(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2020년05월04일
(11) 등록번호 10-2106360
(24) 등록일자 2020년04월24일

(51) 국제특허분류(Int. Cl.)
G06F 13/00 (2006.01) *G06F 15/00* (2006.01)
G06T 1/20 (2018.01) *G06T 1/60* (2006.01)
(52) CPC특허분류
G06F 13/00 (2018.05)
G06F 15/00 (2013.01)
(21) 출원번호 10-2015-7008715
(22) 출원일자(국제) 2013년08월09일
심사청구일자 2018년08월08일
(85) 번역문제출일자 2015년04월03일
(65) 공개번호 10-2015-0052282
(43) 공개일자 2015년05월13일
(86) 국제출원번호 PCT/US2013/054340
(87) 국제공개번호 WO 2014/039210
국제공개일자 2014년03월13일
(30) 우선권주장
13/602,958 2012년09월04일 미국(US)
(56) 선행기술조사문헌
EP00293700 A2
US20050257026 A1
US20070245123 A1

(73) 특허권자
미레플리카 테크놀로지, 엘엘씨
미국 텍사스 78132-3552 뉴 브라운펠스 애슈 주니퍼 223
존슨 윌리암 엠.
미국 텍사스 78746 오스틴 코퀴나 레인 606
(72) 발명자
존슨 윌리암 엠.
미국 텍사스 78746 오스틴 코퀴나 레인 606
(74) 대리인
박장원

전체 청구항 수 : 총 23 항 심사관 : 유주영

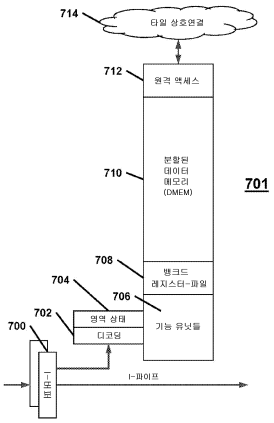
(54) 발명의 명칭 **2-차원의 상호관련된 데이터 세트들의 효율적인 고-처리량 프로세싱을 위한 프로세서, 시스템 및 방법**

(57) 요 약

단일 프로그램을 실행시키면서 동작들을 병렬로 수행하기 위해 프로세싱 데이터경로들을 체계화시키기 위한 시스템들, 프로세서들, 및 방법들이 개시된다. 각각의 데이터경로는 신규한 명령 시퀀싱 방법을 사용하여 명령들의 동일한 시퀀스를 실행한다. 각각의 데이터경로는 동일한 영역들로 분할된 데이터 메모리를 갖는 프로세서를 통해

(뒷면에 계속)

대 표 도 - 도7



구현된다. 마스터 프로세서는 명령들을 페치하여 이들을 데이터경로 프로세서들에 전달한다. 모든 프로세서들은 명령들이 병렬 데이터경로들에서 실행되도록 명령 파이프라인에 의해 직렬로 연결되어 있고, 여기서 각각의 데이터경로에서의 실행은 인접하는 데이터경로들에서의 실행으로부터 시간적으로 일 클럭 싸이클만큼 오프셋되어 있다. 본 시스템은 수평 차원 및 수직 차원 모두에서의 데이터의 완전 공유를 가능하게 하는 상호연결 네트워크를 포함하며, 이 경우 공통 사용에서 프로세싱 싸이클들을 추가함이 없이 임의의 데이터경로가 임의의 다른 데이터경로의 메모리에 결합되게 된다. 이러한 접근법은 하드와이어링 해법들의 처리량에 가까운 그러한 처리량을 갖는 프로그래밍가능 비주얼 컴퓨팅을 가능하게 한다.

(52) CPC특허분류
G06T 1/20 (2013.01)
G06T 1/60 (2013.01)

명 세 서

청구범위

청구항 1

데이터 프로세싱 시스템(data processing system)으로서,

제 1 방향을 따라 나란히 정렬된 적어도 두 개의 데이터 메모리들과, 각각의 데이터 메모리는 제 2 방향을 따라 동일한 크기의 적어도 두 개의 영역들로 분할되고, 여기서 영역들의 개수는 2를 거듭제곱한 값이고, 상기 제 2 방향은 상기 제 1 방향과 직교하며;

데이터 할당 회로(data allocation circuitry)와, 상기 데이터 할당 회로는,

2차원 어레이(two-dimensional array)의 요소(element)들을 나타내는 디지털 방식으로 코딩된 데이터(digitally-coded data)를 수신하는 것과,

상기 어레이의 제 1 차원을 따라 있는 미리결정된 개수의 인접하는 요소들의 제 1 시퀀스(sequence)를 나타내는 데이터를, 제 1 데이터 메모리의 제 1 영역 내의 연속적으로 인접하는 메모리 위치들에, 저장하는 것과, 그리고

