(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2022-0017944
(43) 공개일자 2022년02월14일

(51) 국제특허분류(Int. Cl.)
*G06F 7/575* (2006.01) *G06F 15/80* (2006.01)
*G06F 5/06* (2006.01) *G06F 7/544* (2017.01)
*G06N 3/063* (2006.01)
(52) CPC특허분류
*G06F 7/575* (2013.01)
*G06F 15/8092* (2013.01)
(21) 출원번호 10-2021-7042903(분할)
(22) 출원일자(국제) 2018년06월20일
심사청구일자 2022년01월26일
(62) 원출원 특허 10-2020-7004715
원출원일자(국제) 2018년06월20일
심사청구일자 2020년02월18일
(85) 번역문제출일자 2021년12월28일
(86) 국제출원번호 PCT/US2018/038618
(87) 국제공개번호 WO 2019/022872
국제공개일자 2019년01월31일
(30) 우선권주장
15/710,433 2017년09월20일 미국(US)
*(뒷면에 계속)*

(71) 출원인
**테슬라, 인크.**
미합중국 캘리포니아주 94304, 팔로 알토, 디어 크릭 로드 3500
(72) 발명자
**다스 사르마, 데비잇**
미국 94304 캘리포니아주 팔로 알토 디어 크릭 로드 3500
**텔프스, 에밀**
미국 94304 캘리포니아주 팔로 알토 디어 크릭 로드 3500
**배넌, 피터, 조셉**
미국 94304 캘리포니아주 팔로 알토 디어 크릭 로드 3500
(74) 대리인
**특허법인 무한**

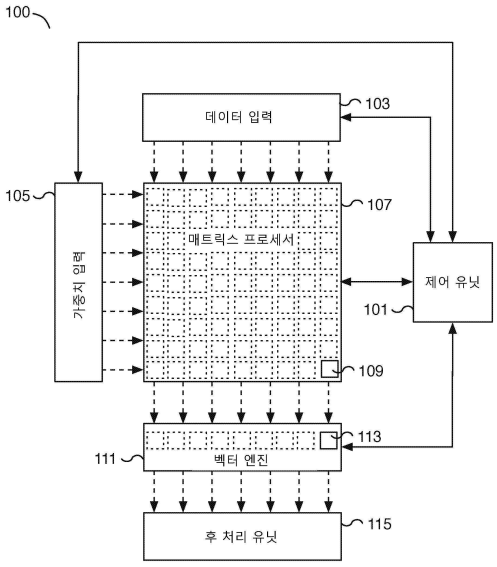
전체 청구항 수 : 총 1 항

(54) 발명의 명칭 **벡터 계산 유닛**

## (57) 요 약

마이크로 프로세서 시스템은 계산 어레이 및 벡터 계산 유닛을 포함한다. 계산 어레이는 복수의 계산 유닛을 포함한다. 벡터 계산 유닛은 계산 어레이와 통신하고 복수의 처리 엘리먼트(processing element)을 포함한다. 처리 엘리먼트는 계산 어레이로부터 출력 데이터 엘리먼트를 수신하고 수신된 출력 데이터 엘리먼트를 병렬로 처리하도록 구성된다.

*대 표 도* - 도1



(52) CPC특허분류
*G06F 5/06* (2021.08)
*G06F 7/5443* (2013.01)
*G06N 3/063* (2013.01)

(30) 우선권주장
15/920,156 2018년03월13일 미국(US)
62/625,251 2018년02월01일 미국(US)
62/536,399 2017년07월24일 미국(US)

# 명 세 서

## 청구범위

### 청구항 1

마이크로 프로세서 시스템에 있어서,

복수의 계산 유닛들을 포함하는 계산 어레이; 및

상기 계산 어레이와 통신하는 벡터 계산 유닛

을 포함하고,

상기 계산 유닛들은,

복수의 레인들로 그룹화되고,

각 레인은,

하나의 FIFO 큐를 형성하는 열(column)로 배치된 상기 계산 유닛들의 서브 세트를 포함하고,

상기 FIFO 큐들은,

병렬적으로 동작하고,

상기 계산 어레이를 통하여 수신된 입력을 상기 FIFO 큐들을 통하여 상기 벡터 계산 유닛으로 시프트하는,

마이크로 프로세서 시스템.

## 발명의 설명

### 기 술 분 야

[0001] 본 출원은 2018년 2월 1일에 벡터 계산 유닛이라는 명칭으로 출원된 미국 특허 가출원 제62/625,251호에 대한 우선권을 주장하고, 2017년 7월 24일에 가속된 수학 엔진이라는 명칭으로 출원된 미국 특허 가출원 제62/536,399호에 대한 우선권을 주장하며, 2017년 9월 20일에 가속된 수학 엔진이라는 명칭으로 출원된 계속중인 미국 특허출원 제15/710,433호의 일부계속출원(continuation-in-part)이며, 2017년 7월 24일에 가속된 수학 엔진이라는 명칭으로 출원된 미국 특허 가출원 제62/536,399호에 대한 우선권으로 주장하며, 이 모두는 모든 목적을 위해 참조로 여기에 포함된다.

### 배 경 기 술

[0002] 머신 러닝(machine learning) 및 인공 지능(artificial intelligence)을 처리하는 경우, 일반적으로 많은 양의 데이터에 대해 수학 연산(mathematical operation)을 수행해야 하며 여러 컨볼루션 레이어(convolution layer) 및 풀링 레이어(pooling layer)를 해결해야하는 경우가 종종 있다. 머신 러닝 및 인공 지능 기술은 일반적으로 매트릭스 연산(matrix operations) 및 활성화 함수(activation function)와 같은 비선형 함수(non-linear function)을 사용한다. 머신 러닝의 어플리케이션에는 자율 주행(self-driving) 및 운전자 보조 자동차(driver-assisted automobile)가 포함된다. 일부 시나리오에서 컴퓨터 프로세서는 머신 러닝 훈련(training)과 추론(inference)을 수행하는데 사용된다. 기존의 컴퓨터 프로세서는 단일 수학 연산을 매우 빠르게 수행할 수 있지만 일반적으로 제한된 양의 데이터에서만 동시에 작동 할 수 있다. 대안으로서, 그래픽 프로세싱 유닛(GPU: graphical processing unit)이 이용될 수 있고 동일한 수학 연산을 수행할 수 있지만 더 큰 데이터 세트에 대해 병렬로 수행할 수 있다. GPU는 여러 프로세서 코어를 사용하여 여러 작업을 병렬로 수행할 수 있으며 일반적으로 기존 컴퓨터 프로세서보다 더 빠르게 병렬 처리를 사용하는 대형 그래픽 처리 작업을 완료할 수 있다. 그러나 GPU 나 기존 컴퓨터 프로세서는 원래 머신 러닝이나 인공 지능 작업용으로 설계되지 않았다. 머신 러닝 및 인공 지능 작업은 종종 매우 큰 데이터 세트에 대해 일련의 특정 머신 러닝 프로세서 작업을 반복적으로 적용하는데 의존한다. 따라서, 각각의 병렬 동작에 대해 다수의 프로세싱 코어의 오버 헤드 없이 큰 데이터 세트에 대해 머신 러닝 및 인공 지능 특정 처리 동작 수행을 지원하는 마이크로 프로세서 시스템이 필요하다.

## 발명의 내용

[0003] 벡터 계산 유닛(vector computational unit) 및 벡터 계산 유닛 명령 세트 아키텍처(vector computational unit instruction set architecture)를 이용하는 마이크로 프로세서 시스템(microprocessor system)이 개시된다. 예를 들어, 마이크로 프로세서 시스템은 벡터 계산 유닛과 통신하는 계산 어레이(computational array)를 포함한다. 다양한 실시 예에서, 계산 어레이는 2 개의 입력 벡터에 대해 산술 연산(arithmetic operation)을 수행할 수 있는 매트릭스 프로세서(matrix processor)이며, 입력 벡터로부터 M 피연산자(M operand)와 N 피연산자(N operand)를 수신하기 위한 복수의 계산 유닛(computation unit)을 포함한다. 일부 실시 예에서, 계산 유닛은 산술 논리 유닛(arithmetic logic unit), 어큐뮬레이터(accumulator) 및 내적(dot-product) 생성 및 컨벌루션(convolution)을 위한 다양한 처리 수행과 같은 작업을 수행하기 위한 섀도우 레지스터(shadow register)를 포함하는 서브-회로(sub-circuit)이다. 종래의 GPU(Graphic Processing Unit) 또는 CPU(Central Processing Unit) 처리 코어와 달리, 각각의 코어가 자신의 고유한 처리 명령(own unique processing instruction)을 수신하도록 구성되고, 계산 어레이의 계산 유닛은 각각 계산 어레이에 의해 수신된 개별 명령(individual instruction)에 응답하여 동일한 계산을 병렬로 수행한다. 다양한 실시 예에서, 벡터 계산 유닛은 병렬로 입력 데이터의 벡터에 대해 로드(load), 산술(arithmetic) 및 저장(store) 동작을 수행하기 위한 복수의 처리 엘리먼트(processing element)를 포함한다. 벡터 계산 유닛의 처리 엘리먼트는 계산 어레이로부터 출력을 수신하도록 구성된다. 다양한 실시 예에서, 계산 어레이의 출력 및 벡터 계산 유닛으로의 입력은 데이터의 어레이(array)이다. 벡터 계산 유닛으로의 수신된 입력은 단일 프로세서 명령(a single processor instruction)에 응답하여 병렬로 처리된다. 계산 어레이와 유사하게, 벡터 계산 유닛의 처리 엘리먼트는 각각 벡터 계산 유닛에 의해 수신된 개별 명령에 응답하여 동일한 계산을 병렬로 수행한다. 일부 실시 예에서, 마이크로 프로세서 시스템은 벡터 계산 유닛에 명령을 제공하도록 구성된 제어 유닛(control unit)을 더 포함한다. 각각의 단일 프로세서 명령(single processor instruction)은 벡터 계산 유닛에 의해 실행될 복수의 컴포넌트 명령(component instruction)을 지정할 수 있다. 단일 명령에 응답하여, 벡터 계산 유닛의 복수의 처리 엘리먼트의 각각은 다른 처리 엘리먼트와 병렬로 벡터 입력의 상이한 데이터 엘리먼트를 처리한다. 일부 실시 예에서, 벡터 계산 유닛의 출력은 풀링 연산(pooling operation)과 같은 후 처리(post-processing)를 수행하기 위해 후 처리 유닛(post-processing unit)으로 공급된다.

## 도면의 간단한 설명

[0004] 본 발명의 다양한 실시 예는 다음의 상세한 설명 및 첨부 도면에 개시되어 있다.

도 1은 머신 러닝 처리를 수행하기 위한 마이크로 프로세서 시스템의 실시 예를 도시한 블록도이다.

도 2는 머신 러닝 처리를 수행하기 위한 마이크로 프로세서 시스템의 실시 예를 도시한 블록도이다.

도 3은 머신 러닝 처리를 수행하기 위한 마이크로 프로세서 시스템의 실시 예를 도시한 블록도이다.

도 4a는 머신 러닝 처리를 수행하기 위한 벡터 계산 유닛의 실시 예를 도시한 블록도이다.

도 4b는 벡터 레지스터의 예시적인 앨리어스(alias)를 나타내는 테이블이다.

도 5는 마이크로 프로세서 시스템에 대한 프로세서 명령을 결정하기 위한 프로세스의 실시 예를 도시한 흐름도이다.

도 6a는 벡터 계산 유닛의 실행(running execution)을 위한 프로세스의 실시 예를 도시한 흐름도이다.

도 6b는 벡터 계산 유닛에 의해 벡터 데이터를 처리하기 위한 프로세스의 실시 예를 도시한 흐름도이다.

도 7은 벡터 계산 유닛 명령을 위한 인코딩 포맷의 실시 예를 도시한 블록도이다.

도 8은 벡터 계산 유닛에 의해 단일 벡터 계산 유닛 명령을 수행하기 위한 프로세스의 실시 예를 도시한 흐름도이다.

도 9는 벡터 계산 유닛의 예시적인 명령 사이클을 도시한 도면이다.

도 10은 계산 어레이의 계산 유닛의 실시 예를 도시한 블록도이다.

## 발명을 실시하기 위한 구체적인 내용

[0005] 본 발명은 프로세스, 장치, 시스템, 물질의 구성, 컴퓨터 판독 가능한 저장 매체에 구현된 컴퓨터 프로그램 제품 및/또는 프로세서와 결합된 메모리에 저장되거나 제공되는 명령을 실행하도록 구성된 프로세서와 같은 다양한 방법으로 구현될 수 있다. 본 명세서에서, 이러한 구현들 또는 본 발명이 취할 수 있는 임의의 다른 형태는 기술로 지칭될 수 있다. 일반적으로, 개시된 프로세스의 단계 순서는 본 발명의 범위 내에서 변경될 수 있다. 달리 명시되지 않는 한, 작업을 수행하도록 구성되는 것으로 기술되는 프로세서 또는 메모리와 같은 컴포넌트(component)는 주어진 시간에 작업을 일시적으로 수행하도록 구성된 일반 구성 요소 또는 작업을 수행하도록 제조된 특정 구성 요소로서 구현될 수 있다. 본 명세서에서 사용되는 용어 "프로세서"는 컴퓨터 프로그램 명령과 같은 데이터를 처리하도록 구성된 하나 이상의 장치, 회로 및/또는 처리 코어를 지칭한다.

[0006] 본 발명의 하나 이상의 실시 예에 대한 상세한 설명은 본 발명의 원리를 설명하는 첨부 도면과 함께 이하에 제공된다. 본 발명은 이러한 실시 예와 관련하여 설명되지만, 본 발명은 실시 예로 제한되지 않는다. 본 발명의 범위는 청구 범위에 의해서만 제한되며 본 발명은 수많은 대안(alternatives), 수정(modifications) 및 등가물(equivalents)을 포함한다. 본 발명의 완전한 이해를 제공하기 위해 다음의 설명에서 다수의 특정 세부 사항이 설명된다. 이들 세부 사항은 예시의 목적으로 제공되며, 본 발명은 이러한 특정 세부 사항의 일부 또는 전부를 제외하고 청구 범위에 따라 실시될 수 있다. 명확성을 위해, 본 발명과 관련된 기술 분야에서 알려진 기술 자료는 본 발명이 불필요하게 모호해지지 않도록 상세하게 설명되지 않았다.

[0007] 벡터 계산 유닛(vector computational unit) 및 벡터 계산 유닛 명령 세트 아키텍처(vector computational unit instruction set architecture)를 이용하는 마이크로 프로세서 시스템(microprocessor system)이 개시된다. 예를 들어, 마이크로 프로세서 시스템은 벡터 계산 유닛과 통신하는 계산 어레이(computational array)를 포함한다. 다양한 실시 예에서, 계산 어레이는 2 개의 입력 벡터에 대해 산술 연산(arithmetic operation)을 수행할 수 있는 매트릭스 프로세서(matrix processor)이며, 입력 벡터로부터 M 피연산자(M operand)와 N 피연산자(N operand)를 수신하기 위한 복수의 계산 유닛(computation unit)을 포함한다. 일부 실시 예에서, 계산 유닛은 산술 논리 유닛(arithmetic logic unit), 어큐뮬레이터(accumulator) 및 내적(dot-product) 생성 및 컨벌루션(convolution)을 위한 다양한 처리 수행과 같은 작업을 수행하기 위한 섀도우 레지스터(shadow register)를 포함하는 서브-회로(sub-circuit)이다. 종래의 GPU(Graphic Processing Unit) 또는 CPU(Central Processing Unit) 처리 코어와 달리, 각각의 코어가 자신의 고유한 처리 명령(own unique processing instruction)을 수신하도록 구성되고, 계산 어레이의 계산 유닛은 각각 계산 어레이에 의해 수신된 개별 명령(individual instruction)에 응답하여 동일한 계산을 병렬로 수행한다. 다양한 실시 예에서, 벡터 계산 유닛은 병렬로 입력 데이터의 벡터에 대해 로드(load), 산술(arithmetic) 및 저장(store) 동작을 수행하기 위한 복수의 처리 엘리먼트(processing element)를 포함한다. 벡터 계산 유닛의 처리 엘리먼트는 계산 어레이로부터 출력을 수신하도록 구성된다. 다양한 실시 예에서, 계산 어레이의 출력 및 벡터 계산 유닛으로의 입력은 데이터의 어레이(array)이다. 벡터 계산 유닛으로의 수신된 입력은 단일 프로세서 명령(a single processor instruction)에 응답하여 병렬로 처리된다. 계산 어레이와 유사하게, 벡터 계산 유닛의 처리 엘리먼트는 각각 벡터 계산 유닛에 의해 수신된 개별 명령에 응답하여 동일한 계산을 병렬로 수행한다. 일부 실시 예에서, 마이크로 프로세서 시스템은 벡터 계산 유닛에 명령을 제공하도록 구성된 제어 유닛(control unit)을 더 포함한다. 각각의 단일 프로세서 명령(single processor instruction)은 벡터 계산 유닛에 의해 실행될 복수의 컴포넌트 명령(component instruction)을 지정할 수 있다. 단일 명령에 응답하여, 벡터 계산 유닛의 복수의 처리 엘리먼트의 각각은 다른 처리 엘리먼트와 병렬로 벡터 입력의 상이한 데이터 엘리먼트를 처리한다. 일부 실시 예에서, 벡터 계산 유닛의 출력은 풀링 연산(pooling operation)과 같은 후 처리(post-processing)를 수행하기 위해 후 처리 유닛(post-processing unit)으로 공급된다.

[0008] 일부 실시 예에서, 마이크로 프로세서 시스템은 적어도 계산 어레이 및 벡터 계산 유닛을 포함한다. 예를 들어, 계산 어레이는 벡터 계산 유닛에 통신 가능하게 연결되어 계산 어레이의 출력이 벡터 계산 유닛에 입력으로서 공급된다. 다양한 실시 예에서, 계산 어레이는 복수의 계산 유닛을 포함한다. 예를 들어, 계산 유닛은 하나 이상의 곱셈(multiply), 가산(add) 및 시프트(shift) 연산을 수행하기 위한 기능을 포함하는 매트릭스 프로세서(matrix processor)의 서브 회로(sub-circuit) 일 수 있다. 다른 예로서, 계산 유닛은 내적 연산(dot-product operation)을 수행하기 위한 기능(functionality)을 포함하는 서브 회로 일 수 있다. 다양한 실시 예에서, 계산 어레이는 데이터 입력에 대해 다수의 연산을 병렬로 수행하기에 충분한 수의 계산 유닛을 포함한다. 예를 들어, M 피연산자와 N 피연산자를 수신하도록 구성된 계산 어레이는 적어도 M x N 계산 유닛을 포함할 수 있다. 다양한 실시 예에서, 마이크로 프로세서 시스템은 계산 어레이와 벡터 계산 유닛 사이의 처리를 조정(coordinating)하기 위한 제어 유닛을 더 포함한다. 예를 들어, 제어 유닛은 메모리에서 계산 어레이로 공급되는 데이터, 계

산 어레이로부터 벡터 계산 유닛으로 공급되는 데이터, 및/또는 벡터 계산 유닛으로부터 메모리에 저장되는 데이터를 조정하거나 또는 후 처리 장치로 공급할 수 있다. 일부 실시 예에서, 제어 유닛은 계산 어레이 명령을 계산 어레이에 제공하고, 벡터 계산 유닛 명령을 벡터 계산 유닛에, 및/또는 후 처리 명령을 후 처리 유닛에 제공하도록 구성된다.

[0009] 일부 실시 예에서, 계산 어레이와 통신하는 벡터 계산 유닛은 계산 어레이로부터 출력 데이터 엘리먼트를 입력으로서 수신하도록 구성된 복수의 처리 엘리먼트를 포함한다. 예를 들어, 벡터 엔진(vector engine)과 같은 벡터 계산 유닛은 처리를 위해 벡터를 입력으로서 수신한다. 벡터 계산 유닛은 입력 벡터의 각 엘리먼트에 대한 처리 엘리먼트를 포함할 수 있다. N 엘리먼트(또는 피연산자)의 벡터를 수신하도록 구성된 예시적인 벡터 계산 유닛은 N 개의 엘리먼트를 병렬로 처리하기 위한 N 처리 엘리먼트를 포함할 수 있다. 다양한 실시 예에서, 처리 엘리먼트는 계산 어레이로부터 출력 데이터 요소를 수신하도록 구성된다. 예를 들어, 계산 어레이로부터의 출력은 벡터 계산 유닛의 처리 엘리먼트에 의해 수신되도록 공급되는 데이터 엘리먼트의 벡터 일 수 있다. 다양한 실시 예에서, 각각의 벡터 계산 유닛은 단일 프로세서 명령에 응답하여 계산 어레이로부터 수신된 출력 데이터 엘리먼트를 병렬로 처리한다. 예를 들어, 단일 프로세서 명령은 대응하는 데이터 엘리먼트에 대해 수행되는 벡터 계산 유닛의 각각의 처리 엘리먼트에 적용된다.

[0010] 일부 실시 예에서, 제어 유닛은 벡터 계산 유닛에 적어도 하나의 프로세서 명령을 제공하도록 구성된다. 단일 프로세서 명령은(예를 들어, 단일 프로세서 명령에 응답하여) 벡터 계산 유닛에 의해 실행되는 복수의 컴포넌트 명령(component instruction)을 지정한다. 예를 들어, 제어 유닛은 다수의 컴포넌트 명령을 포함하는 명령 트라이어드(instruction triad)와 같은 단일 벡터 명령을 벡터 계산 유닛에 제공한다. 일부 실시 예에서, 명령 트라이어드는 개별 로드 명령(separate load instruction), 산술 논리 유닛(ALU: arithmetic logic unit) 명령 및 저장 명령과 같은 최대 3 개의 컴포넌트 명령을 포함하는 간단한 프로세서 명령이다. 3 개의 컴포넌트 명령은(예를 들어, 명령 트라이어드에 응답하여) 벡터 계산 유닛에 의해 수신되고 실행된다. 예를 들어, 로드 명령(load instruction), ALU 명령 및 저장 명령(store instruction)을 번들로 묶는 명령 트라이어드를 수신하는 벡터 계산 유닛은 로드 명령, 산술 명령(arithmetic instruction) 및 저장 명령을 실행한다. 다양한 실시 예에서, 단일 프로세서 명령에 응답하여, 벡터 계산 유닛의 복수의 처리 엘리먼트는 다른 처리 엘리먼트와 병렬로 상이한 데이터 엘리먼트를 처리하도록 구성된다. 예를 들어, 각각의 처리 엘리먼트는 입력 벡터로부터 벡터 계산 유닛으로 상이한 데이터 요소를 병렬로 처리할 수 있다. 다른 예로서, 단일 벡터 프로세서 명령 트라이어드의 각각의 컴포넌트 명령은, 벡터 계산 유닛을 사용하여 병렬로 N 엘리먼트의 전체 입력 벡터의 처리를 완료하기 위해, 벡터 입력의 엘리먼트의 각각에 적용될 수 있다.

