	-	$R_{11}=x_5.CW_{5,0}+P_{11}$	$R_9=x_5.CW_{3,0}$	$R_7=x_5.CW_{1,0}$
$x_6 P_{12}P_{13}$	-	$R_{13}=x_6.CW_{5,0}+P_{13}$	$R_{11}=x_6.CW_{3,0}$	$R_9=x_6.CW_{1,0}$

Table 3. 156: f16v4sisosic, 1x4 example sequence

\$aSrc[0]1	\$AAC[14]	\$AAC[10]	\$AAC[6]	\$AAC[2]
$x_0 P_0P_1$	$R_1=x_0.CW_{7,0}+P_1$	-	-	-
$x_1 P_2P_3$	$R_3=x_1.CW_{7,0}+P_3$	$R_1=x_1.CW_{5,0}$	-	-
$x_2 P_4P_5$	$R_5=x_2.CW_{7,0}+P_5$	$R_3=x_2.CW_{5,0}$	$R_1=x_2.CW_{3,0}$	-
$x_3 P_6P_7$	$R_7=x_3.CW_{7,0}+P_7$	$R_5=x_3.CW_{5,0}$	$R_3=x_3.CW_{3,0}$	$R_1=x_3.CW_{1,0}$
$x_4 P_8P_9$	$R_9=x_4.CW_{7,0}+P_9$	$R_7=x_4.CW_{5,0}$	$R_5=x_4.CW_{3,0}$	$R_3=x_4.CW_{1,0}$
$x_5 P_{10}P_{11}$	$R_{11}=x_5.CW_{7,0}+P_{11}$	$R_9=x_5.CW_{5,0}$	$R_7=x_5.CW_{3,0}$	$R_5=x_5.CW_{1,0}$
$x_6 P_{12}P_{13}$	$R_{13}=x_6.CW_{7,0}+P_{13}$	$R_{11}=x_6.CW_{5,0}$	$R_9=x_6.CW_{3,0}$	$R_7=x_6.CW_{1,0}$

P_i 은 단-정밀도 입력 부분-합 n

x_n 은 f16v4 입력 벡터

$CW_{m,n}$ 은 공통 기중치 상배 $\$CWELm,n$

R_n 은 P_n 으로 시작하는 연속적인 내적 누적의 최종 단-정밀도 결과

도면14a

\$AACC[12]	\$AACC[8]	\$AACC[4]	\$AACC[0]	\$aDsIO
-	$R_0 = x_0.CW_{4,0} + P_0$	-	-	-
-	$R_2 = x_1.CW_{4,0} + P_2$	$R_0 = x_1.CW_{2,0}$	-	-
-	$R_4 = x_2.CW_{4,0} + P_4$	$R_2 = x_2.CW_{2,0}$	$R_0 = x_2.CW_{0,0}$	-
-	$R_6 = x_3.CW_{4,0} + P_6$	$R_4 = x_3.CW_{2,0}$	$R_2 = x_3.CW_{0,0}$	R_0, R_1
-	$R_8 = x_4.CW_{4,0} + P_8$	$R_6 = x_4.CW_{2,0}$	$R_4 = x_4.CW_{0,0}$	R_2, R_3
-	$R_{10} = x_5.CW_{4,0} + P_{10}$	$R_8 = x_5.CW_{2,0}$	$R_6 = x_5.CW_{0,0}$	R_4, R_5
-	$R_{12} = x_6.CW_{4,0} + P_{12}$	$R_{10} = x_6.CW_{2,0}$	$R_8 = x_6.CW_{0,0}$	R_6, R_7
\$AACC[12]	\$AACC[8]	\$AACC[4]	\$AACC[0]	\$aDsIO
$R_0 = x_0.CW_{6,0} + P_0$	-	-	-	-
$R_2 = x_1.CW_{6,0} + P_2$	$R_0 = x_1.CW_{4,0}$	-	-	-
$R_4 = x_2.CW_{6,0} + P_4$	$R_2 = x_2.CW_{4,0}$	$R_0 = x_2.CW_{2,0}$	-	-
$R_6 = x_3.CW_{6,0} + P_6$	$R_4 = x_3.CW_{4,0}$	$R_2 = x_3.CW_{2,0}$	$R_0 = x_3.CW_{0,0}$	-
$R_8 = x_4.CW_{6,0} + P_8$	$R_6 = x_4.CW_{4,0}$	$R_4 = x_4.CW_{2,0}$	$R_2 = x_4.CW_{0,0}$	R_0, R_1
$R_{10} = x_5.CW_{6,0} + P_{10}$	$R_8 = x_5.CW_{4,0}$	$R_6 = x_5.CW_{2,0}$	$R_4 = x_5.CW_{0,0}$	R_2, R_3
$R_{12} = x_6.CW_{6,0} + P_{12}$	$R_{10} = x_6.CW_{4,0}$	$R_8 = x_6.CW_{2,0}$	$R_6 = x_6.CW_{0,0}$	R_4, R_5