상기 어레이의 상기 제 1 차원을 따라 있는 상기 미리결정된 개수의 인접하는 요소들의 상기 어레이의 제 2 차원을 따라 있는 인접하는 시퀀스들을 나타내는 데이터를, 상기 데이터 메모리들의 각각의 인접하는 영역들의 대응하는 연속적으로 인접하는 메모리 위치들에, 저장하는 것을 수행하고,

상기 제 1 데이터 메모리의 영역들 내에 저장된 데이터가 나타내는, 상기 제 1 시퀀스가 포함된, 상기 인접하는 시퀀스들의 개수는 상기 제 1 데이터 메모리 내의 영역들의 개수와 동일하고,

제 2 데이터 메모리의 영역들에 저장된 데이터는 동일한 개수의 인접하는 시퀀스들을 나타내고, 그리고

인접하는 데이터 메모리들의 대응하는 영역들에 저장된 데이터가 나타내는 시퀀스들은 상기 제 1 데이터 메모리 내의 영역들의 개수와 동일한 개수의 요소들의 수만큼 상기 어레이의 상기 제 2 차원을 따라 변위(displace)되어 있고;

인접하는 데이터 메모리들 내의 대응하는 어드레스(address)들에서의 데이터에 관해 동일한 명령들의 실행을 지시하도록 되어 있는 명령 라우팅 회로(instruction routing circuitry)와, 여기서 상기 동일한 명령의 실행은 상기 제 1 방향을 따라 있는 각각의 인접하는 데이터 메모리에 대해 일 클럭 싸이클(one clock cycle)만큼 오프셋(offset)되어 있으며;

상기 적어도 두 개의 데이터 메모리들 각각에 결합된 적어도 두 개의 명령 실행 유닛(instruction execution unit)들과, 여기서 상기 적어도 두 개의 명령 실행 유닛들은 각각의 상기 데이터 메모리에 저장된 데이터가 나타내는 요소들에 관한 동작들을 위한 명령들을 상기 명령 라우팅 회로로부터 수신하여 실행하도록 되어 있고; 그리고

상기 명령 라우팅 회로를 통해 상기 명령 실행 유닛들 모두에 결합된 마스터 프로세서 유닛(master processor unit)을 포함하여 구성되며,

상기 마스터 프로세서 유닛은 상기 데이터 메모리들에 저장된 데이터가 나타내는 요소들에 관한 동작들을 위한 명령들을 저장(store), 페치(fetch), 및 분배(distribute)하도록 되어 있는 것을 특징으로 하는 데이터 프로세싱 시스템.

청구항 2

제1항에 있어서,

상기 제 1 시퀀스 및 상기 인접하는 시퀀스들은 상기 어레이의 각각의 컬럼(column)들의 일부분들을 포함하는 것을 특징으로 하는 데이터 프로세싱 시스템.

청구항 3

삭제

청구항 4

삭제

청구항 5

삭제

청구항 6

제1항에 있어서,

상기 어레이의 요소들은 상기 어레이를 나타내기 위해 사용되는 포맷(format)을 따르는 다양한 타입을 가지며, 상이한 영역들 내의 대응하는 상대적 메모리 위치들에 저장된 데이터는 동일한 타입의 요소들을 나타내는 것을 특징으로 하는 데이터 프로세싱 시스템.

청구항 7

제6항에 있어서,

상기 어레이의 요소들은 이미지 픽셀 값(image pixel value)들을 포함하는 것을 특징으로 하는 데이터 프로세싱 시스템.

청구항 8

제1항에 있어서,

상기 어레이의 요소들은 물리적 수량(physical quantity)의 값들을 포함하는 것을 특징으로 하는 데이터 프로세싱 시스템.

청구항 9

데이터 프로세서(data processor)로서,

동일한 크기의 적어도 두 개의 영역들로 분할된 데이터 메모리와, 여기서 영역들의 개수는 2를 거듭제곱한 값이고;

프로그램 명령들을 수신하기 위한 명령 버퍼(instruction buffer)와;

상기 데이터 메모리의 영역들에 저장된 데이터를 사용하여 상기 프로그램 명령들을 실행하도록 되어 있는 프로세서 로직(processor logic)과; 그리고

상기 데이터 프로세서와 추가의 상호연결된 데이터 프로세서들의 그룹의 임의의 데이터 프로세서 간의 데이터 액세스 요청(data access request)의 경로(route)를 특정하기 위해 수평 어드레스 성분(horizontal address component)을 사용하도록 되어 있는 원격 액세스 회로(remote access circuitry)를 포함하는 것을 특징으로 하는 데이터 프로세서.

청구항 10

제9항에 있어서,

상기 데이터 프로세서에 의해 실행되는 프로그램 명령들의 서브그룹(subgroup)의 식별자를 저장하기 위한 영역 상태 로직(region state logic)을 더 포함하는 것을 특징으로 하는 데이터 프로세서.