[0011] 도 1은 머신 러닝 처리를 수행하기 위한 마이크로 프로세서 시스템의 실시 예를 도시한 블록도이다. 도시된 예에서, 마이크로 프로세서 시스템(100)은 제어 유닛(101), 데이터 입력(103), 가중치 입력(weight input)(105), 매트릭스 프로세서(matrix processor)(107), 벡터 엔진(vector engine)(111) 및 후 처리 유닛(post-processing unit)(115)을 포함한다. 데이터 입력(103) 및 가중치 입력(105)은 매트릭스 프로세서(107)를 위한 데이터를 준비하기 위한 입력 모듈이다. 일부 실시 예에서, 데이터 입력(103) 및 가중치 입력(105)은 각각 입력 데이터 포맷터(input data formatter), 캐시(cache) 또는 버퍼(buffer), 및/또는 매트릭스 프로세서(107)를 위한 데이터를 준비하기 위한 논리 회로(logic circuit)를 포함한다. 예를 들어, 데이터 입력(103)은 이미지 데이터(image data)에 대응하는 2 차원 어레이로부터 N 피연산자를 준비할 수 있고, 가중치 입력(105)은 매트릭스 프로세서(107)에 의해 처리되는 가중치 값(weight value)의 벡터에 대응하는 M 피연산자를 준비할 수 있다. 일부 실시 예에서, 도 5의 프로세스는 매트릭스 프로세서(107)를 위한 매트릭스 프로세서 명령(matrix processor instruction) 및 벡터 엔진(111)을 위한 벡터 엔진 명령(vector engine instruction)을 포함하는 마이크로 프로세서 시스템(100)상에서 동작하기 위한 명령을 준비하기 위해 수행된다. 일부 실시 예에서, 벡터 엔진(111)을 포함하는 마이크로 프로세서 시스템(100)은 도 6a, 도 6b 및 도 8과 관련하여 아래에서 설명되는 프로세스를 수행한다.

[0012] 일부 실시 예에서, 매트릭스 프로세서(107)는 복수의 계산 유닛(computation unit)을 포함하는 계산 어레이(computational array)이다. 예를 들어, 가중치 입력(105) 및 데이터 입력(103)으로부터 M 피연산자(M operand)와 N 피연산자(N operand)를 각각 수신하는 매트릭스 프로세서는 M x N 계산 유닛을 포함한다. 도시된 도면에서, 매트릭스 프로세서(107) 내부의 작은 정사각형은 매트릭스 프로세서(107)가 계산 유닛의 논리적 2 차원 어레이를 포함한다는 것을 나타낸다. 계산 유닛(109)은 매트릭스 프로세서(107)의 복수의 계산 유닛 중 하나이다. 일부 실시 예에서, 각각의 계산 유닛은 데이터 입력(103)으로부터 하나의 피연산자를 수신하고 가중치 입력(105)으로부터 하나의 피연산자를 수신하도록 구성된다. 일부 실시 예에서, 계산 유닛은 논리적 2 차원 어레

이에 따라 구성되지만, 매트릭스 프로세서는 물리적 2 차원 어레이로서 레이아웃 된 계산 유닛으로 반드시 제조될 필요는 없다. 예를 들어, 데이터 입력(103)의 i번째 피연산자와 가중치 입력(105)의 j번째 피연산자는 매트릭스 프로세서(107)의 i번째 x j번째 계산 유닛에 의해 처리되도록 구성된다.

[0013] 다양한 실시 예에서, 컴포넌트(component) 데이터 입력(103), 가중치 입력(105), 매트릭스 프로세서(107), 벡터 엔진(111) 및 후 처리 유닛(115)의 데이터 폭(data width)은 넓은 데이터 폭이고 하나 이상의 피연산자를 병렬로 전송하는 능력을 포함한다. 일부 실시 예에서, 데이터 입력(103) 및 가중치 입력(105)은 각각 96-바이트 폭(bytes wide)이다. 일부 실시 예에서, 데이터 입력(103)은 192 바이트 폭이고 가중치 입력(105)은 96 바이트 폭이다. 다양한 실시 예에서, 데이터 입력(103) 및 가중치 입력(105)의 폭은 동적으로 구성 가능하다. 예를 들어, 데이터 입력(103)은 96 또는 192 바이트로 동적으로 구성될 수 있고 가중치 입력(105)은 96 또는 48 바이트로 동적으로 구성될 수 있다. 일부 실시 예에서, 동적 구성은 제어 유닛(101)에 의해 제어된다. 다양한 실시 예에서, 96 바이트의 데이터 폭은 96 피연산자가 병렬로 처리될 수 있게 한다. 예를 들어, 96 바이트 폭으로 구성된 데이터 입력(103)을 갖는 실시 예에서, 데이터 입력(103)은 96 피연산자를 매트릭스 프로세서(107)에 병렬로 전송할 수 있다.

[0014] 다양한 실시 예에서, 매트릭스 프로세서(107)는 데이터 입력(103)으로부터 N 바이트를 수신하고 가중치 입력(105)으로부터 M 바이트를 수신하도록 구성되며 적어도 M x N 계산 유닛을 포함한다. 예를 들어, 매트릭스 프로세서(107)는 데이터 입력(103)으로부터 96 바이트 및 가중치 입력(105)으로부터 96 바이트를 수신하도록 구성될 수 있고 적어도 96 x 96 계산 유닛을 포함한다. 다른 예로서, 매트릭스 프로세서(107)는 데이터 입력(103)으로부터 192 바이트를 수신하고 가중치 입력(105)으로부터 48 바이트를 수신하도록 구성될 수 있고 적어도 192 x 48 계산 유닛을 포함한다. 다양한 실시 예에서, 매트릭스 프로세서(107)의 디멘션(dimension)는 동적으로 구성될 수 있다. 예를 들어, 매트릭스 프로세서(107)의 디폴트 디멘션(default dimension)는 데이터 입력(103)으로부터 96 바이트 및 가중치 입력(105)으로부터 96 바이트를 수신하도록 구성될 수 있지만, 입력 디멘션(input dimension)는 각각 192 바이트 및 48 바이트로 동적으로 구성될 수 있다. 다양한 실시 예에서, 각각의 계산 유닛의 출력 크기는 입력 크기와 같거나 크다. 예를 들어, 일부 실시 예에서, 각각의 계산 유닛으로의 입력은 2 개의 1 바이트 피연산자이고, 하나는 데이터 입력(103)으로부터의 피연산자에 대응하고 하나는 가중치 입력(105)으로부터의 피연산자에 대응하고, 2 개의 피연산자를 처리하는 출력은 4 바이트 결과이다. 다른 예로서, 매트릭스 프로세서(107)는 데이터 입력(103)으로부터 96 바이트 및 가중치 입력(105)으로부터 96 바이트를 수신하고 96개의 4 바이트 결과를 출력하도록 구성될 수 있다. 일부 실시 예에서, 매트릭스 프로세서(107)의 출력은 벡터이다. 예를 들어, 입력 벡터의 각 엘리먼트(또는 피연산자)가 1 바이트 크기인 두 개의 96-폭 입력 벡터(96-wide input vector)를 수신하도록 구성된 행렬 프로세서는 벡터 결과의 각 엘리먼트가 4 바이트 크기인 96-폭 벡터 결과(96-wide vector result)를 출력할 수 있다.

[0015] 다양한 실시 예에서, 매트릭스 프로세서(107)의 각각의 계산 유닛은 산술 논리 유닛(arithmetic logic unit), 어큐뮬레이터(accumulator) 및 섀도우 레지스터(shadow register)를 포함하는 서브 회로이다. 도시된 예에서, 매트릭스 프로세서(107)의 계산 유닛은 각각 가중치 입력(105) 및 데이터 입력(103)으로부터 M 피연산자 및 N 피연산자에 대한 산술 연산을 수행할 수 있다. 다양한 실시 예에서, 각각의 계산 유닛은 하나 이상의 곱셈(multiply), 가산(add), 어큐뮬레이트(accumulate) 및/또는 시프트(shift) 연산을 수행하도록 구성된다. 일부 실시 예에서, 각각의 계산 유닛은 내적 연산(dot-product operation)을 수행하도록 구성된다. 예를 들어, 일부 실시 예에서, 계산 유닛은 내적 결과(dot-product result)를 계산하기 위해 다수의 내적 컴퍼넌트 연산(dot-product component operation)을 수행할 수 있다. 예를 들어, 매트릭스 프로세서(107)의 계산 유닛의 어레이는 머신 러닝 모델(machine learning model)을 사용하여 추론(inference)을 수행하기 위해 요구되는 컨볼루션 단계(convolution step)를 수행하는데 이용될 수 있다. 이미지와 같은 2 차원 데이터 세트는 포맷되어 한번에 하나의 벡터인 데이터 입력(103)을 사용하여 매트릭스 프로세서(107)로 공급될 수 있다. 병렬로, 가중치를 포맷팅(formatting)하고 가중치 입력(105)을 사용하여 매트릭스 프로세서(107)에 벡터로서 공급함으로써 가중치의 벡터가 2 차원 데이터 세트에 적용될 수 있다. 매트릭스 프로세서(107)의 대응하는 계산 유닛은 가중치 및 데이터 입력의 대응하는 피연산자에 대해 병렬로 매트릭스 프로세서 명령(matrix processor instruction)을 수행한다.

[0016] 일부 실시 예에서, 벡터 엔진(vector engine)(111)은 매트릭스 프로세서(107)에 통신 가능하게 연결된 벡터 계산 유닛이다. 벡터 엔진(111)은 처리 엘리먼트(processing element)(113)를 포함하는 복수의 처리 엘리먼트를 포함한다. 도시된 도면에서, 벡터 엔진(111) 내부의 작은 정사각형은 벡터 엔진(111)이 벡터로서 배열된 복수의 처리 엘리먼트를 포함하는 것을 도시한다. 일부 실시 예에서, 처리 엘리먼트는 데이터 입력(103)과 동일한 방향으로 벡터로 배열된다. 일부 실시 예에서, 처리 엘리먼트는 가중치 입력(105)과 동일한 방향으로 벡터로 배열된

다. 다양한 실시 예에서, 벡터 엔진(111)의 처리 엘리먼트의 데이터 크기는 매트릭스 프로세서(107)의 계산 유닛의 데이터 크기와 동일하거나 더 크다. 예를 들어, 일부 실시 예에서, 계산 유닛(109)은 각각 1 바이트 크기의 2 개의 피연산자를 수신하고 결과를 4 바이트 크기로 출력한다. 처리 엘리먼트(113)는 4 바이트 크기의 입력으로서 계산 유닛(109)으로부터 4 바이트 결과를 수신한다. 다양한 실시 예에서, 벡터 엔진(111)의 출력은 벡터 엔진(111)에 대한 입력과 동일한 크기이다. 일부 실시 예에서, 벡터 엔진(111)의 출력은 벡터 엔진(111)으로의 입력에 비해 크기가 더 작다. 예를 들어, 벡터 엔진(111)은 각 4 바이트 크기인 최대 96 엘리먼트를 수신하고, 각 1 바이트 크기인 96 엘리먼트를 출력할 수 있다. 다양한 실시 예에서, 벡터 엔진(111)은 출력 결과에 대해 양자화(quantization)를 수행하여 벡터 엔진(111)의 출력은 벡터 엔지(111)에 대한 입력에 비해 크기가 더 작아지게 한다. 다양한 실시 예에서, 양자화는 단일 명령(single instruction)의 일부로서 수행된다. 예를 들어, 양자화 및 비선형 함수(non-linear function)는 단일 프로세서 명령으로서 수행된다. 상술한 바와 같이, 일부 실시 예에서, 데이터 입력(103) 및 가중치 입력(105)으로부터 매트릭스 프로세서(107) 로의 통신 채널은, 각 엘리먼트의 크기가 1바이트인 96-엘리먼트 폭(96-elements wide)이고, 벡터 엔진(111)의 출력 크기와 일치한다(각 엘리먼트의 크기가 1바이트인 96-엘리먼트 폭).

[0017] 일부 실시 예에서, 처리 엘리먼트(113)를 포함하는 벡터 엔진(111)의 처리 엘리먼트는 각각 산술 논리 유닛(ALU: arithmetic logic unit)(미 도시)을 포함한다. 예를 들어, 일부 실시 예에서, 각각의 처리 엘리먼트의 ALU는 산술 연산을 수행할 수 있다. 일부 실시 예에서, 처리 엘리먼트의 각각의 ALU는 정류된 선형 유닛(ReLU: rectified linear unit) 함수 및/또는 스케일링(scaling) 함수를 병렬로 수행할 수 있다. 일부 실시 예에서, 각각의 ALU는 비선형 활성화 함수(non-linear activation function)를 포함하는 비선형 함수(non-linear function)를 수행할 수 있다. 다양한 실시 예에서, 벡터 엔진(111)의 각각의 처리 엘리먼트는 입력 피연산자를 수신하기 위한 하나 이상의 플립 플롭(flip-flop)을 포함한다. 일부 실시 예에서, 각각의 처리 엘리먼트는 벡터 엔진 어큐뮬레이터(vector engine accumulator)의 슬라이스(slice) 및/또는 벡터 엔진(111)의 벡터 레지스터(vector register)에 접근(access) 할 수 있다. 예를 들어, 96-엘리먼트를 수신할 수 있는 벡터 엔진은 96-엘리먼트 폭 어큐뮬레이터(96-element wide accumulator)와 하나 이상의 96-엘리먼트 벡터 레지스터(96-element vector register)를 포함한다. 각각의 처리 엘리먼트는 어큐뮬레이터 및/또는 벡터 레지스터의 하나의 엘리먼트 슬라이스에 액세스 할 수 있다. 일부 실시 예에서, 각 엘리먼트는 크기가 4 바이트이다. 다양한 실시 예에서, 어큐뮬레이터 및/또는 벡터 레지스터들은 적어도 입력 데이터 벡터(input data vector)의 크기에 맞도록 크기가 정해진다. 일부 실시 예에서, 벡터 엔진(111)은 벡터 엔진(111)의 출력에 맞게 크기가 정해진 추가 벡터 레지스터를 포함한다.

[0018] 일부 실시 예에서, 벡터 엔진(111)의 처리 엘리먼트는 매트릭스 프로세서(107)로부터 데이터를 수신하도록 구성되며, 각각의 처리 엘리먼트는 수신된 데이터의 일부(portion)를 병렬로 처리할 수 있다. 처리 엘리먼트의 일례로서, 벡터 엔진(111)의 처리 엘리먼트(113)는 매트릭스 프로세서(107)의 계산 유닛(109)으로부터 데이터를 수신한다. 다양한 실시 예에서, 벡터 엔진(111)은 단일 벡터 프로세서 명령(single vector processor instruction)을 수신하고, 각각의 처리 엘리먼트는 다른 처리 엘리먼트와 병렬로 프로세서 명령(processor instruction)을 수행한다. 일부 실시 예에서, 프로세서 명령은 로드(load), 저장(store) 및/또는 산술 논리 유닛 연산(arithmetic logic unit operation)과 같은 하나 이상의 컴포넌트 명령(component instruction)을 포함한다. 다양한 실시 예에서, 노옵 연산(no-op operation)이 컴포넌트 명령을 대체하기 위해 사용될 수 있다.

[0019] 도시된 예에서, 데이터 입력(103)과 매트릭스 프로세서(107), 가중치 입력(105)과 매트릭스 프로세서(107), 매트릭스 프로세서(107)와 벡터 엔진(111) 및 벡터 엔진(111) 및 후 처리 유닛(post-processing unit)(115) 사이의 점선 화살표는 데이터 엘리먼트의 벡터와 같은 다수의 데이터 엘리먼트를 전송할 수 있는 각각의 컴포넌트 쌍 사이의 결합을 도시한다. 일 예로서, 매트릭스 프로세서(107)와 벡터 엔진(111) 사이의 통신 채널은 96 x 32 비트 폭(bits wide)일 수 있고, 각 엘리먼트의 크기가 32 비트인 경우 96 엘리먼트를 병렬로 전송하는 것을 지원할 수 있다. 다른 예로서, 벡터 엔진(111)과 후 처리 유닛(115) 사이의 통신 채널은 96 x 1 바이트 폭일 수 있고, 각 엘리먼트가 1 바이트 크기인 경우 96 엘리먼트를 병렬로 전송하는 것을 지원할 수 있다. 다양한 실시 예에서, 데이터 입력(103) 및 가중치 입력(105)은 메모리 모듈(도 1에 미도시)에 연결되고 각각 메모리 모듈로부터 입력 데이터를 수신할 수 있다. 일부 실시 예에서, 벡터 엔진(111)은 메모리 모듈(도 1에 미도시)에 추가로 결합되고 매트릭스 프로세서(107)로부터의 입력에 추가로 또는 대안적(alternatively)으로 메모리 모듈로부터 입력 데이터를 수신할 수 있다. 다양한 실시 예에서, 메모리 모듈은 일반적으로 정적 랜덤 액세스 메모리(SRAM: static random access memory)이다.

[0020] 일부 실시 예에서, 매트릭스 프로세서(107)의 하나 이상의 계산 유닛은 매트릭스 프로세서(107)가 다중 레인

(multiple lane)을 갖도록 레인(lane)으로 그룹화(group) 될 수 있다. 다양한 실시 예에서, 매트릭스 프로세서(107)의 레인은 데이터 입력(103) 또는 가중치 입력(105)과 정렬될 수 있다. 예를 들어, 가중치 입력(105)과 정렬된 레인은 가중치 입력(105)의 모든 피연산자마다 입력으로써 수신하도록 구성된 계산 유닛의 세트(set)를 포함한다. 유사하게, 데이터 입력(103)과 정렬된 레인은 데이터 입력(103)의 모든 피연산자마다 입력으로써 수신하도록 구성된 계산 유닛의 세트(set)를 포함한다. 도 1에 도시된 예에서, 레인은 수직 열(vertical column)로 가중치 입력(105)을 따라 정렬되고 각각의 레인은 벡터 엔진(111)의 대응하는 레인으로 공급된다. 일부 실시 예에서, 각각의 레인은 곱셈, 가산 및/또는 어큐뮬레이트, 및 시프트 함수를 포함하는 서브-회로들의 수직 열이다. 일부 실시 예에서, 매트릭스 프로세서(107)는 타일(tile)의 매트릭스를 포함하고 각각의 타일은 계산 유닛의 매트릭스이다. 예를 들어, 96 x 96 매트릭스 프로세서는 6 x 6 타일의 매트릭스를 포함할 수 있으며, 각 타일은 16 x 16 계산 유닛을 포함한다. 일부 실시 예에서, 수직 레인(vertical lane)은 타일들의 단일 열(single column of tiles)이다. 일부 실시 예에서, 수평 레인(horizontal lane)은 단일 행(single row)의 타일이다. 다양한 실시 예에서, 레인의 디멘션(dimension)은 동적으로 구성될 수 있고 매트릭스 프로세서(107), 벡터 엔진(111) 및/또는 후 처리 유닛(115)으로의 입력에 대한 정렬 연산(alignment operation)을 수행하기 위해 이용될 수 있다. 일부 실시 예에서, 제어 유닛(101)에 의해 제어되는 프로세서 명령의 사용하고/하거나 제어 유닛(101)을 사용하거나 제어 유닛(101)에 의하여 동적 구성이 수행된다.

[0021] 일부 실시 예에서, 제어 유닛(101)은 매트릭스 프로세서(107), 벡터 엔진(111) 및 후 처리 유닛(115)에 의해 수행된 처리를 동기화(synchronizes)한다. 예를 들어, 제어 유닛(101)은 프로세서 특정 명령(processor specific instruction)을 매트릭스 프로세서(107), 벡터 엔진(111) 및 후 처리 유닛(115) 각각에 전송할 수 있다. 제어 유닛(101)은 매트릭스 프로세서 명령(matrix processor instruction)을 매트릭스 프로세서(107)에 전송할 수 있다. 매트릭스 프로세서 명령은 데이터 입력(103) 및/또는 가중치 입력(105)으로부터의 특정 피연산자를 사용하여, 내적(dot-product) 또는 내적 컴포넌트(dot-product component)와 같은 산술 연산(arithmetic operation)을 수행하도록 계산 어레이에 지시하는 계산 어레이 명령(computational array instruction) 일 수 있다. 제어 유닛(101)은 벡터 프로세서 명령(vector processor instruction)을 벡터 엔진(111)으로 전송할 수 있다. 예를 들어, 벡터 프로세서 명령은 벡터 계산 유닛에 의해 함께 실행될 복수의 컴포넌트 명령을 갖는 단일 프로세서 명령(single processor instruction)을 포함할 수 있다. 제어 유닛(101)은 후 처리 명령(post-processing instruction)을 후 처리 유닛(post-processing unit)(115)으로 전송할 수 있다. 다양한 실시 예에서, 제어 유닛(101)은 데이터 입력(103) 및 가중치 입력(105)으로부터 매트릭스 프로세서(107)에 공급된 데이터를 매트릭스 프로세서(107)로부터 벡터 엔진(111) 및 벡터 엔진(111)으로부터 후 처리 유닛(115)에 동기화한다. 일부 실시 예에서, 제어 유닛(101)은 프로세서 특정 메모리(processor specific memory), 큐(queue) 및/또는 디큐(dequeue) 연산을 이용함으로써 데이터 입력(103), 가중치 입력(105), 매트릭스 프로세서(107), 벡터 엔진(111) 및/또는 후 처리 유닛(115) 사이를 포함하는 마이크로 프로세서 시스템(100)의 상이한 컴포넌트들 사이에서 데이터를 동기화한다. 일부 실시 예에서, 데이터 및 명령 동기화는 제어 유닛(101)에 의해 수행된다. 일부 실시 예에서, 데이터 및 명령 동기화는 매트릭스 프로세서(107), 벡터 엔진(111) 및/또는 후 처리 유닛(115) 사이의 처리를 동기화하기 위해 하나 이상의 시퀀서(sequencer)를 포함하는 제어 유닛(101)에 의해 수행된다.

[0022] 일부 실시 예에서, 매트릭스 프로세서(107) 및 벡터 엔진(111)은 컨볼루션 레이어(convolution layer)를 처리하기 위해 이용된다. 일부 실시 예에서, 벡터 엔진(111)은 매트릭스 프로세서(107)의 출력상에서 활성화 함수(activation function)와 같은 비선형 함수(non-linear function)를 수행하기 위해 이용된다. 예를 들어, 매트릭스 프로세서(107)는 내적을 계산하기 위해 사용될 수 있고 벡터 엔진(111)은 정류된 선형 유닛(ReLU) 또는 시그모이드 함수(sigmoid function)와 같은 활성화 함수를 수행하는데 사용될 수 있다. 일부 실시 예에서, 후 처리 유닛(115)은 풀링 연산(pooling operation)을 수행하기 위해 이용된다. 일부 실시 예에서, 후 처리 유닛(115)은 처리된 데이터를 포맷하고 메모리에 저장하기 위해 이용되고 메모리 기록 레이턴시(writing latency)를 동기화하기 위해 이용될 수 있다.

[0023] 도 2는 머신 러닝 처리를 수행하기 위한 마이크로 프로세서 시스템의 실시 예를 도시한 블록도이다. 도시된 예에서, 마이크로 프로세서 시스템(200)은 제어 유닛(201), 벡터 입력(203), 벡터 엔진 입력 큐(vector engine input queue)(207), 벡터 엔진(211) 및 후 처리 유닛(215)을 포함한다. 벡터 엔진 입력 큐(207)는 계산 유닛(209, 221-229)을 포함하는 복수의 계산 유닛을 포함하고 벡터 엔진(211)은 처리 엘리먼트(213, 231)를 포함하는 복수의 처리 엘리먼트를 포함한다. 벡터 입력(203)은 벡터 엔진 입력 큐(207)에 데이터를 공급하기 위한 입력 모듈이다. 일부 실시 예에서, 벡터 입력(203)은 입력 데이터 포맷터(input data formatter), 캐시 또는 버퍼, 및/또는 벡터 엔진 입력 큐(207)를 위한 데이터를 준비하기 위한 논리 회로(logic circuit)를 포함한다. 예를 들어, 벡터 입력(203)은 벡터 엔진 입력 큐(207)를 선입 선출(FIFO: first-in-first-out) 입력 큐로써 이

용하여 벡터 엔진(211)에 의해 처리되는 2 차원 어레이로부터 N 피연산자(N operand)를 준비할 수 있다. 일부 실시 예에서, 벡터 입력(203)은 데이터를 검색하기 위한 정적 랜덤 액세스 메모리(SRAM)와 같은 메모리(도 2에 미 도시)에 연결된다.