청구항 11

제10항에 있어서,

상기 영역 상태 로직은 또한, 상기 데이터 메모리의 어떤 영역들이 프로그램 명령들의 서브그룹의 실행시 사용

되는 데이터를 포함하고 있는지의 표시를 저장하도록 되어 있는 것을 특징으로 하는 데이터 프로세서.

청구항 12

삭제

청구항 13

제9항에 있어서,

상기 명령 버퍼로부터 페치된 명령을 디코딩(decoding)하고 동시에 상기 명령을 인접하는 데이터 프로세서의 명령 버퍼에 전달하도록 되어 있는 명령 라우팅 회로를 더 포함하는 것을 특징으로 하는 데이터 프로세서.

청구항 14

데이터 프로세서로서,

실행될 프로그램 명령들을 저장하기 위한 명령 메모리와;

상기 명령 메모리로부터 페치된 명령을 디코딩하고 동시에 상기 명령을 인접하는 데이터 프로세서의 명령 버퍼에 전달하도록 되어 있는 명령 라우팅 회로와; 그리고

상기 인접하는 데이터 프로세서 내의 분할된 데이터 메모리의 각각의 연속하는 영역들에 저장된 데이터를 사용하는 실행을 위해, 명령들의 서브그룹을 상기 인접하는 데이터 프로세서에 반복적으로 전달하도록 되어 있는 실행 제어 회로를 포함하는 것을 특징으로 하는 데이터 프로세서.

청구항 15

제10항 또는 제14항에 있어서,

프로그램 명령들의 상기 서브그룹은 상기 데이터 메모리와 상기 데이터 메모리 외부에 있는 메모리 위치 간에 데이터의 이동(movement)을 요구하는 명령으로 끝나는 태스크 구간을 포함하는 것을 특징으로 하는 데이터 프로세서.

청구항 16

제14항에 있어서,

상기 분할된 데이터 메모리의 영역들 중 하나의 영역에 기입(write)된 각각의 데이터 요소에 대해, 상기 데이터 요소의 상기 기입을 실행한 명령들의 상기 서브그룹의 각각의 표시자(indicator)를 저장하도록 되어 있는 제어 메모리를 더 포함하는 것을 특징으로 하는 데이터 프로세서.

청구항 17

데이터 프로세싱 시스템으로서,

적어도 두 개의 데이터 프로세서들과, 각각의 데이터 프로세서는 동일한 크기의 적어도 두 개의 영역들로 분할된 데이터 메모리와, 그리고 데이터 프로세서들 간의 데이터 액세스 요청들을 처리하기 위한 원격 액세스 로직(remote access logic)을 포함하고;

상기 적어도 두 개의 데이터 프로세서들 중 제 1 데이터 프로세서의 상기 데이터 메모리의 영역들 중 하나의 영역에 있는 데이터를 사용하는 실행을 위해 프로그램 명령을 상기 제 1 데이터 프로세서에 전달하도록 되어 있는 마스터 프로세서(master processor)와, 여기서 상기 제 1 데이터 프로세서는 상기 프로그램 명령을 디코딩하는 것과 상기 프로그램 명령을 상기 적어도 두 개의 데이터 프로세서들 중 제 2 데이터 프로세서에 전달하는 것을 모두 수행하도록 되어 있고;

상기 마스터 프로세서로부터 상기 제 1 데이터 프로세서로 그리고 상기 제 2 데이터 프로세서로의 명령들의 순차적 전달을 위해 상기 마스터 프로세서와 상기 데이터 프로세서들을 연결하는 명령 파이프라인(instruction pipeline)과; 그리고

상기 동일한 데이터 프로세서들 각각에 대응하는 멀티플렉서(multiplexer)를 포함하는 데이터 프로세서 상호연결 구조를 포함하여 구성되며,

각각의 멀티플렉서의 출력은 각각의 데이터 프로세서의 각각의 상기 원격 액세스 로직의 입력에 결합되며,

상기 데이터 프로세서들의 로컬 그룹(local group)의 각각의 상기 원격 액세스 로직으로부터의 출력들은 각각의 멀티플렉서에 대한 입력들을 형성하는 것을 특징으로 하는 데이터 프로세싱 시스템.

청구항 18

제17항에 있어서,

상기 로컬 그룹 내의 데이터 프로세서들의 개수는 상기 멀티플렉서들 각각에 의해 처리되는 입력들의 개수와 동일하고, 상기 멀티플렉서들 각각은 단일 출력을 갖는 것을 특징으로 하는 데이터 프로세싱 시스템.

청구항 19

제18항에 있어서,

상기 상호연결 구조는 또한, 데이터 프로세서들의 각각의 로컬 그룹에 대해 추가의 원격 액세스 멀티플렉서를 포함하고,

상기 원격 액세스 멀티플렉서는 상기 로컬 그룹 내의 데이터 프로세서들의 개수와 동일한 개수의 입력들과, 그리고 단일 출력을 가지며,

데이터 프로세서들의 상기 로컬 그룹의 각각의 상기 원격 액세스 로직으로부터의 출력들이 상기 원격 액세스 멀티플렉서의 입력들에 연결되는 것을 특징으로 하는 데이터 프로세싱 시스템.

청구항 20

제 1 데이터 메모리의 분할된 동일한 영역들에 2차원 어레이의 요소들을 나타내는 데이터를 저장하기 위한 방법으로서,

상기 어레이의 제 1 차원을 따라 있는 미리결정된 개수의 인접하는 요소들의 제 1 시퀀스를 나타내는 데이터를, 상기 제 1 데이터 메모리의 제 1 영역의 대응하는 연속적으로 인접하는 메모리 위치들에, 저장하는 단계와; 그리고

상기 미리결정된 개수의 요소들의 상기 어레이의 제 2 차원을 따라 있는 인접하는 시퀀스들을 나타내는 데이터를, 상기 제 1 데이터 메모리의 연속적으로 인접하는 영역들의 대응하는 연속적으로 인접하는 메모리 위치들에 저장하는 단계를 포함하여 구성되며,

상기 제 1 데이터 메모리의 영역들에 저장된 데이터가 나타내는, 상기 제 1 시퀀스가 포함된, 상기 인접하는 시퀀스들의 개수는 상기 제 1 데이터 메모리 내의 영역들의 개수와 동일한 것을 특징으로 하는 데이터를 저장하기 위한 방법.

청구항 21

제20항에 있어서,

상기 제 1 시퀀스 및 상기 인접하는 시퀀스들은 상기 어레이의 각각의 컬럼들의 일부분들을 포함하는 것을 특징으로 하는 데이터를 저장하기 위한 방법.

청구항 22

제21항에 있어서,

상기 제 1 시퀀스 및 상기 인접하는 시퀀스들은 상기 어레이의 각각의 컬럼들 전체를 포함하는 것을 특징으로 하는 데이터를 저장하기 위한 방법.

청구항 23

제20항에 있어서,

추가의 인접하는 시퀀스들을, 추가의 분할된 데이터 메모리들 내의 연속적으로 인접하는 영역들의 대응하는 연속적으로 인접하는 메모리 위치들에, 저장하는 단계를 더 포함하고,

상기 데이터 메모리들 모두는 상기 데이터 메모리들이 영역들로 분할되는 방향에 직교하는 방향을 따라 상기 데이터 메모리와 나란히 정렬되는 것을 특징으로 하는 데이터를 저장하기 위한 방법.

청구항 24

적어도 두 개의 데이터 프로세서들에 결합된 마스터 프로세서에 의해 프로그램을 실행시키는 방법으로서,

각각의 데이터 프로세서는 동일한 크기의 복수의 영역들로 분할된 데이터 메모리를 포함하고, 상기 방법은,

상기 마스터 프로세서 내의 명령 메모리로부터 제 1 명령을 페치하는 단계와;

상기 적어도 두 개의 데이터 프로세서들 중 제 1 데이터 프로세서의 데이터 메모리의 제 1 영역에서의 데이터를 사용하는 실행을 위해, 상기 제 1 명령을 상기 제 1 데이터 프로세서의 명령 버퍼에 전달하는 단계와;

상기 제 1 데이터 프로세서에 의한 상기 제 1 명령의 실행이 상기 제 1 데이터 프로세서의 데이터 메모리와 임의의 다른 데이터 프로세서의 데이터 메모리 간에 데이터의 이동을 요구하는지 여부를 결정하는 단계와;

만약 상기 제 1 명령의 실행이 상기 제 1 데이터 프로세서와 임의의 다른 데이터 프로세서 간에 데이터의 이동을 요구하지 않는다면, 전달된 명령이 상기 데이터 프로세서들 간에 데이터의 이동을 요구한다고 결정될 때까지 후속 명령들을 계속해서 페치하여 상기 명령 버퍼에 전달하는 단계와; 그리고

상기 전달된 명령의 실행이 상기 제 1 데이터 프로세서와 임의의 다른 데이터 프로세서 간에 데이터의 이동을 요구한다고 결정된 경우, 상기 제 1 데이터 프로세서의 데이터 메모리의 연속적으로 인접하는 영역에 있는 데이터를 사용하는 실행을 위해 상기 제 1 명령을 검색하고 상기 제 1 명령을 상기 제 1 데이터 프로세서의 상기 명령 버퍼에 전달하는 단계를 포함하는 것을 특징으로 하는 프로그램을 실행시키는 방법.

청구항 25

제24항에 있어서,

상기 제 1 명령을 검색하고 상기 제 1 명령을 전달하는 단계에 후속하여,

상기 제 1 데이터 프로세서 내의 데이터 메모리의 각각의 인접하는 영역에 대해, 각각의 영역에 저장된 데이터를 사용하는 실행을 위해 상기 제 1 명령으로부터 상기 데이터 프로세서들 간에 데이터의 이동을 요구하는 명령에 이르는 범위를 갖는 명령들의 시퀀스의 전달을 반복하는 단계를 더 포함하는 것을 특징으로 하는 프로그램을 실행시키는 방법.