[0024] 다양한 실시 예에서, 제어 유닛(201), 벡터 입력(203), 벡터 엔진 입력 큐(207), 벡터 엔진(211) 및 후 처리 유닛(215)은 각각 도 1의 제어 유닛(101), 데이터 입력(103), 매트릭스 프로세서(107), 벡터 엔진(111) 및 후 처리 유닛(115)이다. 예를 들어, 도 1의 매트릭스 프로세서(107)는 도 1의 데이터 입력(103)으로부터 데이터를 수신하고 입력의 각 벡터를 도 1의 벡터 엔진(111)으로 반복적으로 시프트함으로써 벡터 엔진 입력 큐(207)와 같은 입력 큐(input queue)를 구현하는 데 사용될 수 있다.

[0025] 일부 실시 예에서, 벡터 엔진 입력 큐(207)는 계산 어레이 유닛이고 열(column)이 선입 선출(FIFO) 큐인 계산 유닛의 매트릭스를 포함한다. 도시된 예에서, 벡터 엔진 입력 큐(207)는 벡터 입력(203)을 위한 입력 큐이며, 벡터 입력(203)으로부터 벡터 엔진(211)으로 다수의 데이터 엘리먼트를 공급하기 위한 넓은 선입 선출(FIFO) 큐로서 기능한다. 예를 들어, 계산 유닛들(221-229)은 단일 FIFO 큐로서 함께 작동하는 계산 유닛들의 수직 열(vertical column)을 구성한다. 다양한 실시 예에서, 벡터 엔진 입력 큐(207)는 계산 유닛(221-229)과 유사한 계산 유닛의 수직 열로 구성된 다수의 FIFO 대기열을 포함한다. 예를 들어, 벡터 엔진 입력 큐(207)가 96 계산 유닛 폭(computation units wide)인 실시 예에서, 벡터 엔진 입력 큐(207)는 96 FIFO 큐에 대응하는 96 수직 열의 계산 유닛을 갖는다. 다른 예로서, 벡터 엔진 입력 큐(207)가 96 계산 유닛 길이(computation units long) 인 실시 예에서, 벡터 엔진 입력 큐(207)는 96 스테이지 길이(stages long)의 FIFO 큐를 갖는다.

[0026] 다양한 실시 예에서, 각각의 선입 선출(FIFO) 큐는 병렬로 작동하고 FIFO 큐를 따라 벡터 입력(203)으로부터 수신된 입력을 벡터 엔진(211)으로 시프트(shift)시킨다. 계산 유닛(221)을 포함하는 벡터 엔진 입력 큐(207)의 계산 유닛의 제1 행은 벡터 입력(203)에 연결된다. 계산 유닛의 제1 행은 벡터 입력(203)으로부터 전체 행의 데이터를 병렬로 수신하도록 구성된다. 벡터 엔진 입력 큐(207)의 계산 유닛의 마지막 행은 벡터 엔진(211)의 처리 엘리먼트의 행에 연결된다. 예를 들어, 벡터 엔진 입력 큐(207)의 계산 유닛의 마지막 행은 계산 유닛(229, 209)을 포함한다. 계산 유닛(209)은 처리 엘리먼트(213)에 연결되고 계산 유닛(229)은 처리 엘리먼트(231)에 연결된다. 처리 엘리먼트(213, 231)는 각각 계산 유닛(209, 229)의 데이터 출력 엘리먼트를 수신하도록 구성된다. 벡터 엔진(211)의 처리 엘리먼트는 벡터 엔진 입력 큐(207)의 마지막 행의 계산 유닛으로부터 데이터의 전체 행(entire row)을 병렬로 수신한다. 다양한 실시 예에서, 벡터 엔진 입력 큐(207)의 계산 유닛들의 마지막 행이 디큐(dequeue) 가능한 데이터를 가질 때, 디큐 준비 신호(dequeue ready signal)는 벡터 엔진(211)에 의해 수신되어 벡터 엔진 입력 큐(207)가 큐 연산(queue operation)을 수신할 준비가 되었음을 표시한다.

[0027] 설명된 예에서, 계산 유닛의 제1 행으로부터의 데이터는 벡터 엔진(211)을 향한 논리적 방향(logical direction)으로 계산 유닛의 다음 행으로 열(column) 아래로 시프트(shift)된다. 예를 들어, 벡터 입력(203)의 데이터 엘리먼트에 대응하는 입력은, 계산 유닛(221)에서 피연산자로 수신되고, 계산 유닛(221)으로부터 계산 유닛(222)으로, 계산 유닛(222)으로부터 계산 유닛(223)으로, 계산 유닛(223)으로부터 계산 유닛(224)으로 등과 같이 시프트되며, 계산 유닛(221)에서 수신된 피연산자가 중간 계산 유닛(222-228)을 통해 계산 유닛(221)으로부터 계산 유닛(229)으로 증가적으로(incrementally) 시프트 될 때까지 시프트된다. 다양한 실시 예에서, FIFO로 푸시된 데이터 엘리먼트는 FIFO가 계산 유닛에서 깊이(deep) 만큼 많은 시프트를 취한다. 예를 들어, 96 계산 유닛과 96 스테이지를 가진 FIFO 큐는 삽입된 엘리먼트를 디큐(dequeue)하기 위해 96 시프트가 필요하다. 다양한 실시 예에서, FIFO의 각 스테이지(stage)는 다른 스테이지들과 병렬로 피연산자를 시프트 할 수 있다. 예를 들어, FIFO 큐의 각각의 중간 계산 유닛이 그의 피연산자를 다음 계산 유닛으로 이동시키는 동안, 제1 계산 유닛은 벡터 입력(203)으로부터 다음 데이터 엘리먼트를 검색할 수 있고 마지막 계산 유닛은 벡터 엔진(211)의 대응하는 처리 엘리먼트에 의해 수신되는 데이터 엘리먼트를 디큐 할 수 있다. 설명된 예에서, 계산 유닛의 각 행을 따른 각각의 계산 유닛은 병렬로 작동하여, 벡터 입력(203)으로부터 벡터 엔진(211)으로 원래 수신된 대응하는 데이터 엘리먼트를 시프트시킨다.

[0028] 일부 실시 예에서, 벡터 엔진 입력 큐(207)는 벡터 입력(203)에 연결되고 계산 유닛의 매트릭스(matrix)의 하나의 디멘션(dimension)은 벡터 입력(203)의 디멘션과 일치한다. 예를 들어, 96 바이트의 폭을 갖는 벡터 입력(203)을 갖는 실시 예에서, 벡터 엔진 입력 큐(207)는 적어도 96 바이트의 폭을 갖는 계산 유닛의 매트릭스를 갖는다. 일부 실시 예에서, 벡터 입력(203)의 폭 및 벡터 엔진 입력 큐(207)에 대한 입력들의 대응하는 폭은 동적으로 구성 가능하다. 예를 들어, 벡터 입력(203)은 96 바이트 또는 96 x 2 바이트로 동적으로 구성될 수 있고 벡터 엔진 입력 큐(207)에 대응하는 입력의 폭은 각각 96 바이트 또는 96 x 2 바이트로 구성 가능하다. 일부 실

시 예에서, 구성은 제어 유닛(201) 및/또는 프로세서 명령을 사용하여 벡터 엔진 입력 큐(207)에 수행된다.

[0029] 일부 실시 예에서, 벡터 엔진(211)은 벡터 엔진 입력 큐(207)에 통신 가능하게 연결된 벡터 계산 유닛이다. 벡터 엔진(211)은 처리 엘리먼트(213, 231)를 포함하는 복수의 처리 엘리먼트를 포함한다. 도시된 도면에서, 벡터 엔진(211) 내부의 작은 정사각형은 벡터 엔진(211)이 벡터로서 배열된 복수의 처리 엘리먼트를 포함하는 것을 도시한다. 일부 실시 예에서, 처리 엘리먼트는 벡터 입력(203)과 동일한 방향으로 벡터로 배열된다. 다양한 실시 예에서, 벡터 엔진(211)의 처리 엘리먼트의 데이터 크기는 벡터 엔진 입력 큐(207)의 계산 유닛의 데이터 크기와 동일하거나 더 크다. 예를 들어, 일부 실시 예에서, 계산 유닛(209)은 1 바이트 크기의 피연산자를 수신하고 또한 1 바이트의 크기를 갖는 처리 엘리먼트(213)로의 출력을 디큐한다. 처리 엘리먼트(213)는 계산 셀(computation cell)(209)로부터 1 바이트 출력을 1 바이트 크기의 입력으로써 수신한다. 다양한 실시 예에서, 벡터 엔진(211)의 출력은 벡터 엔진(211)에 대한 입력과 동일한 크기이다. 다양한 실시 예에서, 벡터 엔진(211)의 출력은 벡터 엔진(211)에 대한 입력에 비해 크기가 더 작다. 예를 들어, 벡터 엔진(211)은 4 바이트 크기마다 최대 96 엘리먼트를 수신하고 1 바이트 크기마다 96 엘리먼트를 출력 할 수 있다. 일부 실시 예에서, 벡터 입력(203)으로부터 벡터 엔진 입력 큐(207) 로의 통신 채널은 크기가 1 바이트 인 각각의 엘리먼트로 96 엘리먼트 폭이고 벡터 엔진(211)의 출력 크기와 일치한다(크기가 1 바이트 인 각 엘리먼트로 96 엘리먼트 폭).

[0030] 일부 실시 예에서, 처리 엘리먼트(213, 231)를 포함하는 벡터 엔진(211)의 처리 엘리먼트는 각각 산술 논리 유닛(미도시)을 포함하고도 1의 벡터 엔진(111)에 대해 더 상세히 설명된다. 일부 실시 예에서, 벡터 엔진(211)의 처리 엘리먼트는 벡터 엔진 입력 큐(207)로부터 데이터를 수신하도록 구성되며, 각각의 처리 엘리먼트는 수신된 데이터 부분을 병렬로 처리 할 수 있다. 처리 엘리먼트의 일례로서, 벡터 엔진(211)의 처리 엘리먼트(213, 231)는 벡터 엔진 입력 큐(207)의 계산 유닛(209, 229)으로부터 각각 데이터를 수신한다. 다양한 실시 예에서, 벡터 엔진(211)은 단일 벡터 프로세서 명령을 수신하고, 각각의 처리 엘리먼트는 다른 처리 엘리먼트와 병렬로 프로세서 명령을 수행한다. 일부 실시 예에서, 프로세서 명령은 로드, 저장 및/또는 산술 논리 유닛 연산과 같은 하나 이상의 컴포넌트 명령을 포함한다. 다양한 실시 예에서, 노옵 연산(no-op operation)이 컴포넌트 명령을 대체하기 위해 사용될 수 있다.

[0031] 도시된 예에서, 벡터 입력(203)과 벡터 엔진 입력 큐(207), 벡터 엔진 입력 큐(207)와 벡터 엔진(211), 및 벡터 엔진(211)과 후 처리 유닛(215) 사이의 점선 화살표는 여러 데이터 엘리먼트를 전송할 수 있는 각각의 컴포넌트 쌍 사이의 결합을 도시한다. 일 예로서, 벡터 엔진 입력 큐(207)와 벡터 엔진(211) 사이의 통신 채널은 96 x 32 비트 폭일 수 있고, 각 엘리먼트가 32 비트 크기 인 경우 96 엘리먼트를 병렬로 전송하는 것을 지원할 수 있다. 다른 예로서, 벡터 엔진(211)과 후 처리 유닛(215) 사이의 통신 채널은 96 x 1 바이트 폭일 수 있고, 각 엘리먼트가 1 바이트 크기 인 경우에 96 엘리먼트를 병렬로 전송하는 것을 지원할 수 있다. 다양한 실시 예에서, 벡터 입력(203)은 메모리 모듈(도 2에 미도시)에 연결되고 메모리 모듈로부터 입력 데이터를 수신 할 수 있다. 일부 실시 예에서, 벡터 엔진(211)은 메모리 모듈(도 1에 미도시)에 추가로 결합되고 벡터 엔진 입력 큐(207)로부터의 입력에 추가로 또는 대안적으로 메모리 모듈로부터 입력 데이터를 수신할 수 있다. 다양한 실시 예에서, 메모리 모듈은 일반적으로 정적 랜덤 액세스 메모리(SRAM)이다.

[0032] 일부 실시 예에서, 벡터 엔진 입력 큐(207)의 하나 이상의 계산 유닛은 벡터 엔진 입력 큐(207)가 다수의 수직 열 레인(vertical column lane)을 갖도록 수직 열(vertical column)로 함께 그룹화 될 수 있다. 도 2에 도시된 예에서, 레인은 상술한 선입 선출(FIFO) 큐와 동일한 수직 열을 따라 정렬되고 각각의 레인은 벡터 엔진(211)의 대응하는 레인으로 공급된다. 일부 실시 예에서, 각각의 레인은 곱셈(multiply), 가산(add), 어큐뮬레이트(accumulate) 및/또는 시프트(shift) 기능을 포함하는 서브-회로들의 수직 열이다. 일부 실시 예에서, 수직 레인(vertical lane)은 계산 유닛들의 단일 열(single column of computation units)이다. 일부 실시 예에서, 수직 레인은 인접한 계산 유닛의 다수 열(multiple columns of adjacent computation units)의 그룹이다. 다양한 실시 예에서, 레인의 디멘션은 동적으로 구성 될 수 있고 벡터 엔진 입력 큐(207), 벡터 엔진(211) 및/또는 후 처리 유닛(215)에 대한 입력에 대한 정렬 연산(alignment operation)을 수행하기 위해 이용 될 수 있다. 일부 실시 예에서, 제어 유닛(201)에 의해 제어되는 프로세서 명령의 사용하고/하거나 제어 유닛(101)을 사용하거나 제어 유닛(101)에 의하여 동적 구성이 수행된다.

[0033] 일부 실시 예에서, 제어 유닛(201)은 벡터 엔진 입력 큐(207), 벡터 엔진(211) 및/또는 사후 처리 유닛(215)에 의해 수행된 처리를 동기화한다. 예를 들어, 제어 유닛(201)은 프로세서 특정 명령을 벡터 엔진 입력 큐(207), 벡터 엔진(211) 및 후 처리 유닛(215) 각각에 전송할 수 있다. 제어 유닛(201)은 벡터 엔진 입력 큐 명령(vector engine input queue instruction)을 벡터 엔진 입력 큐(207)에 전송할 수 있다. 일부 실시 예에서, 벡터 엔진 입력 큐 명령은 도 1의 매트릭스 프로세서(107)가 응답 할 수 있고 도 1과 관련하여 더 설명되는 매트

릭스 프로세서 명령의 서브 세트(subset)이다. 벡터 엔진 입력 큐 명령은 로드 연산(load operation), 시프트 연산(shift operation), 또는 입력 큐와의 인터페이스를 위한 다른 적절한 명령(appropriate instruction)을 계산 어레이(computational array)에 지시(instructs)하는 계산 어레이 명령(computational array instruction)일 수 있다. 제어 유닛(201)은 벡터 프로세서 명령을 벡터 엔진(211)에 전송할 수 있다. 예를 들어, 벡터 프로세서 명령은 벡터 계산 유닛에 의해 함께 실행될 복수의 컴포넌트 명령을 갖는 단일 프로세서 명령을 포함할 수 있다. 제어부(201)는 후 처리 명령어(post-processing instruction)를 후 처리 유닛(215)으로 전송할 수 있다. 다양한 실시 예에서, 제어 유닛(201)은 벡터 입력(203)으로부터 벡터 엔진 입력 큐(207)로, 벡터 엔진 입력 큐(207)로부터 벡터 엔진(211)으로 그리고 벡터 엔진(211)으로부터 후 처리 유닛(215)으로 공급되는 데이터를 동기화한다. 일부 실시 예에서, 제어 유닛(201)은 프로세서 특정 메모리(processor specific memory), 큐 및/또는 디큐 연산을 이용함으로써 상이한 컴포넌트 벡터 입력(203), 벡터 엔진 입력 큐(207), 벡터 엔진(211) 및/또는 후 처리 유닛(215) 사이에서 데이터를 동기화한다. 제어 유닛(201)의 기능은 도 1의 제어 유닛(101)과 관련하여 더 상세히 설명된다.

[0034] 일부 실시 예에서, 제어 유닛(201)은 벡터 엔진 입력 큐(207), 벡터 엔진(211) 및/또는 사후 처리 유닛(215)에 의해 수신될 데이터 엘리먼트의 크기 및 수를 구성하는데 이용된다. 예를 들어, 일부 실시 예에서, 제어 유닛(201)은 벡터 엔진 입력 큐(207)로의 입력을 각각 1 바이트 크기의 96 엘리먼트로 하거나 또는 각각 2 바이트 크기의 48 엘리먼트, 각각 2바이트 크기의 96 엘리먼트, 각각 4비트 크기의 192 엘리먼트 등과 같은 다른 적절한 변형으로 구성하도록 이용될 수 있다. 일부 실시 예에서, 벡터 엔진 입력 큐(207)는 일련의 로드 및 논리 시프트 연산을 수행함으로써 수신할 수 있는 것보다 더 큰 크기의 데이터 엘리먼트를 출력할 수 있다. 예를 들어, 4 바이트 입력 데이터 엘리먼트는 4 바이트 입력 데이터 엘리먼트의 4 개의 순차적인 1 바이트 부분을 판독하고 각 바이트를 적절한 비트 필드(bit field)로 논리적으로 시프트함으로써 벡터 엔진 입력 큐(207)에 로드(load)된다. 다른 예로서, 일부 실시 예에서, 제어 유닛(201)은 벡터 엔진(211)에 대한 입력을 각각 4 바이트 크기의 96 엘리먼트로 하거나 또는 각각 1 바이트 크기의 96 엘리먼트, 각각 2바이트 크기의 48 엘리먼트 등과 같은 다른 적절한 변형으로 구성하기 위해 이용될 수 있다.

[0035] 다양한 실시 예에서, 후 처리 유닛(215)은 벡터 엔진(211)으로부터의 출력의 후 처리를 수행하기 위해 이용된다. 후 처리 유닛(215)의 후 처리 기능은 도 1의 후 처리 유닛(115)과 관련하여 더 상세히 설명된다.

[0036] 도 3은 머신 러닝 처리를 수행하기 위한 마이크로 프로세서 시스템의 실시 예를 도시한 블록도이다. 도시된 예에서, 마이크로 프로세서 시스템(300)은 제어 유닛(301), 메모리(307), 벡터 엔진(311) 및 후 처리 유닛(315)을 포함한다. 다양한 실시 예에서, 메모리(307)는 전형적으로 정적 랜덤 액세스 메모리(SRAM)이다. 다양한 실시 예에서, 후 처리 유닛(315)은 벡터 엔진(311)으로부터 입력 데이터를 수신하고 벡터 엔진(311)으로부터의 출력의 후 처리를 수행하는데 이용된다. 처리 유닛(315)의 후 처리 기능은 도 1의 후 처리 유닛(115)과 관련하여 더 상세히 설명된다.

[0037] 도 3의 블록도는 벡터 엔진(311)이 메모리(307)에 연결되고 메모리(307)로부터 직접 데이터를 검색할 수 있는 시스템 아키텍처 실시 예를 도시한다. 다양한 실시 예에서, 메모리(307)와 벡터 엔진(311) 사이의 통신 채널의 크기는 다수의 데이터 엘리먼트를 메모리(307)에서 벡터 엔진(311)으로 병렬로 전송하도록 구성될 수 있다. 예를 들어, 벡터 엔진(311)이 각각 32 비트 크기의 96 엘리먼트를 병렬로 수신할 수 있는 실시 예에서, 메모리(307)와 벡터 엔진(311) 사이의 통신 채널의 크기는 메모리(307)로부터 벡터 엔진(311)으로 32 비트 크기의 96 엘리먼트를 병렬로 전송하도록 구성된다. 일부 실시 예에서, 메모리(307)는 벡터 엔진(311)으로 전송하기 전에 메모리로부터 데이터를 포맷하기 위한 데이터 캐시 또는 버퍼 및/또는 논리 회로를 포함 할 수 있는 데이터 포맷터(data formatter)(미도시)를 포함한다. 예를 들어, 1 바이트 크기의 데이터 엘리먼트는 메모리(307)의 워드 경계(word boundary)에 저장될 수 있고, 데이터 포맷터는 데이터를 바이트 경계(byte boundary)로 포맷 및/또는 마스크(mask)하는데 이용된다. 다양한 실시 예에서, 제어 유닛(301), 벡터 엔진(311) 및 후 처리 유닛(315)은 각각 도 1의 제어 유닛(101), 벡터 엔진(111) 및 후 처리 유닛(115)이다. 다양한 실시 예에서, 벡터 엔진(311)은 도 1의 매트릭스 프로세서(107)와 관련하여 설명된 바와 같이 매트릭스 프로세서(미도시)에 추가로 결합될 수 있다.

[0038] 일부 실시 예에서, 벡터 엔진(311)은 메모리(307)에 통신 가능하게 연결된 벡터 계산 유닛이다. 벡터 엔진(311)은 처리 엘리먼트(313)를 포함하는 복수의 처리 엘리먼트를 포함한다. 도시된 도면에서, 벡터 엔진(311) 내부의 작은 정사각형은 벡터 엔진(311)이 벡터로서 배열된 복수의 처리 엘리먼트를 포함하는 것을 도시한다. 일부 실시 예에서, 처리 엘리먼트(313)를 포함하는 벡터 엔진(311)의 처리 엘리먼트는 각각 산술 논리 유닛(미도시)을 포함한다. 벡터 엔진(311)의 처리 엘리먼트는 메모리(307)로부터 데이터를 수신하도록 구성되며, 각 처리 엘

리먼트는 수신된 데이터의 부분을 병렬로 처리할 수 있다. 다양한 실시 예에서, 벡터 엔진(311)은 단일 벡터 프로세서 명령을 수신하고, 각각의 처리 엘리먼트는 다른 처리 엘리먼트와 병렬로 프로세서 명령을 수행한다. 일부 실시 예에서, 프로세서 명령은 로드, 저장 및/또는 산술 논리 유닛 연산과 같은 하나 이상의 컴포넌트 명령을 포함한다. 벡터 엔진(311)의 기능은 각각 도 1 및 도 2의 벡터 엔진(111, 211)과 관련하여 더 상세히 설명된다.

[0039] 일부 실시 예에서, 제어 유닛(301)은 벡터 엔진(311) 및 후 처리 유닛(315)에 의해 수행된 처리를 동기화하고 메모리(307)에 대한 접근을 동기화한다. 예를 들어, 제어 유닛(301)은 프로세서 특정 명령을 벡터 엔진(311) 및 후 처리 유닛(315) 각각에 전송할 수 있다. 일부 실시 예에서, 제어 유닛(301)은 벡터 프로세서 명령을 벡터 엔진(311)에 전송할 수 있다. 예를 들어, 벡터 프로세서 명령은 벡터 계산 유닛에 의해 함께 실행될 복수의 컴포넌트 명령을 갖는 단일 프로세서 명령을 포함할 수 있다. 일부 실시 예에서, 제어 유닛(301)은 후 처리 명령을 후 처리 유닛(315)에 전송할 수 있다. 다양한 실시 예에서, 제어 유닛(301)은 메모리(307)로부터 벡터 엔진(311)에 의해 수신되고 벡터 엔진(311)으로부터 후 처리 유닛(315)에 의해 수신된 데이터를 동기화한다. 일부 실시 예에서, 제어 유닛(301)은 벡터 엔진 및/또는 후 처리 유닛 프로세서 특정 연산(post-processing unit processor specific operation)을 이용함으로써 상이한 컴포넌트 벡터 엔진(311) 및/또는 후 처리 유닛(315) 사이에서 데이터를 동기화한다. 제어 유닛(301)의 기능은 도 1의 제어 유닛(101)과 관련하여 더 상세히 설명된다.