청구항 26

인접하는 데이터 프로세서에 결합됨과 아울러 마스터 프로세서에 결합된 데이터 프로세서에 의해 프로그램을 실행시키는 방법으로서,

상기 데이터 프로세서는 명령 버퍼와, 그리고 적어도 두 개의 동일한 영역들로 분할된 데이터 메모리를 포함하고, 상기 방법은,

상기 명령 버퍼 내에 상기 마스터 프로세서로부터의 프로그램 명령을 수신하는 단계와;

상기 프로그램 명령을 디코딩하고 상기 명령을 상기 인접하는 데이터 프로세서의 명령 버퍼에 전달하는 단계와;

상기 데이터 메모리의 제 1 영역에 있는 데이터를 사용하여 상기 프로그램 명령을 실행하는 단계와;

상기 명령의 실행이 상기 데이터 프로세서의 데이터 메모리와 임의의 다른 데이터 프로세서의 데이터 메모리 간에 데이터의 이동을 요구하는지 여부를 결정하는 단계와; 그리고

만약 상기 명령의 실행이 상기 데이터 프로세서와 임의의 다른 데이터 프로세서 간에 데이터의 이동을 요구하지 않는다면, 실행된 명령이 상기 데이터 프로세서들 간에 데이터의 이동에 대한 액세스를 요구한다고 결정될 때까지 데이터 메모리의 동일 영역에 있는 데이터를 사용하여 상기 명령 버퍼 내에 수신된 후속 명령들을 계속해서 실행하는 단계를 포함하는 것을 특징으로 하는 프로그램을 실행시키는 방법.

청구항 27

제26항에 있어서,

실행된 명령이 상기 데이터 프로세서와 임의의 다른 데이터 프로세서 간에 데이터의 이동을 요구한다고 결정된 경우, 상기 데이터 메모리의 다음 인접하는 영역에 저장된 데이터에 수신된 다음 명령의 실행을 적용하는 단계와; 그리고

상기 데이터 메모리의 각각의 인접하는 영역에 대해, 상기 데이터 프로세서들 간에 데이터의 이동을 요구하는 명령들로 끝나는 명령들의 시퀀스의 실행을 반복하는 단계를 더 포함하며, 여기서 실행은 각각의 영역에 저장된 데이터를 사용하는 것을 특징으로 하는 프로그램을 실행시키는 방법.

발명의 설명

기 술 분 야

배 경 기 술

[0001] 많은 애플리케이션(application)들이 2-차원 데이터 세트들과 관련되어 있다. 비주얼 프로세싱 애플리케이션(visual processing application)들이 하나의 예이다. 본 명세서에서 사용되는 바와 같은 용어 "비주얼 프로세싱(visual processing)"은 이미지 및 비디오 픽처 요소들(image and video picture elements)("픽셀들(pixels)") 및 관련 데이터의 프로세싱의 일반적 분류(class)를 지칭한다. 이것은 이미지들을 향상시키는 것과 픽셀 포맷들을 변환시키는 것, 움직임 검출 및 추적, 그리고 정지-이미지 혹은 비디오 프레임들에서의 피처(feature)들 혹은 오브젝트(object)들을 식별하는 것과 같은 애플리케이션들을 포함한다. 2-차원 세트들로 표현될 수 있는 데이터를 수반하는 다른 애플리케이션들은 힘(forces) 또는 전자기장(electromagnetic fields)과 같은 물리적 수량(physical quantities)의 모델링(modeling)을 포함한다. 3-차원 데이터 세트들은 일부 경우에 있어, 예를 들어, 2-차원 평면으로 사영(projection) 혹은 변환(transformation)을 행함으로써, 2-차원으로 나타내어질 수 있거나, 혹은 복수의 인접하는 2-차원 데이터 세트들로서 나타내어 질 수 있다. 종래-기술의 해법들은 두 가지 형태들, 즉 (1) 프로세싱 코어(processing core)를 사용하는 프로그래밍가능 해법들, 혹은 (2) 하드웨어 로직 게이트(hardware logic gate)들을 사용하는 하드와이어링 해법들(hardwired solutions)(혹은 하드웨어-구성가능 해법들(hardware-configurable solutions)) 중 하나의 형태를 취했다. 이러한 옵션들 간의 차이점은, 프로그래밍가능 해법들은 융통성(flexible)이 있고, 다양한 애플리케이션들에 맞게 용이하게 개선 및 조정될 수 있지만 하드와이어링 해법들에 비해 매우 낮은 성능을 갖고, 반면 하드와이어링 해법들은 성능 요건을 쉽게 충족시킬 수 있지만 새로운 요건을 충족시키기 위한 설계, 유지 및 개선이 매우 어렵다.

[0002] 대응하는 단점 없이 프로그래밍가능 프로세싱 및 하드와이어링 프로세싱의 장점들을 결합시킨 해법들을 갖는 것이 바람직하다. 예로서 비주얼 프로세싱 애플리케이션들의 경우를 고려한다. 본 명세서에서 사용되는 바와 같은 용어 "비주얼 컴퓨팅(visual computing)"은 범용 프로세서(general-purpose processor)를 사용하는 것과는 대조적으로, 비주얼 프로세싱을 특정적으로 목표로 하는 컴퓨팅 아키텍처에 근거하는 프로그래밍가능 접근법을 지칭한다. 비주얼-컴퓨팅 해법들의 많은 예들이 존재하지만, 이들 모두는 능력에 있어 근본적인 한계를 갖는다. 이러한 한계는 최신식의 센서 및 디스플레이 해상도를 갖는 경우, 아울러 초당 프레임들로 표현되는 프레임 속도가 증가하는 경우 특히 심해지게 된다.

[0003] 이러한 한계의 성질을 이해하기 위해, 먼저 필요한 것은 비주얼 프로세싱의 전형적인 요건들을 이해하는 것이다. 도 1은 디지털 카메라에서의 정지 이미지(still image)를 프로세싱하기 위한 일부 전형적인 스테이지(stage)들을 예시한다. 센서(100)로부터의 입력은 베이어 모자이크(Bayer mosaic)에서의 적색, 녹색, 및 청색 픽셀 값들 혹은 데이터 요소들을 포함한다(이러한 포맷은 눈(eye)의 녹색 정보에 대한 민감도 때문에 적색 및 청색보다 두 배 많은 녹색 정보를 포함함). 프로세싱은 품질 증진 및 포맷 변환을 수행하고, 이것은 JPEG과 같은 표준 이미지 포맷들을 만드는데 사용될 수 있는 YUV 데이터(108)를 생성한다(YUV는 루미넌스(luminance) 및 크로미넌스(chrominance) 정보를 갖는 3개의 픽셀들임). 본 예에서는 베이어 패턴이 제시되고 있지만, 전형적으로는 판매자별로 특정된 다수의 비-표준 독점 포맷들이 존재한다. 이것은 다양한 포맷들이 공통 해법에 의해 프로세싱될 수 있도록 하기 위한 프로그래밍가능 해법들에 대한 하나의 동기부여가 된다.

[0004] 픽셀 프로세싱은 전형적으로 이웃하는 픽셀 값들에 근거하여 임의의 주어진 위치에서 픽셀 값을 생성한다. 예를 들어, 노이즈 감소 스테이지(104)는 임의의 주어진 픽셀의 값을 그 로컬 이웃에서의 동일 포맷의 픽셀들의 값들과 비교하는 것에 근거한다. 만약 해당 값이, 이웃하는 픽셀 값들을 평균화함으로써 예측된 값의 어떤 임계치 위에 있거나 혹은 아래에 있다면, 이것은 렌즈의 불순물 혹은 센서 결함과 같은, 노이즈로 인한 것이라고 고려

된다. 이러한 경우에, 픽셀 값은 예측된 값으로 교체된다. 화이트 발란스 스테이지(white balance stage)(106)와 같은 다른 프로세싱 스테이지들은 전형적으로 이웃하는 픽셀 값들을 고려하는 동일 접근법을 사용한다. 블랙 레벨 조정(black level adjustment)(102)은 예외(exception)인데, 왜냐하면 이것은 순수하게 어두운 입력(purely dark input)에 대한 제로(zero)로부터 픽셀 값들에서의 드리프트(drift)를 보상하기 위해 각각의 픽셀 값에서 알려진 오프셋(known offset)들을 단순히 공제하기 때문이다.

[0005] 이러한 접근법은 도 2에 제시된 것과 같은 프로세싱 스테이지들 간의 입력 및 출력 관계들이 일어나게 한다. 본 예는 출력 픽셀이 픽셀들의 이웃하는 영역에 의존한다고 가정하고 있으며, 이에 따라 중앙 픽셀(central pixel)은 좌측 및 우측에 대한 픽셀들의 두 개의 컬럼(column)들 및 상부 및 하부의 픽셀들의 두 개의 로우(row)들에 의존하는바, 총 입력 영역은 5개 픽셀들의 폭과 5개의 픽셀들의 높이를 갖는 픽셀들의 블록(5x5)이고, 그 출력은 이 블록 내의 중앙 픽셀에 대응한다. 본 예에서, 도 1에 제시된 프로세싱 스테이지들이 순차적으로 번호가 부여되어 있다고 가정하면, 프로세싱 스테이지 N으로의 9x9 입력 영역(200)은 결과적으로 5x5 출력 영역(202)이 생성되게 하고, 이것은 프로세싱 스테이지 N+1에 대한 입력이 되며, 이러한 5x5 영역은 이후 스테이지 N+1의 출력으로서 단일 픽셀(204)을 발생시킨다.