[0040] 일부 실시 예에서, 제어 유닛(301)은 벡터 엔진(311) 및/또는 후 처리 유닛(315)에 의해 수신될 데이터 엘리먼트의 크기 및 수를 구성하기 위해 이용된다. 예를 들어, 일부 실시 예에서, 제어 유닛(301)은 벡터 엔진(311)이 각각 4 바이트 크기의 96 데이터 엘리먼트, 또는 각각 1 바이트 크기의 96엘리먼트, 각각 2바이트 크기의 48 엘리먼트 등과 같은 다른 적절한 변형을 수신하도록 구성될 수 있다. 도 1 및 도 2와 관련하여 더 설명 된 바와 같이, 벡터 엔진(311)과 후 처리 유닛(315) 사이의 점선 화살표는 다수의 데이터 엘리먼트를 전송할 수 있는 각각의 컴포넌트 쌍 사이의 결합을 도시한다. 일 예로서, 벡터 엔진(311)과 후 처리 유닛(315) 사이의 통신 채널은 96 x 1 바이트 폭일 수 있고 각 엘리먼트가 1 바이트 크기인 경우에 96 엘리먼트를 병렬로 전송하는 것을 지원할 수 있다.

[0041] 도 4a는 머신러닝 처리를 수행하기 위한 벡터 계산 유닛의 실시 예를 도시한 블록도이다. 도시된 예에서, 마이크로 프로세서 시스템(400)은 벡터 계산 유닛(401), 입력 버스(411) 및 출력 버스(431)를 포함한다. 벡터 계산 유닛(401)으로의 입력은 입력 버스(411)로부터 도착한다. 벡터 계산 유닛(401)으로부터의 출력은 출력 버스(output bus)(431)에 기록된다. 일부 실시 예에서, 입력 버스(411) 및 출력 버스(431)는 입력 버스(411) 및 출력 버스(431)의 기능을 모두 포함하는 단일 버스이다. 다양한 실시 예에서, 입력 버스(411) 및 출력 버스(431)는 다수의 데이터 엘리먼트를 병렬로 전송할 수 있는 폭(wide)의 데이터 버스이다. 예를 들어, 입력 버스(411)는 96 x 32 비트 폭 일 수 있고 출력 버스(431)는96 바이트 폭 일 수 있어 계산 유닛(401)의 병렬 처리 기능을 수용할 수 있다. 일부 실시 예에서, 벡터 계산 유닛(401)은 입력 버스(411)를 통해 벡터 계산 유닛 명령들을 수신한다. 일부 실시 예에서, 벡터 계산 유닛(401)은 명령 버스(미도시)와 같은 입력 버스(411) 이외의 통신 채널을 통해 벡터 계산 유닛 명령들을 수신한다.

[0042] 다양한 실시 예에서, 벡터 계산 유닛(401)은 각각 도 1, 도 2 및 도 3의 벡터 엔진(111, 211 및/또는 311)이다. 일부 실시 예에서, 입력 버스(411)는 도 1의 매트릭스 프로세서(107), 도 2의 벡터 엔진 입력 큐(207) 및/또는 도 3의 메모리(307)에 연결된다. 일부 실시 예에서, 출력 버스(431)는 각각 도 1, 도 2 및 도 3의 후 처리 유닛(115, 215 및/또는 315)에 연결된다. 다양한 실시 예에서, 벡터 계산 유닛(401)은 각각 도 1, 도 2 및 도 3의 제어 유닛(101, 201 및/또는 301)과 같은 벡터 계산 유닛(401) 외부의 마이크로 프로세서 시스템(400)의 제어 유닛(미도시)에 양방향으로 결합된다. 다양한 실시 예에서, 마이크로 프로세서 시스템(400)의 제어 유닛은 벡터 계산 유닛 명령들을 벡터 계산 유닛(401)에 전송한다. 일부 실시 예에서, 마이크로 프로세서 시스템(400)의 제어 유닛은 명령 및 데이터를 벡터 계산 유닛(401)에 동기화하기 위한 하나 이상의 시퀀서(sequencer)를 포함한다.

[0043] 도시된 예에서, 벡터 계산 유닛(401)은 레지스터(421), 벡터 엔진 제어 로직(vector engine control logic)(423), 입력 버퍼(input buffer)(425), 산술 논리 유닛(ALU: arithmetic logic unit)(427) 및 출력 버퍼(output buffer)(429)를 포함한다. 입력 버스(411)로부터의 입력 데이터는 입력 버퍼(425)에 의해 수신되고 출력 버스(431)에 기록된 출력은 출력 버퍼(429)로부터 기록된다. 일부 실시 예에서, 입력 버퍼(425) 및 출력 버퍼(429)는 데이터 버퍼 또는 캐시이며 메모리 동기화 기능을 제공한다. 예를 들어, 일부 실시 예에서, 입력 버스(411)로부터의 입력 판독 및/또는 출력 버스(431)로의 출력 기록은 입력 버퍼(425)를 이용하여 입력 데이터를

수신하고, 계산된 결과를 저장하기 위해 출력 버퍼(429)를 이용함으로써 평활화(smooth) 될 수 있는 예측할 수 없는 레이턴시(latency)를 갖는다. 다른 예로서, ALU(427)로부터의 출력이 기록 준비가 되었을 때 출력 버스(431)가 이용 가능하지 않을 수 있다. 일부 실시 예에서, 출력 버퍼(429)는 출력 버스(431)가 출력 버퍼(429)에 저장된 결과를 기록하기 위해 이용 가능할 때까지 ALU(427)가 계류중인 데이터를 계속 처리하도록 한다. 다양한 실시 예에서, 입력 버스(411) 및 출력 버스(431)는 마이크로 프로세서 시스템(400)의 제어 유닛(미도시)에 의해 제어되는 통신 채널이다.

[0044] 상술 한 바와 같이, 다양한 실시 예에서, 벡터 계산 유닛은 복수의 처리 엘리먼트를 포함한다. 일부 실시 예에서, 각각의 처리 엘리먼트는 데이터를 로딩하고, 데이터를 저장하고, 산술 논리 유닛 연산을 수행하기 위한 개별 기능을 포함한다. 개별 처리 엘리먼트는 도 4a의 블록도에 도시되지 않았다. 다양한 실시 예에서, 산술 논리 유닛(ALU)(427)은 각각의 처리 유닛의 대응하는 산술 논리 유닛(ALU)을 포함한다. 유사하게, 입력 버퍼(425) 및 출력 버퍼(429)는 각각의 처리 유닛에 대한 대응하는 입력 버퍼 및 출력 버퍼를 포함한다. 다양한 실시 예에서, ALU(427)는 입력 벡터의 모든 엘리먼트를 벡터 계산 유닛(401)에 병렬로 처리하기 위한 ALU 로직(ALU logic)을 포함한다. 일부 실시 예에서, ALU(427)는 ALU 결과를 양자화(quantizing)하기 위한 로직을 포함한다. 다양한 실시 예에서, ALU 로직, 예를 들어 비선형 함수 및 양자화(quantization)를 수행하기 위한 로직은 단일 프로세서 명령에 응답하여 수행될 수 있다.

[0045] 다양한 실시 예에서, 레지스터(421)는 벡터 계산 유닛(401)의 기능을 구현하기 위한 레지스터를 포함한다. 예를 들어, 레지스터들(421)은 다른 적절한 기능 중에서도 벡터 계산 유닛 명령을 수행하기 위한 피연산자를 저장하고, 비트 마스크를 구현하고, 상이한 메모리 크기의 레지스터 앨리어스(alias)를 사용하여 벡터 엘리먼트를 참조하기 위해 사용될 수 있다. 일부 실시 예에서, 레지스터(421)는 산술 명령어 벡터 레지스터; 마스크 레지스터; 가산(add), 감산(subtract) 및 부동 소수점(floating point) 연산과 같은 산술 연산을 수행하기 위한 레지스터; 및/또는 앨리어싱 벡터 엘리먼트(aliasing vector element)를 위한 레지스터를 포함한다. 일부 실시 예에서, 앨리어싱 벡터 엘리먼트에 사용되는 레지스터는 또한 산술 연산을 수행하는데 이용된다.

[0046] 일부 실시 예에서, 레지스터(421)는 산술 명령어 벡터 레지스터를 포함한다. 예를 들어 레지스터는 로드 연산, 저장 연산 및 ALU(산술 논리 장치) 연산을 위한 피연산자로 사용할 수 있다. 다른 예로서, 일부 실시 예에서, ALU 연산은 최대 4 개의 벡터 레지스터(vector register), 3 개는 소스 레지스터(source register) 및 1 개의 목적지 레지스터(destination register)를 아규먼트(argument)로 취할 수 있다. 다양한 실시 예에서, 프로세서 연산에 의해 사용되는 벡터 레지스터는 벡터 엘리먼트의 크기에 기초하여 상이한 벡터 엘리먼트에 대해 앨리어스(alias)된다. 예를 들어, 일부 실시 예에서, 8 비트, 16 비트, 32 비트 및/또는 부동 소수점 값에서 동작하기 위해 벡터 레지스터의 상이한 세트가 이용 가능하다. 일부 실시 예에서, 32 비트 값에 대한 벡터 레지스터의 세트는 또한 부동 소수점 값에도 사용된다. 다양한 실시 예에서, 32 비트 벡터 레지스터는 16 비트 벡터 레지스터 및 8 비트 벡터 레지스터로 앨리어스 된다. 예를 들어 하나의 32 비트 벡터 레지스터는 2 개의 16 비트 벡터 레지스터와 4 개의 8 비트 벡터 레지스터로 앨리어스 된다. 다른 예로서, 8 개의 96 x 32 비트 벡터 레지스터(RD0-RD7 레지스터)를 갖는 벡터 계산 유닛(401)은 16 개의 96 x 16 비트 벡터 레지스터(RW0-RW15 레지스터) 및 32 개의 96 x 8- 비트 벡터 레지스터(RB0 ~ RB31 레지스터)로 앨리어스 된다. RD0은 96 x 32 비트 벡터 레지스터, RW0은 96 x 16 비트 벡터 레지스터, RB0은 96 x 8 비트 벡터 레지스터이다. 벡터 레지스터 앨리어스의 다른 예가 도 4b에 도시되어 있다.

[0047] 일부 실시 예에서, 레지스터(421)는 벡터 계산 유닛(401)의 처리 엘리먼트의 수에 기초하여 하나 이상의 비트 마스크 레지스터(bit mask register)를 포함한다. 예를 들어, 96 처리 엘리먼트를 갖는 벡터 계산 유닛은 하나 이상의 96 비트 마스크 레지스터를 포함 할 수 있다. 다양한 실시 예에서, 마스크 레지스터는 메모리로부터 비트 마스크를 로딩함으로써 설정 될 수 있다. 마스크 레지스터는 입력 데이터에 대해 수행된 논리 연산의 결과를 벡터 계산 유닛(401)에 저장하는데 사용될 수 있다.

[0048] 일부 실시 예에서, 레지스터(421)는 가산, 감산, 부동 소수점 연산과 같은 산술 연산을 수행하기 위한 레지스터를 포함한다. 예를 들어, 일부 실시 예에서, 벡터 계산 유닛(401)은 벡터 가산 및 감산 명령들을 위한 캐리아웃 비트(carry-out bit) 및 부동 소수점 명령들에 대응하는 상태 비트(status bit)를 저장하기 위한 레지스터들을 포함한다.

[0049] 일부 실시 예에서, 벡터 계산 유닛(401)은 벡터 계산 유닛 명령들의 시퀀스를 저장하기 위한 명령 버퍼(instruction buffer)(미도시)를 포함한다. 일부 실시 예에서, 명령 버퍼는 커맨드 큐(command queue)이다. 다양한 실시 예에서, 명령 버퍼는 수행될 현재 및/또는 마지막 명령을 참조하기 위한 하나 이상의 포인터

(pointer)를 포함한다. 다양한 실시 예에서, 명령 버퍼는 벡터 계산 단위 명령들의 캐시로서 작용한다. 예를 들어, 하나 이상의 벡터 계산 유닛 명령이 벡터 계산 유닛(401)의 명령 버퍼에 로드(load)되고 명령이 실행될 수 있을 때까지 캐시(cache)된다. 명령이 실행되어 더 이상 필요하지 않으면 새로운 명령이 명령 버퍼에 로드 될 수 있다. 일부 실시 예에서, 벡터 계산 유닛 명령어들은 마이크로 프로세서 시스템(400)의 제어 로직(미도시)을 통해 외부 명령어 커맨드 큐(an external instruction command queue)로부터 수신된다.

[0050] 일부 실시 예에서, 벡터 계산 유닛(401)은 벡터 엔진 제어 로직(vector engine control logic)(423)을 포함한다. 벡터 엔진 제어 로직(423)은 벡터 계산 유닛 명령 페치(fetch), 명령 디코딩 및/또는 명령 실행을 포함하는 벡터 계산 유닛(401)의 기능을 구현하는데 이용된다. 다양한 실시 예에서, 벡터 엔진 제어 로직(423)은 입력 버퍼(425), 출력 버퍼(429) 및 레지스터(421)를 통해 데이터를 판독, 기록, 마스크 및/또는 앨리어스 하기 위한 로직을 포함한다. 일부 실시 예에서, 벡터 계산 유닛(401)은 디큐 준비 신호(dequeue ready signal)를 수신하고 벡터 엔진 제어 로직(423)을 사용하여 데이터가 입력 버스(411)를 통해 이용 가능하다는 것을 결정한다. 예를 들어, 벡터 엔진 제어 로직(423)은 디큐 준비 신호를 수신하면 입력 버스(411)에 부착된 입력 선입 선출 큐(미도시)로부터 데이터를 디큐 할 수 있다.

[0051] 도 4b는 벡터 레지스터의 예시적인 앨리어스를 나타내는 테이블이다. 테이블(450)은, 16 개의 96 x 16 비트 벡터 레지스터(레지스터 RW0-RW15) 및 32 개의 96 x 8 비트 벡터 레지스터(RB0-RB31 레지스터)로 앨리어스된 8 개의 96 x 32 비트 벡터 레지스터(레지스터 RD0-RD7)를 가진, 벡터 계산 유닛 구현을 위한, 벡터 레시스터의 앨리어스(aliasing)를 도시한다. 일부 실시 예에서, 테이블(450)의 벡터 레지스터는 도 4a의 벡터 계산 유닛(401)의 레지스터(421)의 벡터 레지스터이다. 도시된 예에서, 행(451)은 그 아래 행에 나열된 각각의 레지스터에 대해 앨리어스(alias)된 바이트 0, 1, 2 및 3에 대한 행을 포함한다. 행(453, 463, 473)은 96 x 32 비트 벡터 레지스터 RD0, RD1 및 RD7에 대응한다. 행(455, 465, 475)는 96 x 16 비트 벡터 레지스터 RW0-3 및 RW14-15에 대응한다. 행(457, 467, 477)은 96 x 8 비트 벡터 레지스터 RB0-7 및 RB28-31에 대응한다. 이 예에서 바이트 0-3은 각각 도 1, 도 2 및 도 3의 벡터 엔진(111, 211 및/또는 311)과 같은 벡터 계산 유닛의 96 레인 중 하나이다.

[0052] 도시된 예에서, 테이블(450)은 벡터 계산 유닛 실시 예의 96 레인의 단일 레인에 대한 벡터 레지스터 앨리어싱을 도시한다. 96 x 32 비트 벡터 레지스터 RD0은 바이트 0에서 바이트 3까지 순서가 지정된 4 바이트를 사용한다. 96 x 16 비트 벡터 레지스터 RW0 및 RW1은 각각 2 바이트로 앨리어스(alias) 된다. 벡터 레지스터 RW0은 바이트 0과 바이트 1에 앨리어스되고 벡터 레지스터 RW1은 바이트 2와 바이트 3에 앨리어스 된다. 96 x 8 비트 벡터 레지스터 RB0-RB3은 각각 바이트 0-3에 대응하는 1 바이트로 앨리어스된다. 유사하게, 96 x 32 비트 벡터 레지스터 RD1은 96 x 16 비트 벡터 레지스터 RW2(바이트 0 및 1) 및 RW3(바이트 2 및 3)에 앨리어스되고,96 x 8 비트 벡터 레지스터 RB4-RB7는 각각 바이트 0-3에 대응한다. 다른 예로, 96 x 32 비트 벡터 레지스터 RD7은 96 x 16 비트 벡터 레지스터 RW14(바이트 0 및 1) 및 RW15(바이트 2 및 3)에 앨리어스되고,96 x 8 비트 벡터 레지스터 RB28-RB31는 각각 바이트 0-3에 대응한다.

[0053] 다양한 실시 예에서, 벡터 계산 유닛 명령들은 벡터 레지스터의 모든 96 레인에서 병렬로 동작한다. 예를 들어, 96 레인 각각에 대해 벡터 레지스터 RB0은 바이트 0에서 작동하고, 벡터 레지스터 RB5는 바이트 1에서 작동하고, 벡터 레지스터 RW2는 바이트 0과 1에서 작동하고, 벡터 레지스터 RW15는 바이트 2와 3에서 작동하며, 벡터 레지스터 RD7 바이트 0-3에서 병렬로 작동한다.

[0054] 도 5는 마이크로 프로세서 시스템에 대한 처리 명령을 결정하기 위한 프로세스의 실시 예를 도시한 흐름도이다. 일부 실시 예에서, 도 5의 프로세스는 하이 레벨(high level) 프로그래밍 언어로 작성된 소프트웨어 프로그램을 계산 어레이 및 벡터 계산 유닛을 갖는 마이크로 프로세서 시스템을 위한 일련의 계산 어레이 및 벡터 계산 유닛 명령으로 변환한다. 다양한 실시 예에서, 마이크로 프로세서 시스템은 도 1의 마이크로 프로세서 시스템(100)이고, 계산 어레이는 도 1의 매트릭스 프로세서(107)이며, 벡터 계산 유닛은 도 1의 벡터 엔진(111)이다. 다양한 실시 예에서, 도 5의 프로세스는 자율 주행(self-driving) 및 운전자 보조 자동차(driver-assisted automobile)와 같은 머신 러닝 모델을 사용하여 추론(inference)을 수행하는 어플리케이션(application)을 포함하여 머신 러닝에 의존하는 어플리케이션을 구현하는데 이용된다.

[0055] 501단계에서는, 수행될 처리 및 처리의 서브 세트가 계산 어레이, 벡터 계산 유닛 및/또는 후 처리 유닛과 같은 상이한 코-처리 컴포넌트(co-processing component)에 할당될 것으로 결정된다. 다양한 실시 예에서, 처리는 상이한 코-처리 컴포넌트의 기능 및 효율(efficiency)에 기초하여 할당된다. 예를 들어, 특정 매트릭스 관련 연산이 계산 어레이에 할당되고 활성화 함수와 같은 비선형 함수를 포함하는 연산이 벡터 계산 유닛에 할당될 수 있다. 일부 실시 예에서, 풀링 연산(pooling operation)은 후 처리 유닛에 할당된다. 다른 예로서, 일부 실시 예

에서, 501단계에서, 컨볼루션 연산(convolution operation)이 내적 연산(dot-product operation)을 필요하고 내적 연산이 계산 어레이에 의해 수행되는 매트릭스 처리를 가장 잘 이용하는 것으로 결정된다. 일부 실시 예에서,이 결정은 여기에 설명 된 마이크로 프로세서 시스템을 목표로 하는 머신 러닝 어플리케이션을 컴파일(compile)함으로써 수행된다.

[0056] 503단계에서는, 501단계에서 결정되고 할당된 처리에 대응하는 하나 이상의 매트릭스 프로세서 명령이 결정된다. 예를 들어, 501단계에서 매트릭스 프로세서에 의해 수행되도록 결정된 내적 연산은 하나 이상의 매트릭스 프로세서 명령으로 변환된다. 다양한 실시 예에서, 매트릭스 프로세서 명령어는 계산 어레이 명령어이다. 일 예로서, 계산 어레이 명령어는 하나 이상의 데이터 벡터(data vector)가 도 1의 데이터 입력(103)과 같은 데이터 입력 컴포넌트(data input component)로부터 수신되고, 하나 이상의 가중치 벡터(weight vector)가 도 1의 가중치 입력(weight input)(105)과 같은 대응하는 가중치 입력 컴포넌트(weight input component)로부터 수신될 것을 요구할 수 있다. 추가적인 계산 어레이 명령어는 내적 연산을 처리하기 위한 곱셈, 어큐뮬레이트 및 시프트 연산을 포함할 수 있다. 예를 들어, 하나 이상의 내적 컴퍼넌트 연산을 사용하여 내적 결과를 계산할 수 있다. 다양한 실시 예에서, 계산 어레이 명령어는 수신된 입력 데이터에 대해 계산 어레이의 대응하는 계산 유닛에 의해 수행되는 처리에 관한 것이다. 일부 실시 예에서, 추가적인 계산 어레이 명령은 벡터 계산 유닛에 의한 처리를 위한 내적 결과를 준비하기 위한 명령들을 포함한다.

[0057] 505단계에서는, 벡터 계산 유닛에 의해 수행될 벡터 엔진 명령에 관한 결정이 이루어진다. 예를 들어, 벡터 엔진에 의해 수행되도록 결정된 501단계에서 결정된 활성화 함수와 관련된 연산은 하나 이상의 벡터 엔진 명령으로 변환된다. 다양한 실시 예에서, 벡터 엔진 명령은 벡터 계산 유닛 명령이다. 일 예로서, 벡터 계산 유닛 명령은 하나 이상의 데이터 벡터가 도 1의 매트릭스 프로세서(107)와 같은 계산 어레이로부터 수신될 것을 요구할 수 있다. 추가의 벡터 계산 유닛 명령은 정류된 선형 유닛(ReLu) 함수와 같은 비선형 활성화 함수(non-linear activation function)를 수행하기 위한 연산을 포함할 수 있다. 다양한 실시 예에서, 벡터 계산 유닛 명령은 벡터 계산 유닛의 대응하는 처리 엘리먼트에 의해 수신된 입력 데이터에 대해 수행되는 처리(processing)에 관한 것이다. 일부 실시 예에서, 추가적인 벡터 계산 유닛 명령은 후 처리 유닛에 의한 후 처리를 위한 처리 엘리먼트의 결과를 준비하기 위한 명령들을 포함한다.

[0058] 다양한 실시 예에서, 각각의 벡터 계산 유닛 명령은 벡터 계산 유닛에 의해 함께 실행될 복수의 컴포넌트 명령을 지정하는 단일 프로세서 명령이다. 복수의 컴포넌트 명령들의 실행은 단일 벡터 계산 유닛 명령에 응답하여 상이한 데이터 입력 엘리먼트에 대해 병렬로 벡터 계산 유닛의 처리 엘리먼트에 의해 수행된다. 예를 들어, 일부 실시 예에서, 단일 프로세서 명령은 3 개의 컴포넌트 명령(개별 로드(separate load), 산술 논리 유닛(arithmetic logic unit) 및 저장 명령(store instruction))을 포함한다. 3 개의 컴포넌트 명령은 벡터 계산 유닛에 의해 수신되고 실행된다. 일부 실시 예에서, 단일 처리 명령으로의 컴포넌트 명령의 번들링(bundling)은 505단계에서 수행된다. 다양한 실시 예에서, 벡터 계산 유닛 명령으로 번들링하기 위한 컴포넌트 명령들의 오더(order) 및 선택은 결정된 데이터 위험(hazard)에 기초한다.

[0059] 507단계에서는, 후 처리 유닛에 의해 수행될 후 처리 명령에 관한 결정이 이루어진다. 예를 들어, 후 처리 기능과 관련된 연산은 501단계에서 후 처리 유닛에 의해 수행되도록 결정되고 하나 이상의 후 처리 명령으로 변환된다. 일 예로서, 후 처리 명령은 하나 이상의 데이터 벡터가 도 1의 벡터 엔진(111)과 같은 벡터 계산 유닛으로부터 수신될 것을 요구할 수 있다. 추가적인 후 처리 명령은 맥스플링(maxpooling)과 같은 풀링 레이어 기능(performing pooling layer functionality)을 수행하기 위한 연산을 포함할 수 있다. 다양한 실시 예에서, 후 처리 명령은 특히 커널 크기(kernel size), 스트라이드(stride) 및/또는 공간 범위(spatial extent)와 같은 풀링 기능을 구성하기 위한 명령을 포함할 수 있다. 일부 실시 예에서, 추가적인 후 처리 명령은 후 처리 결과를 준비하고 기록하기 위한 명령을 포함한다.

[0060] 509단계에서는, 503단계, 505단계 및 507단계에서 결정된 코-프로세서 명령(co-processor instruction)의 수집(collection)의 실행에 대응하는 시퀀스(sequence)가 스케줄링 된다. 예를 들어, 계산 어레이, 벡터 계산 유닛 및/또는 후 처리 유닛과 같은 다양한 코-프로세서(co-processor)에 대한 각각의 프로세서 명령의 관련 오더(relative order) 및/또는 시퀀스(sequence)가 결정된다. 일부 실시 양태에서, 시퀀스는 코-프로세서 간의 인터랙션(interaction) 및 의존성(dependency)에 의존한다. 예를 들어, 벡터 계산 유닛으로의 입력은 계산 어레이로부터의 출력 결과의 이용 가능성에 의존할 수 있다. 다양한 실시 예에서, 데이터 위험을 포함하는 의존성이 결정되고 설명된다. 예를 들어, 다양한 실시 예에서, 벡터 계산 유닛 명령은 복수의 컴포넌트 명령을 포함하고 다수의 벡터 계산 유닛 명령이 병렬로 실행되도록 실행될 수 있다. 이용할 수 없는 데이터 자원에 기초한 데이터 위험이 결정되고 설명된다. 예를 들어, 로드 연산의 완료에 의존하는 산술 논리 유닛 연산이 수행되기 전에 로

드 연산이 완료되도록 하기 위해 벡터 계산 유닛 명령의 컴포넌트 명령에 노옵(no-op)이 삽입될 수 있다. 일부 실시 예에서, 단일 벡터 계산 유닛 명령으로의 컴포넌트 명령의 번들링은 509단계에서 결정된다. 일부 실시 예에서, 코-프로세서 명령의 오더(order)와 같은 명령 스케줄링의 일부 또는 전부는 각각 매트릭스 프로세서 및 벡터 엔진에 대해 503단계 및 505단계에서 수행된다. 예를 들어, 일부 실시 예에서, 각각의 단일 벡터 계산 유닛 명령에 대한 컴포넌트 명령들의 번들링은 505단계에서 결정된다.

[0061] 일부 실시 예에서, 마이크로 프로세서 시스템의 제어 유닛 및/또는 하나 이상의 시퀀서(sequencer)는 코-프로세서 명령의 수집의 처리를 개시(initiate)하고 조정(coordinate)하는데 이용된다. 예를 들어, 509단계에서 결정된 명령 시퀀스는 도 1의 제어 유닛(101)과 같은 제어 유닛에 의해 및/또는 대응하는 코-프로세서 명령을 매트릭스 프로세서(107)와 같은 계산 어레이에 발행하기 위해 하나 이상의 시퀀서에 의해 이용되고, 도 1의 벡터 엔진(111)과 같은 벡터 계산 유닛 및/또는 도 1의 후 처리 유닛(113)과 같은 후 처리 유닛을 포함한다. 일부 실시 예에서, 하나 이상의 시퀀서의 기능은 제어 유닛에 의해 수행된다. 예를 들어, 일부 실시 예에서, 제어 유닛은 특히 실행 시퀀서(execute sequencer), 메모리 접근 시퀀서(memory access sequencer), 네트워크 시퀀서 및/또는 벡터 엔진 시퀀서(vector engine sequencer)를 포함한다.

[0062] 도 6a는 벡터 계산 유닛의 실행(running execution)을 위한 프로세스의 실시 예를 도시한 흐름도이다. 도 6a의 프로세스는 벡터의 엘리먼트를 병렬로 처리하기 위해 벡터 계산 유닛에 의해 수행될 수 있다. 다양한 실시 예에서, 벡터 계산 유닛은 각각 도 1, 도 2, 도 3 및 도 4a의 벡터 엔진(111, 211, 311) 및/또는 벡터 계산 유닛(401)이다. 일부 실시 예에서,도 6a의 프로세스는 도 1의 제어 유닛(101)과 같은 제어 유닛에 의해 개시된다. 다양한 실시 예에서, 도 6a의 프로세스의 단계들 사이의 천이(transition)는 도 4a의 벡터 엔진 제어 로직(423)과 같은 벡터 계산 유닛의 제어 로직에 의해 수행된다.

[0063] 601단계에서는, 벡터 엔진 명령이 검색된다. 다양한 실시 예에서, 벡터 엔진 명령은 벡터 계산 유닛 명령이고 복수의 컴포넌트 명령들을 지정(specify)한다. 예를 들어, 명령 트라이어드(instruction triad)는 최대 3 개 컴포넌트 명령을 지정하는 단일 벡터 계산 유닛 명령이다. 예시적인 명령 트라이어드는 로드 연산(load operation), 산술 논리 유닛 연산(arithmetic logic unit operation) 및 저장 연산(store operation)이 단일 명령(single instruction)으로 포함한다. 601단계에서는, 명령이 검색되면, 프로세스는 603단계 및 605단계로 계속된다.

[0064] 603단계에서는, 추가 명령이 계류(pending)중인지 여부가 결정된다. 예를 들어, 다음 벡터 엔진 명령이 사용 가능하고 검색 준비가 될 수 있다. 다른 예로서, 계류중인 명령(pending instruction)을 캐싱(caching)하기 위한 명령 버퍼(instruction buffer)는 비어 있을 수 있고 다음 이용 가능한 명령을 검색(retrieving) 및/또는 대기(waiting)해야 한다. 일부 실시 예에서, 추가 명령들의 이용 가능성은 명령 버퍼에서 마지막 유효 명령(last valid instruction)을 참조하는 포인터(pointer)를 검사(inspecting)하는 것에 기초한다. 이용 가능한 추가 명령(additional instruction)이 없는 것에 응답하여 처리는 609단계로 진행한다. 하나 이상의 추가 명령의 이용 가능성에 응답하여 처리는 601단계로 되돌아 간다.

[0065] 605단계에서는, 601단계에서 검색된 벡터 엔진 명령이 디코딩된다. 다양한 실시 예에서, 단일 벡터 엔진 명령은 하나 이상의 컴포넌트 명령을 지정한다. 다양한 실시 예에서, 명령 및 컴포넌트 명령은 디코딩된다. 예를 들어, 로드(load), 산술 논리 유닛(arithmetic logic unit) 및 저장 컴포넌트 명령(store component instruction)을 포함하는 명령 트라이어드는 개별 컴퍼넌트 연산(separate component operation)으로 디코딩된다. 일부 실시 예에서, 디코딩은 각각의 컴포넌트 연산에 대한 오피코드(opcode) 및 오피코드에 대응하는 아규먼트(argument) 모두를 결정한다. 일례로서, 로드 컴포넌트 명령(load component instruction)은 바이트 벡터 디큐 연산(byte vector dequeue operation)에 대응하는 오피코드 및 디큐의 결과로서 바이트 벡터를 저장하기 위한 대응하는 목적지 벡터 레지스터(destination vector register)를 모두 포함한다. 다른 예로, 가산 컴포넌트 명령(add component instruction)은 사인(sign)된 16 비트 가산 연산에 해당하는 오피코드(opcode) 및 소스 및 대상 아규먼트에 대한 대응하는 벡터 레지스터를 모두 포함한다.

[0066] 607단계에서는, 605단계에서 디코딩된 명령이 실행된다. 일부 실시 예에서, 다수의 컴포넌트 명령을 지정하는 단일 벡터 엔진 명령은 벡터 계산 유닛의 처리 엘리먼트에 의해 실행된다. 예를 들어, 처리 엘리먼트의 벡터는 605단계에서 디코딩된 단일 벡터 엔진 명령을 실행한다. 일부 실시 예에서, 단일 벡터 엔진 명령의 각각의 컴포넌트 명령은 각각의 처리 엘리먼트에 의해 병렬로 더 실행된다. 예를 들어, 각각의 처리 엘리먼트에 대해, 로드 명령 및 산술 논리 유닛 명령(arithmetic logic unit instruction)이 병렬로 실행될 수 있다. 일부 실시 예에서, 로드 명령, 산술 논리 유닛 명령 및 저장 명령 병렬로 실행될 수 있다. 예를 들어, 다음 컴포넌트 연산은

벡터 엔진의 각 처리 셀(processing cell)에 의해 병렬로 수행된다: 입력 데이터의 벡터가 입력 어큐뮬레이터(input accumulator)로부터 벡터 레지스터로 로드 되고, 부동 소수점 곱셈 연산은 산술 논리 장치 (ALU)에 의해 두 개의 서로 다른 벡터 레지스터에서 수행되고, 16-비트 엘리먼트의 벡터는 벡터 레지스터로부터 메모리로 저장된다. 다양한 실시 예에서, 처리 엘리먼트가 컴포넌트 명령의 실행을 완료하면, 벡터 엔진 명령에 대한 처리가 완료된다.

[0067] 609단계에서는, 벡터 계산 유닛은 다음 명령을 기다린다. 예를 들어, 벡터 계산 유닛은 계류중인 명령을 캐쉬(cache)하기 위한 명령 버퍼가 실행될 유효한 명령을 포함할 때까지 대기한다. 다른 예로서, 벡터 계산 유닛은 다음 명령이 메모리로부터 수신되어 벡터 계산 유닛에 이용 가능해질 때까지 대기한다. 일부 실시 예에서, 벡터 계산 유닛은 추가 명령(additional instruction)이 있을 때까지 계류하면서 609단계에서 정지(halt)한다. 다양한 실시 예에서, 벡터 계산 유닛은 추가 명령을 기다리는 동안 609단계에서 인터럽트(interrupt)에 응답할 수 있다. 추가 명령의 도착에 응답하여, 처리는 다시 601단계로 계속된다.

[0068] 도 6b는 벡터 계산 유닛에 의해 벡터 데이터를 처리하기 위한 프로세스의 실시 예를 도시한 흐름도이다. 예를 들어, 도 6b는 계산 어레이 및/또는 선입 선출(FIFO) 큐와 같은 입력 소스로부터 벡터 계산 유닛에 의해 수신된 벡터 데이터에 적용되는 프로세스를 도시한다. 일부 실시 예에서, 도 6b의 프로세스는 벡터 결과를 계산하기 위해 벡터 입력에 대해 벡터 연산을 수행하기 위한 벡터 계산 유닛에 의해 수행되는 단계들을 도시한다. 다양한 실시 예에서, 도 6b의 프로세스는 벡터의 엘리먼트에 대한 처리를 병렬로 수행하기 위해 벡터 계산 유닛의 복수의 처리 엘리먼트를 이용한다. 다양한 실시 예에서, 벡터 계산 유닛은 각각 도 1, 도 2, 도3 및 도 4a의 벡터 엔진(111, 211, 311) 및/또는 벡터 계산 유닛(401)이다.

[0069] 651단계에서는, 로드 연산이 디코드(decode) 및 발행(issue)된다. 일부 실시 예에서, 벡터 계산 유닛으로 데이터를 수신하기 위해 로드 연산이 요구된다. 예를 들어, 일부 실시 예에서, 디큐 연산은 벡터 계산 유닛의 처리 엘리먼트에 의해 수신될 계산 어레이로부터 데이터 엘리먼트의 벡터를 디큐하는 로드 연산이다. 다양한 실시 예에서, 로드 연산은 단일 벡터 계산 유닛 명령을 구성하는 다수의 컴포넌트 명령 중 하나 일 수 있다. 로드 연산의 디코딩은 로드 연산의 특정 유형 및 적절한 연산을 결정한다. 예를 들어, 서로 다른 크기의 벡터 엘리먼트를 서로 다른 지정된 벡터 레지스터에 로드하기 위해 다양한 로드 연산이 존재한다. 651단계에서는, 선입 선출(FIFO) 큐로부터의 데이터 결과의 벡터의 디큐와 같은 입력 데이터의 수신을 개시하기 위해 로드 연산은 디코드 및 발행된다.

[0070] 653단계에서는, 벡터 계산 유닛은 651단계에서 발행된 로드 연산의 결과로서 벡터 형태의 입력 데이터를 수신한다. 예를 들어, 벡터 계산 유닛은 도 1의 매트릭스 프로세서(107)와 같은 계산 어레이, 도 2의 벡터 엔진 입력 큐(207)와 같은 선입 선출(FIFO) 큐, 또는 다른 적절한 데이터 소스(other appropriate data source)로부터 입력 데이터 엘리먼트의 벡터를 수신한다. 일부 실시 예에서, 입력 데이터는 입력 버퍼에 저장된다. 일부 실시 예에서, 입력 버퍼는 입력 데이터를 저장하기 위해 플립 플롭의 세트 및/또는 하나 이상의 어큐뮬레이터를 이용한다. 655단계에서 하나 이상의 벡터 레지스터에 로드될 수 있도록 입력 벡터의 크기의 입력 버퍼가 입력 데이터를 저장하는데 이용될 수 있다.

[0071] 655단계에서는, 653단계에서 수신된 벡터 데이터가 적절한 레지스터(appropriate register)에 로드된다. 예를 들어, 653단계에서 판독된 벡터 데이터는 로드 명령에 의해 지정된 벡터 레지스터에 로드된다. 일부 실시 예에서, 레지스터 앨리어싱(register aliasing)은 데이터가 벡터 레지스터에 어떻게 로드되는지를 결정하기 위해 사용된다. 예를 들어, 데이터는 동일한 레지스터의 메모리 위치에 로드되지만 사용된 명령 및 앨리어스된 레지스터(aliased register)를 기반으로 바이트, 하프 워드(half-word) 또는 워드 경계(word boundary)에 정렬될 수 있다. 일부 실시 예에서, 벡터 데이터를 벡터 레지스터로 로딩하는 것은 벡터의 어떤 바이트가 어느 레지스터 메모리 위치(register memory location)에 로드(load)될지를 결정하기 위해 벡터 비트 마스크(vector bit mask)와 같은 비트 마스크(bit mask)를 이용한다. 예를 들어, 96-비트 마스크는 벡터 레지스터의 어느 엘리먼트가 데이터를 수신해야 하는지 결정하는데 이용될 수 있다.

[0072] 657단계에서는, 추가 데이터가 필요한지에 대한 결정이 이루어진다. 예를 들어, 현재 벡터 계산 유닛 명령에 기초하여, 산술 논리 유닛(ALU) 연산을 수행하기 전에 추가 데이터가 필요할 수 있다. 추가 데이터가 필요하지 않은 경우, 처리는 661단계로 계속된다. 일 예로서, 현재 벡터 계산 유닛 명령이 노옵 연산(no-op operation)이 아닌 ALU 컴포넌트 연산(가산 연산과 같은)을 포함하는 경우 처리는 계속 661단계로 진행된다. 추가 데이터가 필요한 경우(예를 들어, 로드 연산이 계류 중이고 ALU 연산이 계류 중이 아닌 경우), 처리는 계속 659단계로 진행된다. 일부 실시 예에서, 명령 트라이어드는 ALU 연산을 현재 명령에 대해 ALU 연산이 수행되어서는 안된다는

것을 지시하는 노옵(no-op)으로 대체 할 수 있다.

[0073] 659단계에서는, 추가 데이터가 처리를 위해 벡터 계산 유닛에 로드된다. 예를 들어, 입력 가중치의 벡터와 같은 추가 입력 데이터는 메모리를 판독하고, 매트릭스 프로세서의 결과를 수신하거나, 선입 선출(FIFO) 큐를 디큐하거나, 다른 적절한 기술에 의해 로드될 수 있다. 일부 실시 예에서, 추가 데이터는 정적 랜덤 액세스 메모리(SRAM)와 같은 메모리를 판독함으로써 로드될 수 있다. 다양한 실시 예에서, 판독 버퍼(read buffer)와 같은 추가 컴퍼넌트는 데이터의 로딩을 동기화하고 및/또는 판독 지연 및 레이턴시를 설명하기 위해 이용될 수 있다. 다양한 실시 예에서, 659단계에서 로딩된 데이터는 가중치 입력의 벡터와 같은 입력 데이터의 벡터 일 수 있다.

[0074] 661단계에서는, 벡터 산술 논리 유닛(ALU) 연산이 수행된다. 다양한 실시 예에서, 벡터 ALU 연산은 특히 가산(사인(부호 있는)(signed) 및 언사인(부호 없는)(unsigned)), 감산(사인(부호 있는)(signed) 및 언사인(부호 없는)(unsigned)), 곱셈, 절대 값(absolute value) 및 논리 연산자(logical operator)를 위한 벡터 연산들을 포함한다. 벡터 ALU 연산은 다른 피연산자 크기에서 수행될 수 있다. 피연산자 크기의 예에는 8 비트, 16 비트, 32 비트 및 부동 소수점 값(floating point value)이 포함된다. 일부 실시 예에서, 상이한 피연산자 크기는 레지스터 앨리어싱(register aliasing) 및/또는 연산의 오피코드에 기초하여 결정된다. 예를 들어, 8 비트 피연산자에 대한 벡터 가산 연산(vector add operation)은 8 비트 벡터 레지스터를 사용한다. 도 4a 및 도 4b와 관련하여 보다 상세하게 설명된 바와 같이, 레지스터 앨리어싱(register aliasing)은 동일한 메모리 위치가 다른 앨리어스(alias)를 사용하여 참조될 수 있게 한다. 예를 들어, 32 비트 메모리 블록은 바람직한 결과(desired result)에 따라 단일 4 바이트 피연산자, 2 개의 2 바이트 피연산자 또는 4 개의 1 바이트 피연산자로 참조될 수 있다. 다양한 실시 예에서, 벡터 계산 유닛의 각각의 처리 엘리먼트는 다른 처리 엘리먼트와 병렬로 동일한 ALU 연산(예를 들어, 가산, 감산, 곱셈 등)을 수행한다. 일부 실시 예에서, 출력 결과는 ALU 결과의 양자화된 버전이다. 예를 들어, 출력 결과는 ALU 결과보다 표현하는데 더 적은 비트가 필요한 양자화된 버전(quantized version)이다. 일부 실시 예에서, ALU 결과는 입력 피연산자보다 적은 비트를 사용하여 표현된 결과를 사용하여 계산된다. 예를 들어, 입력 피연산자는 각각 4 바이트이고 출력 결과의 크기는 1 바이트 일 수 있다.

[0075] 663단계에서는, 661에서 수행된 산술 논리 유닛(ALU) 연산의 벡터 결과가 벡터 계산 유닛으로부터 기록된다. 일부 실시 예에서, 벡터 결과는 출력 버스가 데이터를 수신할 수 없는 경우 다음 ALU 연산 동안 처리가 계속되도록 하는 출력 버퍼를 이용하여 기록된다. 일부 실시 예에서, 벡터 출력 결과는 각각 도 1, 도 2 및 도 3의 후처리 유닛(115, 215 및/또는 315)과 같은 후 처리 유닛으로 전송된다. 예를 들어, ALU 연산을 수행한 결과는 후처리 풀링 연산(post-processing pooling operation)을 수행하기 위한 후 처리 유닛에 기록된다. 일부 실시 예에서, 출력 벡터 결과는 정적 랜덤 액세스 메모리(SRAM)와 같은 메모리에 기록된다. 다양한 실시 예에서, 출력은 각각의 엘리먼트가 1 바이트의 크기를 갖는 96 엘리먼트 벡터와 같은 엘리먼트의 벡터로서 기록된다.

[0076] 도 7은 벡터 계산 유닛 명령을 위한 인코딩 포맷의 실시 예를 도시한 블록도이다. 도시된 예에서, 벡터 계산 유닛 명령(710)은 단일 명령에 의해 지정된 다수의 컴포넌트 명령의 인코딩을 도시한다. 벡터 계산 유닛 명령(740)은 단일 명령에 의해 지정된 다수의 컴포넌트 명령 각각의 포맷을 더 상세히 설명한다. 벡터 계산 유닛 명령(710)은 인코딩된 명령 트라이어드이며 로드 연산(711), 산술 논리 유닛(ALU) 연산(713) 및 저장 연산(715)을 포함한다. 벡터 계산 유닛 명령(740)은 필드(오피코드(741), 레지스터(743), 오피코드(751), 레지스터(753), 오피코드 구성 필드(opcode configuration field)(755), 직계 필드(immediate field)(757), 오피코드(761) 및 레지스터(763))를 포함한다. 벡터 계산 유닛 명령(710)에 의해 도시된 컴포넌트 명령(로드 연산, ALU 연산 및 저장 연산에 대응)에 대한 필드는 벡터 계산 유닛 명령(740)에 매핑된다. 벡터 계산 유닛 명령(740)은 인코딩된 로드 연산(오피코드(741), 레지스터(743)), 산술 논리 유닛 연산(오피코드(751), 레지스터(753), 오피코드 구성 필드(opcode configuration field)(755), 직계 필드(immediate field)(757)) 및 저장 연산(오피코드(761) 및 레지스터(763))을 포함한다. .

[0077] 일부 실시 예에서, 벡터 계산 유닛 명령은 3 개의 컴포넌트 명령들을 지정하는 명령 트라이어드이다. 예를 들어, 로드 연산, 산술 논리 유닛(ALU) 연산 및 저장 연산은 128 비트 형식을 사용하여 단일 명령으로 번들(bundle)될 수 있다. 다양한 실시 예에서, 더 크거나 더 작은 비트 포맷(bit format)이 3 개의 컴포넌트 명령들을 적절히 번들(bundle)하는데 이용될 수 있다. 일부 실시 예에서, 로드 및 저장 연산은 13 비트로 인코딩되고 ALU 연산은 64 비트로 인코딩된다. 다양한 실시 예에서, 번들된 로드(bundled load), 저장 및 ALU 연산에 의해 사용되지 않는 임의의 나머지 비트(remaining bit)는 패딩 비트(padding bit)이다. 일부 실시 예에서, 오피코드는 8 비트로 인코딩되고, 레지스터는 5 비트로 인코딩되며, 직계 필드는 32 비트로 인코딩된다. 다양한 실시 예에서, 상이한 길이의 인코딩이 적절하게 이용될 수 있고 명령 크기, 지원되는 벡터 연산의 수, 레지스터의 수, 벡터 크기 및/또는 다른 적절한 팩터(factor)에 기초한다. 일부 시나리오에서는 하나 이상의 컴포넌트 명령이

사용되지 않는 경우 노옵 연산이 사용된다.

[0078] 도시된 예에서, 벡터 계산 유닛 명령(740)의 인코딩된 로드 연산은 오피코드(741) 및 레지스터(743)를 포함한다. 오피코드(761)는 벡터 로드 연산에 대응하고 레지스터(743)는 로드 연산에 대응하는 목적지 벡터 레지스터이다. 예를 들어, 오피코드(741)는 데이터를 로드하는 디큐 연산을 위한 오피코드를 저장하는데 사용될 수 있고 레지스터(743)는 로드된 데이터를 저장하기 위한 목적지 레지스터이다. 다양한 실시 예에서, 로드 연산은 벡터 계산 유닛에 의해 처리하기 위해 입력 데이터의 벡터를 벡터 레지스터에 로드하기 위해 사용된다. 일부 실시 예에서, 오피코드(741)는 8 비트 필드이고 레지스터(743)는 5 비트 필드이다.