[0006] 입력으로서 요구된 것이지만 임의의 대응하는 출력을 갖지 않는 픽셀들의 영역들은 에이프런(apron)들로서 지칭된다. 에이프런 요건들(apron requirements)은 프로세싱 스테이지의 결과 영역의 크기를 입력 영역보다 더 작아지게 하고, 이러한 감소가 크면 클수록 프로세싱은 더 복잡하게 되고 이에 따라 품질은 더 높아지게 된다. 더 높은 품질은 더 많은 수의 프로세싱 스테이지들을 요구하고 아울러 출력을 생성하기 위해 각각의 스테이지에서 사용되는 에이프런 픽셀들의 수를 더 많이 요구한다.

[0007] 전형적인 종래-기술의 비주얼-프로세싱 하드웨어의 동작이 도 3에서 예시되는바, 여기서 프로세싱 스테이지들은 도 2에서의 프로세싱 스테이지들에 대응한다. 메모리 버퍼(300)에 의해 스테이지 N에 대한 입력이 제공된다. 이러한 버퍼는 에이프런 요건들을 충족시키기에 충분한 콘텍스트(context)를 보유하며, 그 입력에는 한번에 픽셀들의 하나의 세트가 제공되는바, 본 예에서 픽셀들의 세트는 4개의 픽셀들(302)과 같은 그러한 4개의 픽셀들의 세트이다. 버퍼를 채우는 픽셀들의 세트의 입력이 버퍼의 중앙에 있는 픽셀들에 대해 에이프런 요건들을 충족시키기 때문에, 입력 픽셀들의 각각의 세트는 출력 픽셀들의 하나의 세트를 발생시키기에 충분하다. 예를 들어, 픽셀 세트(302)의 입력은, 각각의 방향에서 4개의 픽셀들의 5개의 세트들을 갖는 5x5 영역에서 그 중앙에 있는 픽셀 세트(304)에 대한 에이프런 요건들을 충족시키기 위해 버퍼를 채운다. 스캐닝 하드웨어(scanning hardware)(306)는 이러한 5x5 영역 내의 픽셀들에 액세스하고 스테이지 N에 대한 동작 하드웨어(operation hardware)(308)에 입력을 제공한다. 스캐닝 하드웨어가 요구되는 이유는 각각의 입력 세트가 이미지 내의 상이한 위치에 있고 스테이지 N에 대해 요구된 픽셀들의 버퍼(300) 내의 상대적 위치들이 각각의 입력 세트에 따라 변하기 때문이다. 이에 대응하여, 스테이지 N의 출력은 병합 하드웨어(merging hardware)(310)에 의해 프로세싱되고, 여기서 병합 하드웨어(310)는 복수의 입력 픽셀 세트들에 대한 스테이지 N의 출력을 버퍼(312)에 기입하는바, 이것은 버퍼(312)에 기입되는 결과들을 이전에-스캐닝된 픽셀 세트들에 관한 동작들로부터 보존(preserve)하는 방식으로 행해진다. 버퍼(312)는 버퍼(300) 내의 픽셀들에 대한 버퍼(312) 내의 픽셀들의 상대적 위치들을 명확히 하기 위해 오프셋되어 보여진다. 픽셀들(302)에 의해 인에이블(enable)된 출력은 버퍼(312) 내의 픽셀들(314)을 업데이트하고, 이것은 버퍼(312) 내의 픽셀들(316)의 영역에 대한 스테이지 N+1로의 입력을 인에이블시킨다.

[0008] 도 3에서 보여지는 하드웨어의 구성은 소프트웨어 프로그램에 의해 복제될 수 없는데, 왜냐하면 하드웨어 프로세싱 스테이지들 모두가 (스캔 라인을 따라 상이한 위치들에 있는 픽셀들에 관해) 동시에 동작하기 때문인바, 반면 소프트웨어 프로그램들은 한번에 하나의 명령을 실행하고 동시 동작들을 제어할 수 없기 때문이다. 각각의 하드웨어 스테이지는 하드와이어링되어 있거나 혹은 제한된 하드와이어링 구성들을 갖고, 이에 따라 다른 스테이지들과 동시에 동작하게 된다. 이러한 스테이지들의 처리량(throughput)은 전형적으로 매 프로세싱 싸이클마다 픽셀들의 하나의 세트이고, 따라서 예컨대, 450 메가-헤르츠(mega-Hertz)에서 동작하는 하드웨어는 일 싸이클당 450 메가-픽셀(mega-pixels)의 처리량을 제공할 수 있고, 이것은 최신식 센서들에 대해 요구되는 것인데, 이들은 초당 15 프레임의 속도에서 프레임당 30 메가-픽셀의 입력을 제공한다. 그러나, 스테이지들은 하드와이어링되어 있기 때문에 이러한 해법은 융통성이 없다.