[0079] 도시된 예에서, 벡터 계산 유닛 명령(740)의 인코딩된 저장 연산은 연산 오피코드(761) 및 레지스터(763)를 포함한다. 오피코드(761)는 벡터 저장 연산에 대응하고 레지스터(763)는 저장 연산에서 데이터의 벡터를 판독해야 하는 대응하는 소스 벡터 레지스터이다. 예를 들어, 오피코드(761)는 레지스터(763)로부터 정적 랜덤 액세스 메모리(SRAM)와 같은 외부 메모리로 데이터를 저장하는 저장 동작을 위한 오피코드를 저장하는데 사용될 수 있다. 일부 실시 예에서, 저장에 사용되는 메모리의 시작 어드레스(start address)는 메모리 위치를 참조하기 위해 기록 포인터(write pointer)를 사용하여 외부 시퀀서(external sequencer) 또는 제어 유닛에 의해 유지된다. 일부 실시 예에서, 저장 연산은 데이터의 벡터를 출력 데이터 버스(output data bus)에 기록하는데 사용된다. 일부 실시 예에서, 오피코드(761)는 8 비트 필드이고 레지스터(763)는 5 비트 필드이다.

[0080] 도시된 예에서, 인코딩 된 산술 논리 유닛(ALU) 동작은 오피코드(751), 레지스터(753), 오피코드 구성 필드(755) 및 직계 필드(757)를 포함한다. 오피코드(751)는 ALU 오피코드를 인코딩하는 데 사용된다. 예를 들어, ALU 오피코드는 특히 가산(사인 및 언사인), 감산(사인 및 언사인), 곱셈, 절대 값 및 논리 연산자(logical operator)에 대한 벡터 연산에 대응하는 오피코드를 포함할 수 있다. 벡터 ALU 연산에 따라, 연산은 필드(레지스터(753), 오피코드 구성 필드(opcode configuration field)(755) 및 직계 필드(757))를 이용할 수 있다. 일부 실시 예에서, 레지스터(753)는 3 개의 소스 레지스터 및 하나의 목적지 레지스터를 포함하여 최대 4 개의 벡터 레지스터를 지정한다. 일부 실시 예에서, 레지스터(753)는 20 비트 필드이고 각각의 레지스터에 대해 5 비트를 이용한다.

[0081] 일부 실시 예에서, 인코딩된 산술 논리 유닛(ALU) 연산은 특정 ALU 연산에 의해 이용되는 오피코드 구성 필드(755)를 포함한다. 일부 실시 예에서, 오피코드 구성 필드(755)는 5 비트 필드이고 레지스터 크기 필드(register size field)(2 비트), 마스크 비트(mask bit)(1 비트) 및 직계 유효 비트(immediate valid bit)(1 비트)를 포함한다. 예를 들어, 일부 시나리오에서, 레지스터 크기 필드(2 비트)에 저장된 값은 레지스터의 크기(예를 들어, 8 비트, 16 비트 또는 32 비트)를 지정하는 데 사용될 수 있다. 추가적인 예로서, 마스크 비트(1 비트)는 직계 필드(757)를 비트 마스크로서 처리하기 위해 이용될 수 있고 직계 유효 비트(1 비트)는 직계 필드(757)의 유효성을 식별하기 위해 이용될 수 있다. 다양한 실시 예에서, 직계 필드(757)는 직계 필드를 요구하는 ALU 연산에 이용되는 32 비트 필드이다. 예를 들어, 벡터 이동 연산(vector move operation)은 32 비트 값을 직계 필드(757)에서 목적지 벡터 레지스터로 이동시키도록 구성될 수 있다.

[0082] 일부 실시 예에서, 벡터 계산 유닛은 벡터 비트 마스크(vector bit mask)를 벡터 마스크 레지스터(vector mask register)에 로드하기 위해 벡터 마스크 이동 명령(mask move instruction)(미도시)을 지원한다. 일부 실시 예에서, 벡터 마스크 이동 명령은 대응하는 오피코드 필드, 목적지 레지스터 필드(destination register field) 및 직계 필드를 포함한다. 예를 들어, 벡터 마스크 이동은 직계 필드에 저장된 벡터 비트 마스크를 벡터 마스크 레지스터에 로드한다. 일부 실시 예에서, 벡터 계산 유닛에 의해 지원되는 벡터들의 크기(예를 들어, 96 엘리먼트 폭)는 비트 마스크를 저장하기에 충분히 큰 직계 필드(예를 들어, 96 비트)를 필요로 한다. 일부 실시 예에서, 벡터 마스크 이동 명령은 벡터 계산 유닛 명령(710, 740)의 인코딩 포맷으로 제한되지 않는다. 예를 들어, 직계 필드의 크기에 따라 벡터 마스크 이동은 다른 컴퍼넌트 명령과 번들(bundle)로 제공되지 않을 수 있다.

[0083] 다양한 실시 예에서, 벡터 계산 유닛 명령의 컴포넌트 명령은 도 5의 프로세스를 사용하여 함께 번들(bundle)된다. 일부 실시 예에서, 도 7의 인코딩 포맷은 각각 벡터 엔진(111, 211, 311) 및/또는 벡터 계산 유닛(401), 도 1, 도 2, 도 3 및 도 4a의 벡터 계산 유닛(401)과 같은 벡터 계산 유닛에 의해 이용된다. 일부 실시 예에서, 벡터 계산 유닛 명령은 마이크로 프로세서 시스템의 시퀀서 또는 시퀀서를 포함하는 제어 유닛에 의해 벡터 계산 유닛에 발행된다.

[0084] 도 8은 벡터 계산 유닛에 의해 단일 벡터 계산 유닛 명령을 수행하기 위한 프로세스의 실시 예를 도시한 흐름도이다. 도 8의 프로세스는 벡터 계산 유닛의 처리 엘리먼트를 이용하여 병렬로 벡터의 엘리먼트 상에서 벡터 계산 유닛에 의해 수행될 수 있다. 일부 실시 예에서, 도 8의 프로세스는 각각 벡터 엔진(111, 211, 311) 및/또는

도 1, 도 2, 도 3 및 도 4a의 벡터 계산 유닛(401)과 같은 벡터 계산 유닛에 의해 수행된다.

[0085] 801단계에서는, 벡터 계산 유닛 명령이 페치(fetch)된다. 일부 실시 예에서, 명령은 명령 버퍼(instruction buffer) 및/또는 커맨드 큐(command queue)로부터 페치(fetch)된다. 다양한 실시 예에서, 명령 버퍼는 수행될 현재 명령을 참조하기 위한 하나 이상의 포인터를 포함한다. 다양한 실시 예에서, 명령 버퍼는 벡터 계산 유닛 명령의 캐시로서 작용한다.

[0086] 821단계에서는, 벡터 계산 유닛 명령이 디코딩된다. 예를 들어, 명령 트라이어드인 벡터 계산 단위 명령어는 3개의 컴포넌트 명령어로 디코딩된다. 다양한 실시 예에서, 각각의 컴포넌트 명령에 의해 이용되는 아규먼트 및 필드가 디코딩된다. 예를 들어, 도 7의 레지스터(753)와 같은 레지스터 필드에 의해 지정된 벡터 레지스터는 소스 및 목적지 레지스터로 디코딩된다.

[0087] 831단계에서는, 컴포넌트 명령(component instruction)이 발행된다. 일부 실시 예에서, 컴포넌트 명령의 발행은 리소스 및/또는 데이터 위험이 존재하는지를 결정하는 것을 포함한다. 위험이 존재하는 경우, 일부 실시 예에서, 벡터 계산 유닛은 위험이 해결되기를 기다린다. 예를 들어, 이전 클록 사이클(clock cycle)에서의 로드 연산으로 인한 자원 위험(resource hazard)의 경우, 벡터 계산 유닛은 로드가 완료되고 자원이 이용 가능할 때까지 하나 이상의 클록 사이클을 기다린다.

[0088] 일부 실시 예에서, 복수의 컴포넌트 명령은 함께 발행되고 병렬로 실행된다. 예를 들어, 로드 연산, 산술 논리 유닛(ALU) 연산 및 명령 트라이어드의 저장 연산은 동일한 클록 사이클 동안 함께 실행된다. 컴포넌트 명령이 함께 실행되는 시나리오에서, 로드 연산(845단계), ALU 연산(855단계) 및 저장 연산(865단계)과 그에 대응하는 노옵 대안(no-op alternative)(843, 854, 863단계)은 동일한 클록 사이클에서 개시되고 병렬로 실행이 진행된다.

[0089] 일부 실시 예에서, 상이한 컴포넌트 명령은 스태거된 시작(staggered start)으로 실행된다. 예를 들어, 일부 실시 예에서, 로드 연산이 먼저 실행된 다음, 산술 논리 유닛(ALU) 연산이 수행된 다음 저장 연산이 수행된다. 스태거된 시나리오(staggered scenario)에서, 제1 벡터 계산 유닛 명령의 ALU 연산은 다음 벡터 계산 유닛 명령의 로드 연산과 병행하여 실행될 수 있다.

[0090] 다양한 실시 예에서, 상이한 산술 논리 유닛(ALU) 연산을 포함하는 상이한 연산은 완료하는데 하나 이상의 클록 사이클(clock cycle)이 필요하고, 동일한 클럭 사이클이 끝날 때까지 상이한 연산이 완료되는 것을 보장하지 않는다. 일부 실시 예에서, 페치(fetch)(801단계), 디코드(821단계) 및 발행(issue)(831단계) 중 하나 이상이 동일한 명령 사이클(instruction cycle) 동안 수행될 수 있다.

[0091] 841단계에서는, 벡터 계산 유닛 명령이 로드 연산을 포함하는지에 대한 결정이 이루어진다. 예를 들어, 일부 시나리오에서는 로드 연산이 수행되지 않아야 함을 나타내기 위해 로드 연산이 노옵(no-op)으로 대체될 수 있다. 노옵에 응답하여 처리는 843단계로 계속된다. 로드 연산이 존재하는 경우, 처리는 845단계로 계속된다.

[0092] 843단계에서는, 노옵이 처리되고 로드 연산이 수행되지 않는다. 예를 들어, 로드 명령이 841단계에서 명령에 존재하지 않고 대신 노옵에 대한 오피코드가 사용되었다.

[0093] 845단계에서는, 로드 연산이 벡터 계산 유닛에 의해 실행된다. 예를 들어, 벡터 엔진 입력 큐(207)와 같은 선입선출 큐로부터 입력 벡터를 로드하기 위한 디큐 연산이 수행된다.

[0094] 851단계에서는, 벡터 계산 유닛 명령이 산술 논리 유닛(ALU) 연산을 포함하는지에 대한 결정이 이루어진다. 예를 들어, 일부 시나리오에서 ALU 연산 노옵으로 대체되어 ALU 연산이 수행되지 않아야 함을 표시할 수 있다. 노옵에 응답하여 처리는 853단계로 계속된다. ALU 연산이 존재하는 경우, 처리는 계속 855단계로 진행된다.

[0095] 853단계에서는, 노옵이 처리되고 산술 논리 유닛(ALU) 연산이 수행되지 않는다. 예를 들어, 851다계에서 명령에 ALU 명령이 존재하지 않고 대신 노옵에 대한 오피코드가 사용된다.

[0096] 855단계에서는, 산술 논리 유닛(ALU) 연산이 벡터 계산 유닛에 의해 실행된다. 예를 들어, 벡터 가산 연산(vector add operation)에 응답하여, 벡터 계산 유닛의 산술 논리 유닛은 벡터 가산 연산을 수행하여 2 개의 소스 벡터 레지스터(source vector register)의 컨텐츠(content)를 가산하고 그 결과를 목적지 벡터 레지스터(destination vector register)에 저장한다. 일부 실시 예에서, 벡터 계산 유닛의 산술 논리 유닛은 도 4a의 산술 논리 유닛(ALU)(427)이다.

[0097] 861단계에서는, 벡터 계산 유닛 명령이 저장 연산을 포함하는지에 대한 결정이 이루어진다. 예를 들어, 일부 시

나리오에서, 저장 연산이 수행되지 않아야 함을 표시하기 위해 저장 연산이 노옵으로 대체될 수 있다. 노옵에 응답하여 처리는 863단계로 계속된다. 저장 연산이 존재하는 경우 처리는 계속 865단계로 진행된다.

[0098] 863단계에서는, 노옵이 처리되고 저장 연산이 수행되지 않는다. 예를 들어, 861단계에서 명령에 저장 명령이 없으면, 대신 노옵에 대한 오피코드가 사용된다.

[0099] 865단계에서는, 저장 연산이 벡터 계산 유닛에 의해 실행된다. 예를 들어, 벡터 레지스터의 벡터 데이터를 메모리에 저장하기 위한 저장 연산이 수행된다.

[0100] 도 9는 벡터 계산 유닛의 예시적인 명령 사이클을 도시한 도면이다. 도 9의 프로세스는 병렬로 수행되지만 스태거된 시작과 함께 3 개의 벡터 계산 유닛 명령의 시퀀스 및 오더의 예를 도시한다. 일부 실시 예에서, 도 9의 예시적인 명령 사이클은 각각 도 1, 도 2, 도 3 및 도 4a의 벡터 엔진(111, 211, 311) 및/또는 벡터 계산 유닛(401)에 의해 이용된다. 도 9의 예에서, 단일 명령으로서 번들된 컴포넌트 명령은 로드 연산이 먼저 실행된 다음 산술 논리 유닛(ALU) 연산이 수행된 다음 저장 연산이 뒤따르도록 스태거된 시작으로 실행된다. 일부 실시 예에서, 순차 벡터 계산 유닛 명령(sequential vector computational unit instruction)은 파이프 라인(pipeline)되지만 컴포넌트 명령은 병렬로 실행되며 도 9에 도시된 스태거된 시작을 따르지 않는다.

[0101] 도시된 예에서, 제1 명령 사이클(910)은 제1 벡터 계산 유닛 명령에 대응하는 페치 단계(911), 디코드 단계(921), 발행 단계(931), 로드 실행 단계(941), 산술 논리 유닛(ALU) 실행 단계(951) 및 저장 실행 단계(961)를 포함한다. 제2 명령 사이클(920)은 제2 벡터 계산 유닛에 대응하는 페치 단계(923), 디코드 단계(933), 발행 단계(943), 로드 실행 단계(953), 산술 논리 유닛(ALU) 실행 단계(963) 및 저장 실행 단계(973)를 포함한다. 제3 명령 사이클(930)은 제3 벡터 계산 유닛에 대응하는 페치 단계(935), 디코딩 단계(945), 발행 단계(955), 로드 실행 단계(965), 산술 논리 유닛(ALU) 실행 단계(975) 및 제3 벡터 계산 유닛에 대응하는 저장 실행 단계(985)를 포함한다. 일부 실시 예에서, 점선 수직선은 클록 사이클 경계(clock cycle boundary)이다. 다양한 실시 예에서, 동일한 클록 사이클 경계 내의 단계는 동일한 클록 사이클 동안 시작된다.

[0102] 일부 실시 예에서, 명령 사이클의 시작은 하나의 단계만큼 스테거(stagger)된다. 예를 들어, 제1 명령 사이클(910)은 제2 명령 사이클(920)에 비해 처리에서 한 단계 앞서고, 제3 명령 사이클(930) 보다 두 단계 앞서 있다. 임의의 주어진 클록 사이클 동안, 상이한 벡터 계산 유닛 명령은 상이한 스테이지와 관련된 하드웨어 자원(페치, 디코드, 발행, 로드 실행, 산술 논리 유닛(ALU) 실행 및 저장 실행)을 이용할 수 있다. 예를 들어, 제1, 제2 및 제3 명령 사이클(910, 920, 930)의 발행 스테이지(issue stage)(931), 디코드 스테이지(decode stage)(933) 및 페치 스테이지(fetch stage)(935)는 각각 동일한 클록 사이클 동안 실행된다. 다른 예로서, 제1, 제2 및 제3 명령 사이클(910, 920, 930)의 저장 실행 단계(store execution step)(961), 산술 논리 유닛(ALU) 실행 단계(arithmetic logic unit(ALU) execution step)(963) 및 로드 실행 단계(and load execution step)(965)는 각각 동일한 클록 사이클 동안 실행된다.

[0103] 일부 실시 예에서, 벡터 계산 유닛의 명령 사이클은 클록 사이클 당 하나의 벡터 계산 유닛 명령의 처리량(throughput)을 달성한다. 일부 실시 예에서, 페치, 디코드 및/또는 발행 단계는 단일 클록 사이클로 압축(compress)된다. 예를 들어, 일부 실시 예에서, 명령 버퍼는 페치 시간을 최소화하기 위해 이용되며 페치 및 디코드 단계는 함께 수행된다. 일부 실시 예에서, 명령 사이클의 각 단계는 완료하기 위해 하나 이상의 클록 사이클을 취할 수 있다. 일부 실시 예에서, 스테이지 자체는 파이프 라인(pipeline) 된다. 예를 들어, 실행 단계가 완료하기 위해 하나 이상의 사이클이 걸리는 경우, 실행 단계는 다수의 클록 사이클에 걸쳐 완료되도록 파이프라인 될 수 있다. 일부 실시 예에서, 다수의 실행 단계들이 파이프 라인 방식(pipelined manner)으로 병렬로 처리될 수 있고 각각의 실행 단계는 상이한 벡터 계산 유닛 명령에 대응할 수 있다. 일부 실시 예에서, 페치 단계(911, 923, 935)는 도 8의 801단계에 대응하고, 디코드 단계(921, 933, 945)는 도 8의 821단계에 대응하고, 발행 단계(931, 943, 955)는 도 8의 831단계에 대응하고, 로드 실행 단계(941, 953, 965)는 도 8의 845단계에 대응하고, 산술 논리 유닛 (ALU) 실행 단계(951, 963, 975)는 도 8의 855단계에 대응하고, 저장 실행 단계(961, 973, 985)는 도 8의 865단계에 대응한다.

[0104] 대안적인 실시 예(미도시)에서, 명령 사이클의 페치, 디코드 및 발행 단계는 도 9와 동일한 순서로 수행된다. 도 9의 예시적인 실시 예와 달리, 로드, 산술 논리 유닛(ALU) 및 저장 실행 단계는 동일한 클록 사이클 동안 함께 병렬로 실행된다. 예를 들어, 동일한 벡터 계산 유닛 명령의 로드 실행 단계(941), ALU 실행 단계(951) 및 저장 실행 단계(961)가 함께 실행된다.

[0105] 도 10은 계산 어레이의 계산 유닛의 실시 예를 도시한 블록도이다. 도시된 예에서, 계산 유닛(1000)은 계산 유

닛(1000)은 입력 값(가중치(1002), 데이터(1004) 및 ResultIn(1006)), 신호(ClearAcc 신호(1008), 클록 신호(1010), ResultEnable 신호(1012), ResultCapture 신호(1014) 및 ShiftEn 신호(1016)), 컴포넌트(어큐뮬레이터(1024), 멀티플렉서(multiplexer)(1026), 섀도우 레지스터(1028), 곱셈기(1030) 및 가산기(adder)(1032)), 논리(1034, 1036, 1038), 및 출력값(ResultOut(1050))을 포함한다. 일부 실시 예에서, 로직(1034, 1036, 1038)은 AND 게이트이다. 일부 실시 예에서, 추가 신호(additional signal)이 적절하게 포함된다. 다양한 실시 예에서, 도 10의 계산 유닛은 도 1의 매트릭스 프로세서(107)와 같은 계산 어레이의 계산 유닛(109)과 같은 복수의 계산 유닛 각각에 대해 반복된다. 계산 유닛(1000)은 계산 연산을 병렬로 구현하기 위해 이용될 수 있다. 다양한 실시 예에서, 계산 어레이의 각각의 계산 유닛은 다른 계산 유닛과 병렬로 계산을 수행한다. 다양한 실시 예에서, 계산 유닛(1000)은 하나 이상의 곱셈, 가산, 어큐뮬레이트 및/또는 시프트 연산을 수행하기 위한 기능을 포함하는 매트릭스 프로세서의 서브-회로이다. 예를 들어, 계산 유닛(1000)은 내적 연산을 수행하기 위한 기능을 포함하는 서브 회로일 수 있다. 다양한 실시 예에서, 계산 유닛(1000)은 도 1의 계산 유닛(109) 및/또는 도 1의 계산 유닛(209 및/또는 221-229)이다.

[0106] 일부 실시 예에서, 클록 신호(1010)는 계산 유닛(1000)에 의해 수신된 클록 신호이다. 다양한 실시 예에서, 계산 어레이의 각각의 계산 유닛은 동일한 클록 신호를 수신하고, 클록 신호는 각각의 계산 유닛의 처리를 다른 계산 유닛과 동기화하기 위해 이용된다.

[0107] 도시된 예에서, 곱셈기(1030)는 입력값 데이터(1004) 및 가중치(1002)에 대해 곱셈 연산(multiplication operation)을 수신하고 수행한다. 곱셈기(1030)의 출력은 가산기(1032)에 공급된다. 가산기(1032)는 곱셈기(1030)의 출력 및 로직(1034)의 출력에 대한 가산을 수신 및 수행한다. 가산기(1032)의 출력은 어큐뮬레이터(1024)에 공급된다. 일부 실시 예에서, 입력값인 데이터(1004) 및 가중치(1002)는 계산 유닛을 교차하고 대응하는 데이터 및/또는 가중치를 이웃하는 계산 유닛에 공급하는 라인이다. 예를 들어, 일부 실시 예에서, 데이터(1004)는 동일한 열(column)의 모든 계산 유닛에 공급되고 가중치(1002)는 동일한 행(row)의 모든 계산 유닛에 공급된다. 다양한 실시 예에서, 데이터(1004) 및 가중치(1002)는 각각 데이터 입력(103) 및 가중치 입력(105)으로부터 계산 유닛(1000)에 공급되는 입력 엘리먼트에 대응한다. 다양한 실시 예에서, 데이터(1004) 및 가중치(1002)는 각각 데이터 하드웨어 데이터 포맷터 및 가중치 하드웨어 데이터 포맷터로부터 계산 유닛(1000)에 공급되는 입력 엘리먼트에 대응한다.

[0108] 일부 실시 예에서, ClearAcc 신호(1008)는 어큐뮬레이터(1024)의 컨텐츠를 클리어(clear)한다. 예로서, 어큐뮬레이트 연산은 어큐뮬레이터(1024)를 클리어함으로써 리셋 될 수 있고 곱셈기(1030)의 결과를 어큐뮬레이트하는데 사용될 수 있다. 일부 실시 예에서, ClearAcc 신호(1008)는 새로운 내적 연산을 수행하기 위해 어큐뮬레이터(1024)를 클리어하는데 사용된다. 예를 들어, 엘리먼트 별 곱셈(elements-wise multiplications)은 곱셈기(1030)에 의해 수행되고 부분 내적 결과(partial-dot-product result)는 가산기(1032) 및 어큐뮬레이터(1024)를 사용하여 가산된다.

[0109] 다양한 실시 예에서, 어큐뮬레이터(1024)는 가산기(1032)의 결과를 어큐뮬레이트 할 수 있고, 간접적으로 곱셈기(1030)의 결과를 어큐뮬레이트 할 수 있는 어큐뮬레이터이다. 예를 들어, 일부 실시 예에서, 어큐뮬레이터(1024)는 ClearAcc 신호(1008)의 상태에 기초하여 곱셈기(1030)의 결과와 어큐뮬레이터(1024)의 컨텐츠를 어큐뮬레이트 하도록 구성된다. 다른 예로서, ClearAcc 신호(1008)의 상태에 기초하여, 어큐뮬레이터(1024)에 저장된 현재 결과는 가산기(1032)에 의해 무시(ignore)될 수 있다. 도시된 예에서, 어큐뮬레이터(1024)는 32 비트 폭 어큐뮬레이터(32-bit wide accumulator)이다. 다양한 실시 예에서, 어큐뮬레이터(1024)는 적절하게, 예를 들어, 8 비트, 16 비트, 64 비트 등의 크기가 상이 할 수 있다. 다양한 실시 예에서, 계산 어레이의 복수의 계산 유닛의 각각의 어큐뮬레이터는 동일한 크기이다. 다양한 실시 예에서, 어큐뮬레이터(1024)는 데이터를 어큐뮬레이트 및 저장하거나, 데이터를 어큐뮬레이트 및 삭제하거나, 또는 단지 데이터를 클리어할 수 있다. 일부 실시 예에서, 어큐뮬레이터(1024)는 어큐뮬레이트 레지스터(accumulation register)로서 구현될 수 있다. 일부 실시 예에서, 어큐뮬레이터(1024)는 레지스터를 포함하는 한 세트의 산술 논리 유닛(ALU)을 포함할 수 있다.

[0110] 일부 실시 예에서, ResultEnable 신호(1012)는 데이터(1004)가 유효하다는 결정에 응답하여 활성화된다. 예를 들어, ResultEnable 신호(1012)는 곱셈기(1030) 및 가산기(1032)에 의한 어큐뮬레이터(1024) 로의 처리와 같은 계산 유닛에 의한 처리를 가능하게 하도록 할 수 있다.

[0111] 일부 실시 예에서, ResultCapture 신호(1014)는 멀티플렉서(multiplexer)(1026)의 기능성을 결정하기 위해 이용된다. 멀티플렉서(1026)는 입력 ResultIn(1006), 어큐뮬레이터(1024)의 출력 및 ResultCapture 신호(1014)를 수신한다. 다양한 실시 예에서, ResultCapture 신호(1014)는 ResultIn(1006) 또는 어큐뮬레이터(1024)의 출력

이 멀티플렉서(1026)의 출력으로서 통과할 수 있게 하는데 사용된다. 일부 실시 예에서, 멀티플렉서(1026)는 출력 레지스터로서 구현된다. 일부 실시 예에서, ResultIn(1006)은 계산 유닛(1000)과 동일한 열의 계산 유닛에 연결된다. 예를 들어, 이웃 계산 유닛의 출력은 입력값 ResultIn(1006)으로서 계산 유닛(1000)에 공급된다. 일부 실시 예에서, 이웃하는 계산 유닛의 입력은 계산 유닛의 대응하는 ResultOut 값이다.

[0112] 일부 실시 예에서, 섀도우 레지스터(1028)는 멀티플렉서(1026)의 출력을 입력으로서 수신한다. 일부 실시 예에서, 섀도우 레지스터(1028)는 ResultCapture 신호(1014)의 값에 따라 멀티플렉서(1026)를 통해 어큐뮬레이터(1024)의 출력을 수신하도록 구성된다. 도시된 예에서, 섀도우 레지스터(1028)의 출력은 출력값 ResultOut(1050)이다. 다양한 실시 예에서, 일단 결과가 섀도우 레지스터(1028)에 삽입되면, 어큐뮬레이터(1024)는 새로운 계산을 시작(commence)하는데 사용될 수 있다. 예를 들어, 최종 내적 결과가 섀도우 레지스터(1028)에 저장되면, 어큐뮬레이터(1024)는 클리어되고 부분 결과(partial result) 및 새로운 가중치 및 데이터 입력값에 대한 새로운 내적 연산의 최종 결과를 어큐뮬레이트 및 저장하기 위해 사용될 수 있다. 도시된 예에서, 섀도우 레지스터(1028)는 신호인 ShiftEn 신호(1016)를 수신한다. 다양한 실시 예에서, ShiftEn 신호(1016)는 섀도우 레지스터(1028)에 값의 저장을 가능하게 하거나 불가능하게 하기 위해 사용된다. 일부 실시 예에서, ShiftEn 신호(1016)는 섀도우 레지스터(1028)에 저장된 값을 출력값인 ResultOut(1050)으로 시프트하는데 사용된다. 예를 들어, ShiftEn 신호(1016)가 활성화되면, 섀도우 레지스터(1028)에 저장된 값은 섀도우 레지스터(1028) 밖으로 출력값 ResultOut(1050)으로서 시프트 된다. 일부 실시 예에서, ResultOut(1050)은 이웃하는 계산 유닛의 입력값 ResultIn에 연결된다. 일부 실시 예에서, 계산 유닛의 열의 마지막 셀은 계산 어레이의 출력에 연결된다. 다양한 실시 예에서, 계산 어레이의 출력은 벡터 프로세싱을 위해도 1의 벡터 엔진(111)과 같은 벡터 엔진으로 공급된다. 예를 들어, 도 1의 계산 셀(computation cell)(109)과 같은 계산 셀의 출력 ResultOut(1050)은 도 1의 벡터 엔진(111)의 처리 엘리먼트(113)와 같은 벡터 엔진의 처리 엘리먼트에 공급될 수 있다.

[0113] 도시된 예에서, 섀도우 레지스터(1028)는 32 비트 폭이다. 다양한 실시 예에서, 쉐도우 레지스터(1028)는 적절하게 예를 들어, 8 비트, 16 비트, 64 비트 등 다르게 크기가 정해질 수 있다. 다양한 실시 예에서, 계산 어레이의 복수의 계산 유닛의 각각의 섀도우 레지스터는 동일한 크기이다. 다양한 실시 예에서, 섀도우 레지스터(1028)는 어큐뮬레이터(1024)와 동일한 크기이다. 다양한 실시 예에서, 멀티플렉서(1026)의 크기는 어큐뮬레이터(1024) 및/또는 쉐도우 레지스터(1028)의 크기(예를 들어, 동일한 크기 이상)에 기초한다.

[0114] 일부 실시 예에서, 로직(1034, 1036, 1038)은 제어 신호와 같은 신호를 수신하여 계산 유닛(1000)의 기능을 활성화 및/또는 구성한다. 다양한 실시 예에서, 로직(1034, 1036, 1038)은 AND 게이트 및/또는 AND 게이트에 대응하는 기능을 사용하여 구현된다. 예를 들어, 상술한 바와 같이, 로직(1034)은 ClearAcc 신호(1008) 및 어큐뮬레이터(1024)에 저장된 값에 대응하는 입력 값을 수신한다. ClearAcc 신호(1008)에 기초하여, 로직(1034)의 출력이 결정되어 가산기(1032)에 공급된다. 다른 예로서, 로직(1036)은 ResultEnable 신호(1012) 및 클록 신호(1010)를 수신한다. ResultEnable 신호(1012)에 기초하여, 로직(1036)의 출력이 결정되어 어큐뮬레이터(1024)에 공급된다. 다른 예로서, 로직(1038)은 ShiftEn 신호(1016) 및 클록 신호(1010)를 수신한다. ShiftEn 신호(1016)에 기초하여, 로직(1038)의 출력이 결정되어 섀도우 레지스터(1028)에 공급된다.

[0115] 다양한 실시 예에서, 계산 유닛들은 곱셈, 가산 연산, 및 쉬프트 연산을 동시에, 즉 단일 사이클 내에서 수행하여, 각 사이클에서 발생하는 총 연산 수를 두 배로 증가시킬 수 있다. 일부 실시 예에서, 결과들은 단일 클록 사이클에서, 즉 중간 실행 및 저장 연산의 필요없이 멀티플렉서(1026)로부터 섀도우 레지스터(1028)로 이동된다. 다양한 실시 예에서, 클록 사이클은 클록 신호(1010)에서 수신된 신호에 기초한다.

[0116] 다양한 실시 예에서, 입력값인 가중치(1002) 및 데이터(1004)는 8 비트 값이다. 일부 실시 예에서, 가중치(1002)는 사인된(부호가 있는)(signed) 값이고 데이터(1004)는 언사인(부호가 없는)(unsign)이다. 다양한 실시 예에서, 가중치(1002) 및 데이터(1004)는 적절히 사인(부호가 있는)(signed)되거나 언사인(부호가 없는)(unsigned)일 수 있다. 일부 실시 예에서, ResultIn(1006) 및 ResultOut(1050)은 32 비트 값이다. 다양한 실시 예에서, ResultIn(1006) 및 ResultOut(1050)은 입력 피연산자인 가중치(1002) 및 데이터(1004) 보다 더 많은 수의 비트를 사용하여 구현된다. 다수의 비트를 이용함으로써, 예를 들어 내적 결과를 계산하기 위해 다수의 가중치(1002)와 데이터(1004)의 쌍을 곱한 결과가 스칼라 결과(scalar result)를 오버플로(overflow)하지 않고 어큐뮬레이트 될 수 있다.

[0117] 일부 실시 예에서, 계산 유닛(1000)은 어큐뮬레이터(1024)에서 중간(intermediate) 및/또는 최종 계산 결과(final computation result)를 생성한다. 최종 계산 결과는 멀티플렉서(1026)를 통해 섀도우 레지스터(1028)에

저장된다. 일부 실시 예에서, 멀티플렉서(1026)는 출력 레지스터로서 기능하고 어큐뮬레이터(1024)의 출력을 저장한다. 다양한 실시 예에서, 최종 계산 결과는 컨볼루션 연산(convolution operation)의 결과이다. 예를 들어, ResultOut(1050)에서의 최종 결과는 가중치(1002)를 사용하여 입력값으로서 계산 유닛(1000)에 의해 수신된 필터와 데이터(1004)를 사용하여 입력값으로서 계산 유닛(1000)에 의해 수신된 센서 데이터의 2 차원 영역 사이의 컨벌루션(convolution) 결과이다.

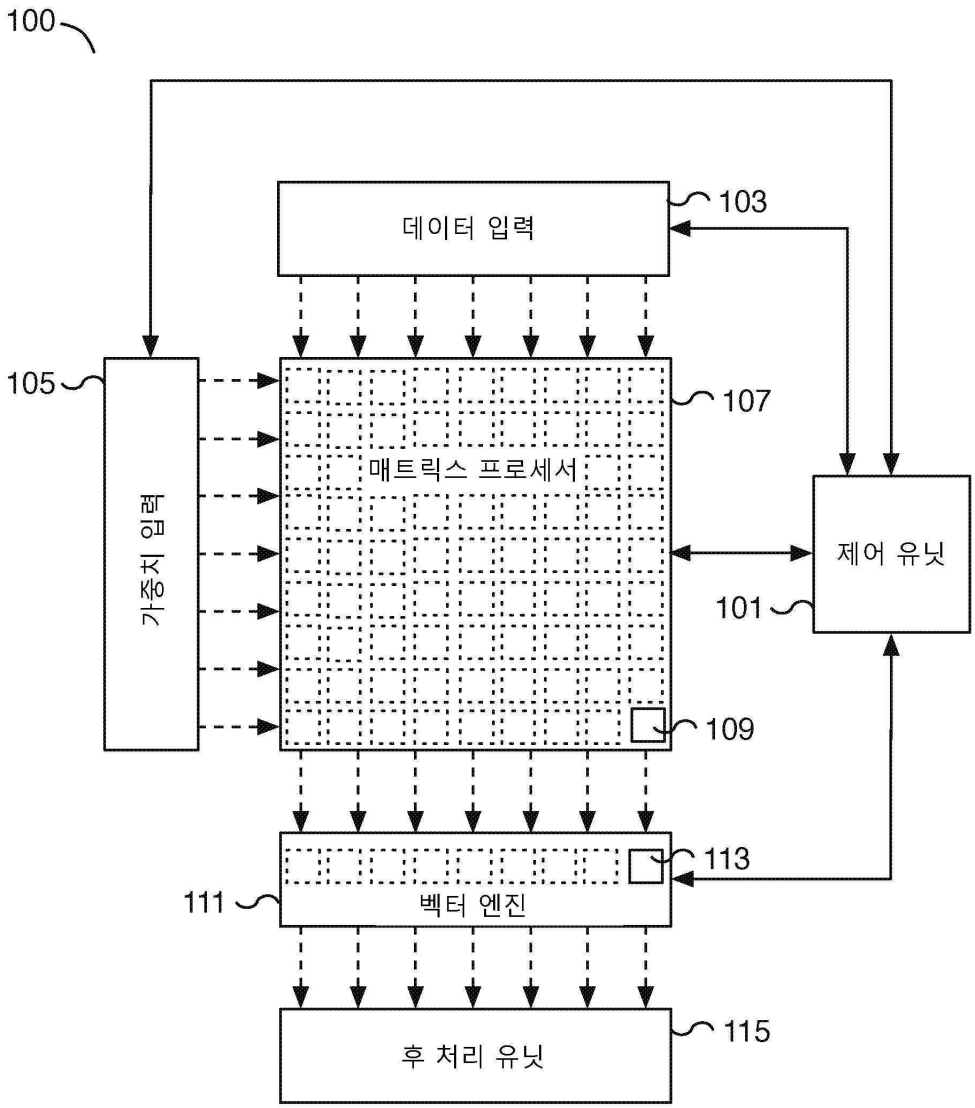
[0118] 일 예로서, 컨볼루션 연산은 센서 데이터의 영역에 대응하는 2x2 데이터 입력 매트릭스[d0 d1; d2 d3] 및 2x2 가중치의 행렬 [w0 w1; w2 w3]에 대응하는 필터 filter ()상에서 계산 유닛(1000)을 사용하여 수행될 수 있다. 2x2 데이터 입력 매트릭스는 제1 행[d0 d1] 및 제2 행[d2 d3]을 갖는다. 필터 매트릭스는 제1 행[w0 w1] 및 제2 행[w2 w3]을 갖는다. 다양한 실시 예에서, 계산 유닛(1000)은 클록 사이클 당 하나의 엘리먼트인 1 차원 입력 벡터[d0 d1 d2 d3]로서 데이터(1004)를 통해 데이터 매트릭스를 수신하고, 클럭 사이클 당 하나의 엘리먼트인 1 차원 입력 벡터[w0 w1 w2 w3 ]로서 가중치(1002)를 통해 가중치 매트릭스를 수신한다. 계산 유닛(1000)을 사용하여, 2 개의 입력 벡터의 내적은 ResultOut(1050)에서 스칼라 결과를 생성하기 위해 수행된다. 예를 들어, 곱셈기(1030)는 입력 가중치 및 데이터 벡터의 각각의 대응하는 엘리먼트를 곱하기 위해 사용되고 결과는 어큐뮬레이터(1024)에서 이전 결과에 저장되고 가산된다. 예를 들어, 엘리먼트(d0)에 엘리먼트(w0)을 곱한 결과(예를 들어, d0 * w0)는 먼저 클리어 어큐뮬레이터(1024)에 저장된다. 다음에, 엘리먼트(d1)에 엘리먼트(w1)을 곱하고, 어큐뮬레어트(1024)에 저장된 이전 결과(예를 들어, d0 * w0)에 가산기(1032)를 사용하여 d0 * w0 + d1 * w1의 등가물(equivalent)을 계산한다. 처리는 어큐뮬레이터(1024)에서 d0 * w0 + d1 * w1 + d2 * w2의 등가물을 계산하기 위해 엘리먼트(d2 및 w2)의 세번째 쌍으로 계속한다. 엘리먼트의 마지막 쌍이 곱해지고 내적의 최종 결과(예를 들어, d0 * w0 + d1 * w1 + d2 * w2 + d3 * w3)가 어큐뮬레이터(1024)에 저장된다. 그런 다음 내적 결과는 섀도우 레지스터(1028)에 카피(copy)된다. 섀도우 레지스터(1028)에 저장되면, 예를 들어, 센서 데이터의 상이한 영역을 사용하여 새로운 내적 연산이 개시될 수 있다. ShiftEn 신호(1016)에 기초하여, 섀도우 레지스터(1028)에 저장된 내적 결과는 섀도우 레지스터(1028)로부터 ResultOut(1050)으로 시프트된다. 다양한 실시 예에서, 가중치 및 데이터 매트릭스는 상기 예와 다른 디멘션(dimension) 일 수 있다. 예를 들어, 더 큰 디멘션이 사용될 수 있다.

[0119] 일부 실시 예에서, 바이어스 파라미터(a bias parameter)가 어큐뮬레이터(1024)를 사용하여 내적 결과에 가산되고 부가된다. 일부 실시 예에서, 바이어스 파라미터는 다른 입력값으로서 곱셈 식별 엘리먼트(multiplication identity element)와 함께 가중치(1002) 또는 데이터(1004)에서 입력으로서 수신된다. 바이어스 파라미터는 식별 엘리먼트에 곱하여 바이어스 파라미터를 보존하고 곱셈 결과(예를 들어, 바이어스 파라미터)는 가산기(1032)를 사용하여 내적 결과에 가산된다. 바이어스 값에 의한 내적 결과 오프셋(dot-product result offset) 인 가산 결과(addition result)는 어큐뮬레이터(1024)에 저장되고 나중에 섀도우 레지스터(1028)를 사용하여 ResultOut(1050)에서 시프트된다. 일부 실시 예에서, 바이어스는 도 1의 벡터 엔진(111)과 같은 벡터 엔진을 사용하여 도입된다.

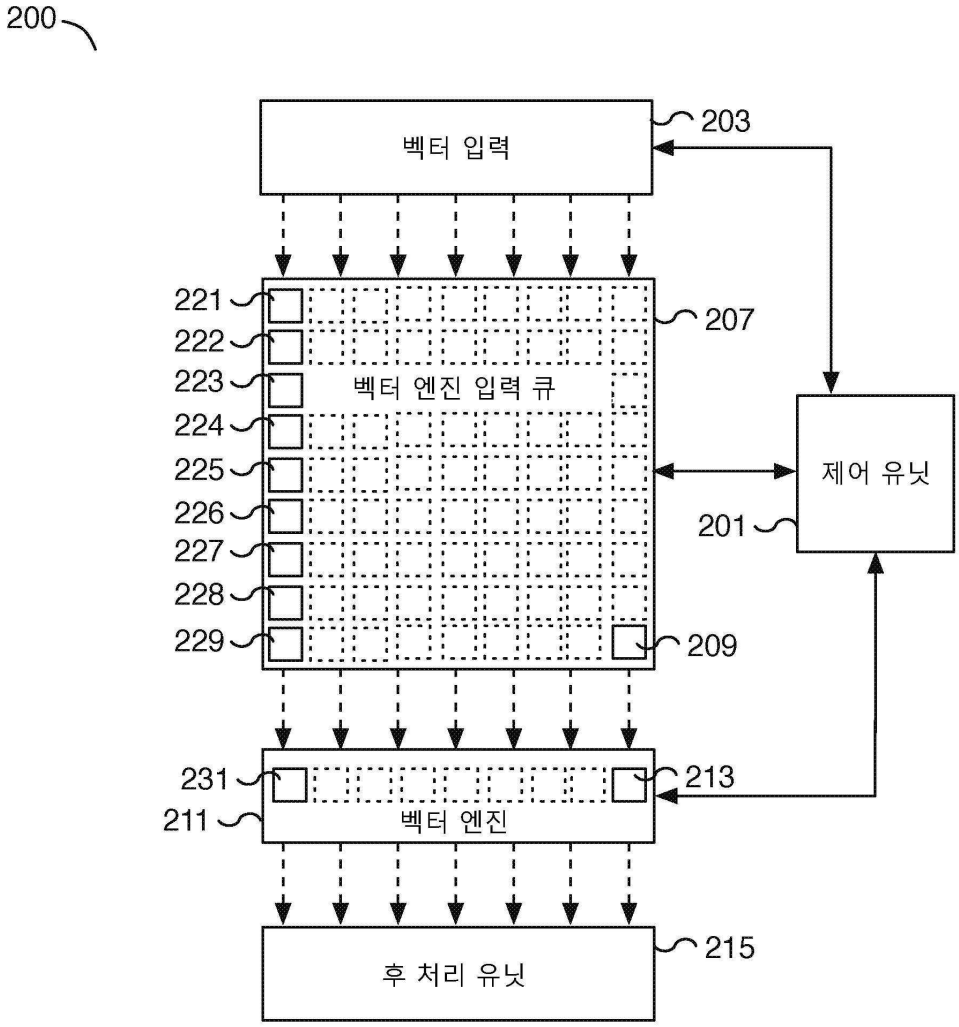
[0120] 상술한 실시 예들이 이해의 명확성을 위해 일부 상세하게 설명되었지만, 본 발명은 제공된 세부 사항들로 제한되지 않는다. 본 발명을 구현하는 많은 대안적인 방법이 있다. 개시된 실시 예는 예시적이고 제한적이지 않다.

도면

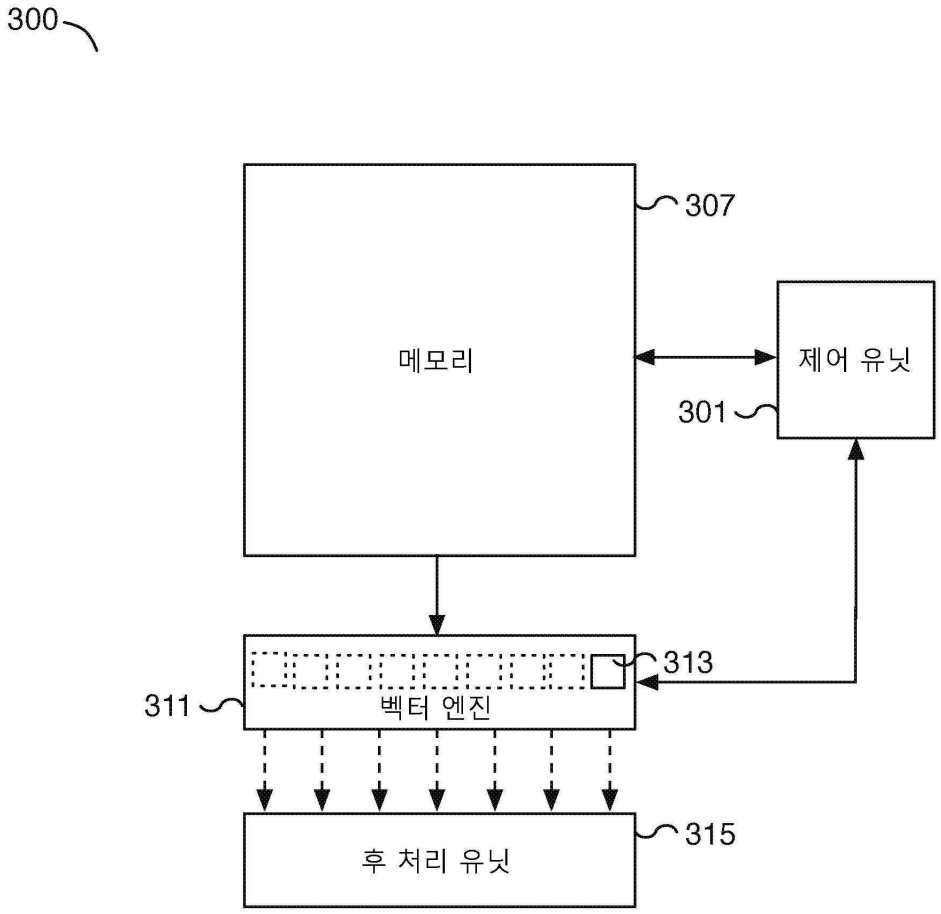
도면1


100
데이터 입력
103
105
가중치 입력
107
매트릭스 프로세서
제어 유닛
101
109
113
111
벡터 엔진
후 처리 유닛
115

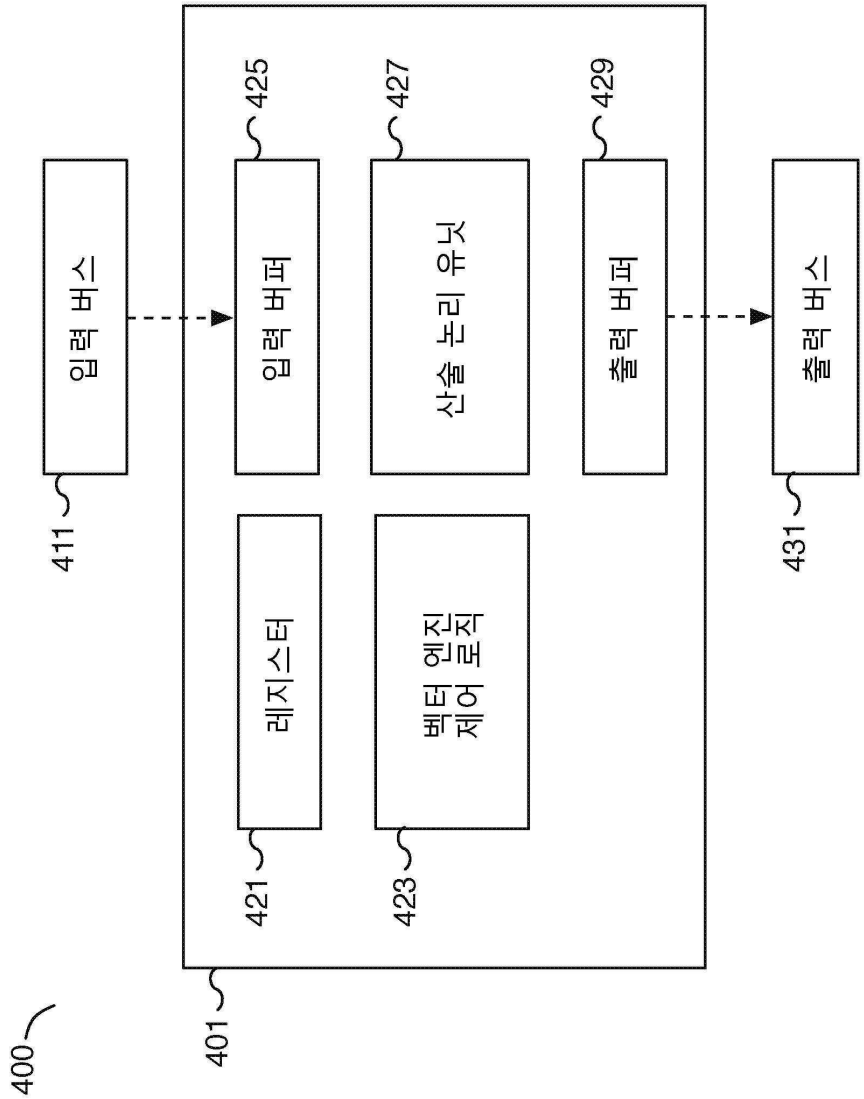
도면2


200
벡터 입력
203
221
222
223
224
225
226
227
228
229
벡터 엔진 입력 큐
207
209
제어 유닛
201
231
211
벡터 엔진
213
후 처리 유닛
215

도면3



도면4a

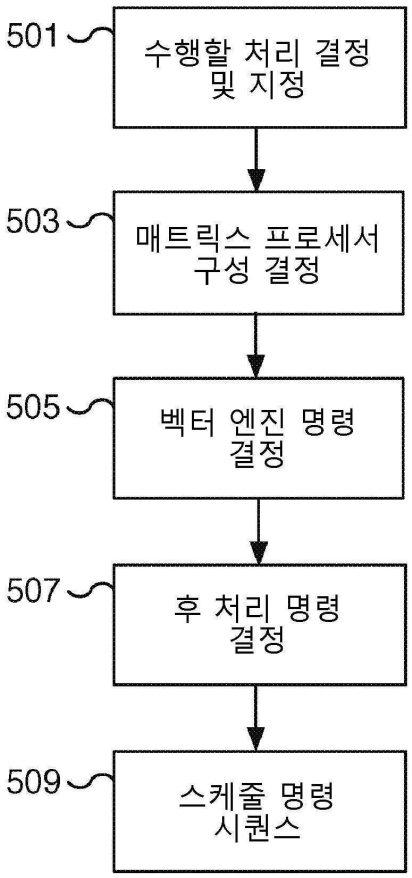



도면4b

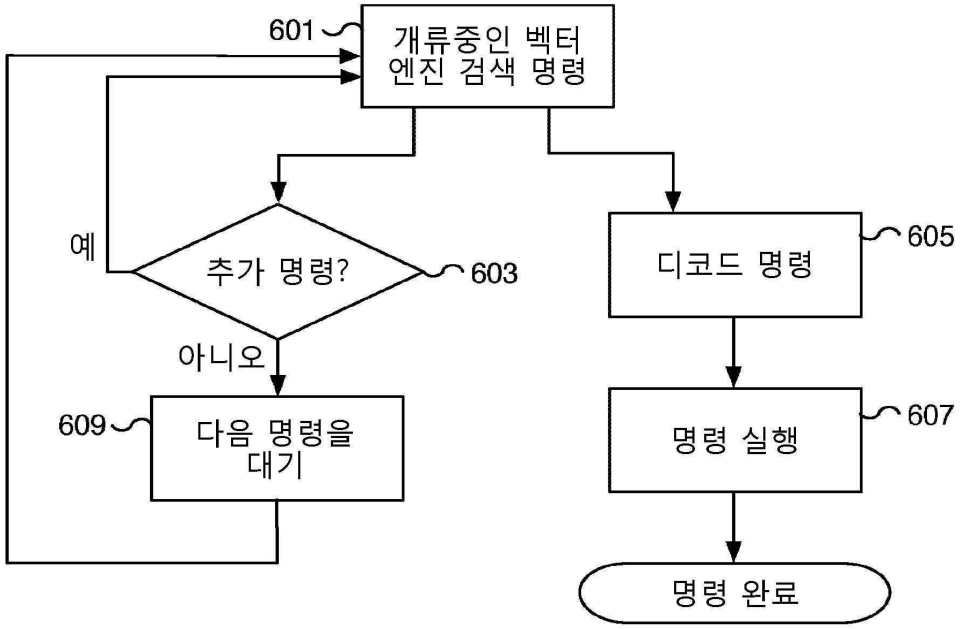
테이블 450

| | | | | |
|---|---|---|---|---|
| 451 | Byte 3 | Byte 2 | Byte 1 | Byte 0 |
| 453 | RD0 | | | |
| 455 | RW1 | | RW0 | |
| 457 | RB3 | RB2 | RB1 | RB0 |
| 463 | RD1 | | | |
| 465 | RW3 | | RW2 | |
| 467 | RB7 | RB6 | RB5 | RB4 |
| 473 | RD7 | | | |
| 475 | RW15 | | RW14 | |
| 477 | RB31 | RB30 | RB29 | RB28 |

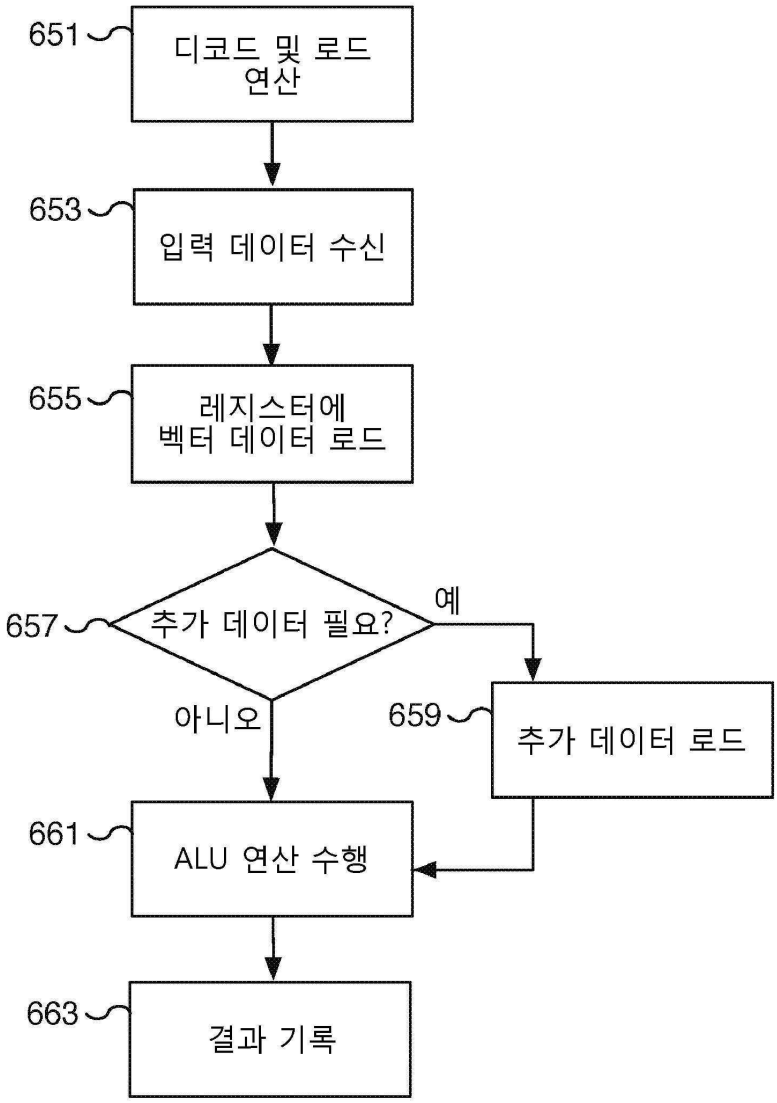
도면5


501
수행할 처리 결정 및 지정
503
매트릭스 프로세서 구성 결정
505
벡터 엔진 명령 결정
507
후 처리 명령 결정
509
스케줄 명령 시퀀스

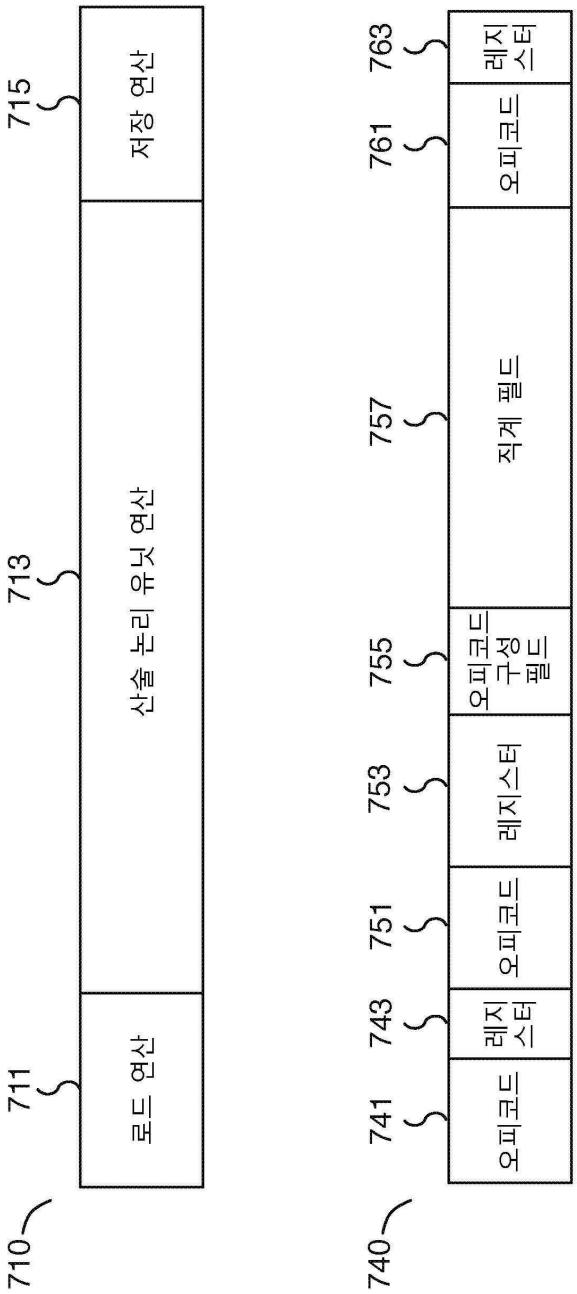
도면6a


601
개류중인 벡터 엔진 검색 명령
예
추가 명령?
603
아니오
609
다음 명령을 대기
605
디코드 명령
607
명령 실행
명령 완료

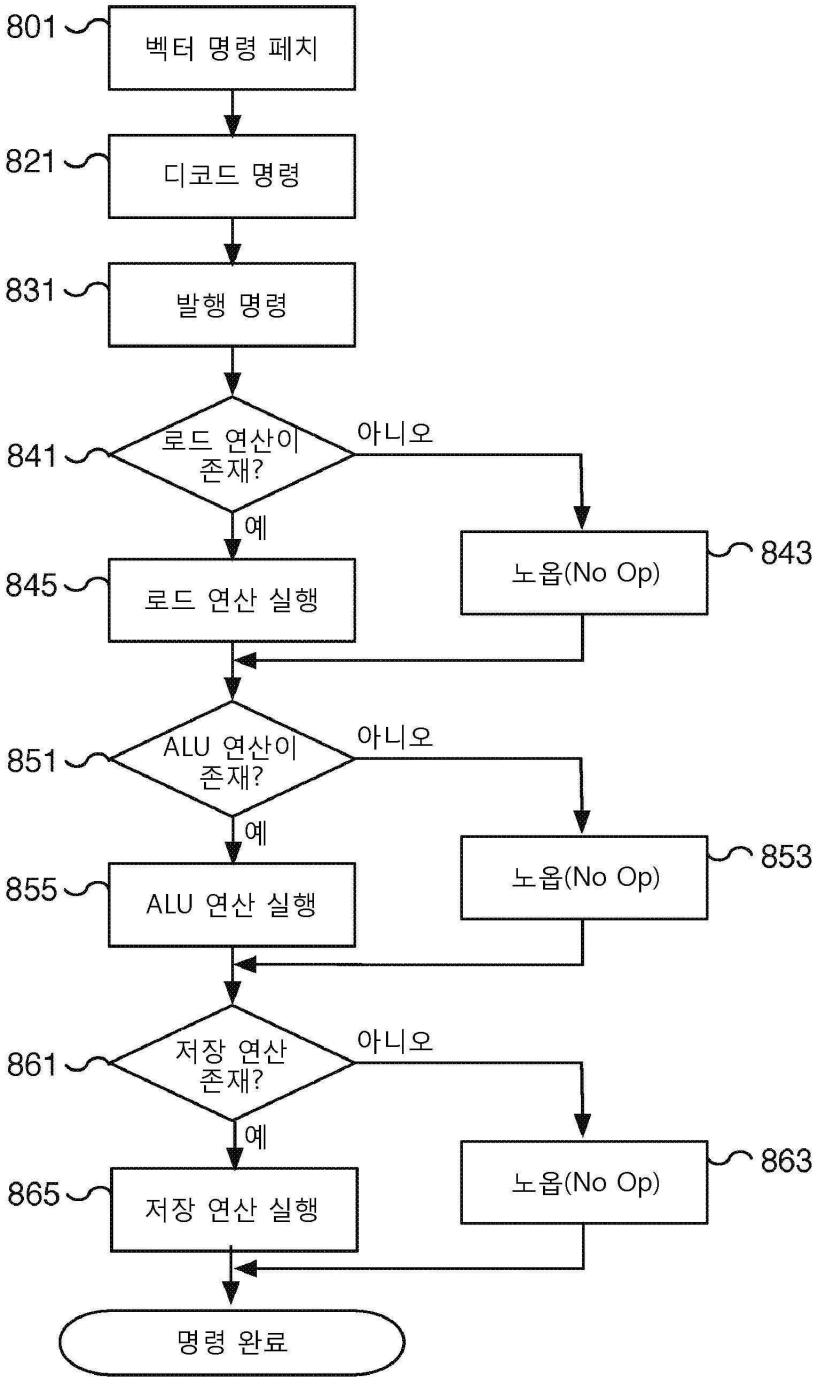
도면6b


651
디코드 및 로드 연산
653
입력 데이터 수신
655
레지스터에 벡터 데이터 로드
657
추가 데이터 필요?
예
아니오
659
추가 데이터 로드
661
ALU 연산 수행
663
결과 기록

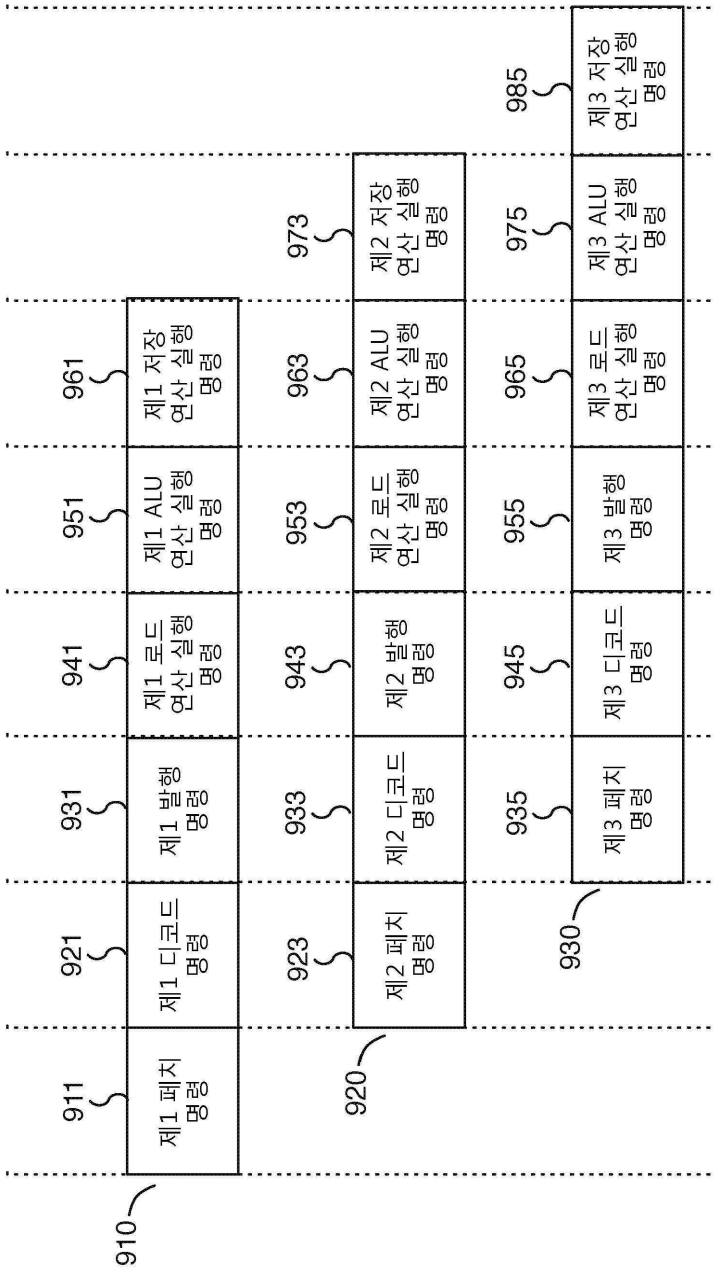
도면7


710
711
로드 연산
713
산술 논리 유닛 연산
715
저장 연산
740
741
오피코드
743
레지스터
751
오피코드
753
레지스터
755
오피코드 구성 필드
757
직계 필드
761
오피코드
763
레지스터

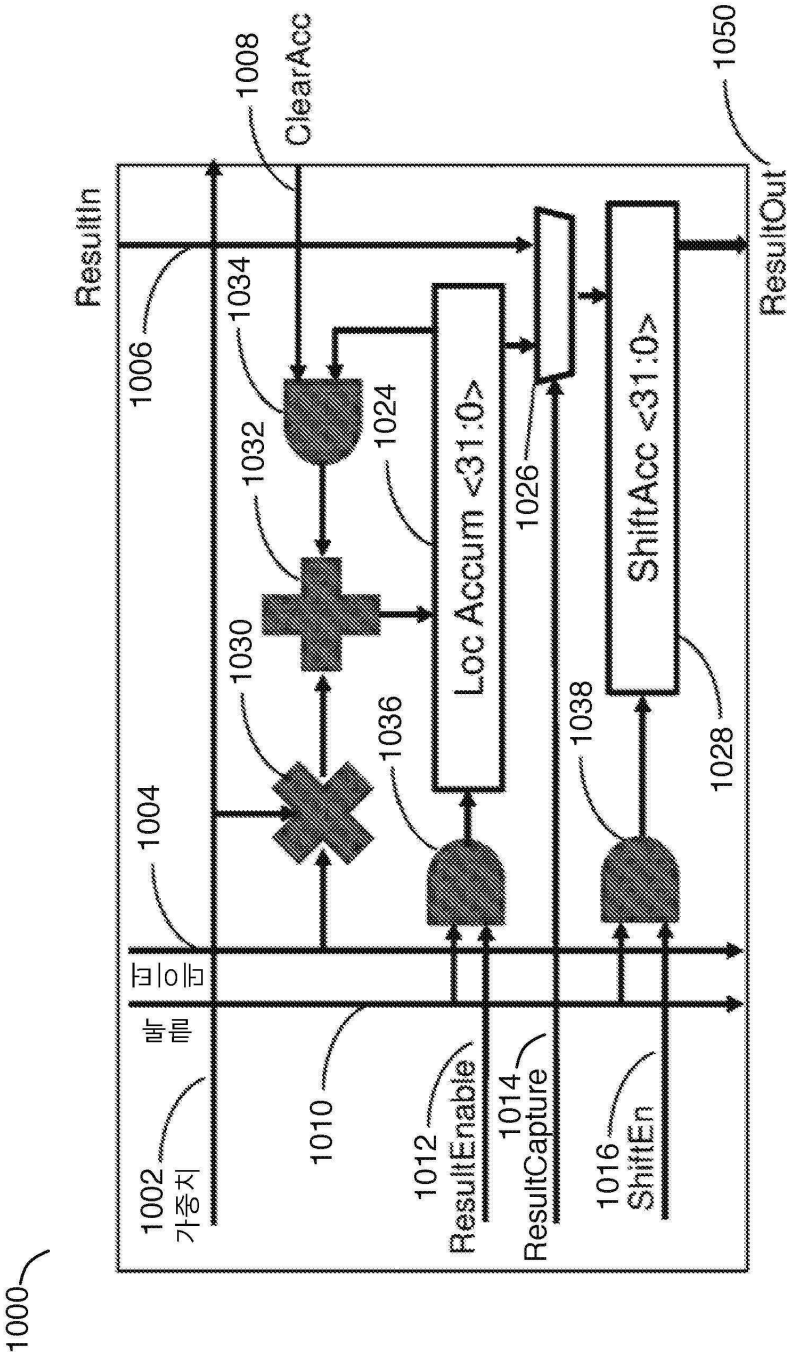
도면8


801
벡터 명령 페치
821
디코드 명령
831
발행 명령
841
로드 연산이 존재?
아니오
예
843
노옵(No Op)
845
로드 연산 실행
851
ALU 연산이 존재?
아니오
예
853
노옵(No Op)
855
ALU 연산 실행
861
저장 연산 존재?
아니오
예
863
노옵(No Op)
865
저장 연산 실행
명령 완료

도면9


910
911
제1 페치 명령
921
제1 디코드 명령
931
제1 발행 명령
941
제1 로드 연산 실행 명령
951
제1 ALU 연산 실행 명령
961
제1 저장 연산 실행 명령
920
923
제2 페치 명령
933
제2 디코드 명령
943
제2 발행 명령
953
제2 로드 연산 실행 명령
963
제2 ALU 연산 실행 명령
973
제2 저장 연산 실행 명령
930
935
제3 페치 명령
945
제3 디코드 명령
955
제3 발행 명령
965
제3 로드 연산 실행 명령
975
제3 ALU 연산 실행 명령
985
제3 저장 연산 실행 명령

도면10


1000
ResultIn
1006
1008
ClearAcc
1034
1032
1024
1030
1036
Loc Accum <31:0>
1026
1038
ShiftAcc <31:0>
1028
1050
ResultOut
1004
데이터
1002
가중치
1010
1012
ResultEnable
1014
ResultCapture
1016
ShiftEn