[0009] 프로그래밍가능 해법들은 이러한 융통성이 없는 문제를 극복한다. 그러나, 프로그램 실행은 사실상 직렬로 일어나기 때문에, 프로그램들은 입력 이후 단지 일정 횟수의 순차적 프로세싱 단계들 혹은 싸이클들을 거쳐 출력들의 세트를 발생시킨다. 싸이클들의 개수는 프로세싱 스테이지들의 개수 및 각각의 스테이지에서 수행된 동작들의 복잡도에 의해 결정되며, 전형적으로는 대략 1000개 내지 2000개 싸이클들이다. 따라서, 450 메가-헤르츠에

서 동작하는 프로그래밍가능 해법은 하드웨어의 처리량보다 훨씬 낮은, 기껏해야 초당 0.45 메가-픽셀의 처리량을 제공한다.

[0010] 프로그래밍가능 비주얼 프로세싱의 처리량을 향상시키기 위해, 프로그램은 각각의 프로세싱 스테이지에서 많은 수의 픽셀들을 프로세싱 및 출력해야 한다. 예를 들어, 1000개의 싸이클들의 실행을 요구하는 프로그램은, 만약 이 프로그램이 입력 픽셀들의 각각의 세트에 대해서 출력의 픽셀들의 1000개의 세트들을 생성한다면, 하드웨어 프로세싱의 처리량에 필적할 수 있다. 그러나, 에이프런 요건들 때문에, 이러한 출력을 생성하기 위해 요구되는 입력은, 도 4에서 예시되는 가상의 이상적 경우를 제외하면, 입력들의 1000개의 세트들보다 훨씬 더 많다. 이러한 개념적 예에서, 다수의 프로세싱 데이터경로들(402)이 존재하며, 프로세싱 데이터경로들(402)의 수는 메모리(400) 내의 데이터에 관해 동작하는 전체 이미지의 픽셀들에서의 폭과 동일하다(메모리(400)도 또한 이러한 폭을 가짐). 각각의 데이터경로는 수평 방향에서 에이프런 액세스를 위해 충분한 이러한 메모리의 수평 영역에 액세스할 수 있으며(예를 들어, 5x5 영역에 대해 우측 및 좌측에 대한 2개의 픽셀들), 다수의 라인들에서 버퍼들의 깊이가 수직 방향에서 에이프런 액세스를 위해 충분한 그러한 버퍼들 내에 데이터가 체계화된다(예를 들어, 5x5 영역에 대해 5개 라인들의 깊이).

[0011] 입력이 이러한 이상적 해법에 전체 스캔-라인으로 한번에 제공되고, 가장 오래된 스캔-라인은 폐기된다. 이러한 라인들은 메모리(400) 내의 수평 라인들(404)에 의해 나타내어지고, 본 예에서는 베이어 포맷에서의 대응하는 픽셀 컬러들로 라벨링(labeling)되어 있다. 유사한 포맷(컬러)의 픽셀들은 개별적 버퍼들 내에 있어야만 하는데, 왜냐하면 프로그램은 모든 픽셀들에 관해 동일한 동작들을 동시에 수행하고 이들은 올바른 결과를 위해 동일한 포맷을 가져야만 하기 때문이다. 프로그램이 그 요구된 횟수의 싸이클들을 취하면서 실행되어 데이터경로들의 수에 비례하는 다수의 픽셀들을 생성하는바, 이것은 하드웨어의 처리량에 필적하기 위해 대략 수천 번이어야 한다. 이러한 예가 단지 예시적인 것임을 이해해야 한다. 이러한 많은 데이터경로들을 동시에 동작시키는 것은 물리적으로 불가능하고 어느 경우에서건 엄청난 비용을 필요로 한다.

[0012] 물리적 한계는 전형적인 구현에서 데이터경로들의 수를 제한하는바, 이에 따른 결과적인 전형적 체계가 도 5에서 보여진다. 도 4의 개념적 메모리 및 데이터경로 뱅크(datapath bank)는, 데이터경로 뱅크들(510-516)로 분할(partition)되고 아울러 데이터경로들의 수에 대응하는 메모리 세그먼트(memory segment)들(500-506)로 분할되는바, 이는 전형적으로 32개 혹은 64개로 적절하게 구현될 수 있다. 그러나, 입력으로부터 출력까지 프로세싱의 전체 범위에 걸쳐 이러한 데이터경로 뱅크들을 동작시키는 것은 실행의 유효성(effectiveness)에서 허용될 수 없는 손실을 일으키는바, 이것은 "x"로 표시된 음영화된 회색 영역들로 나타나 있고, 이는 각각의 프로세싱 스테이지에서의 에이프런 요건들로 인해 발생되는 유효 출력의 손실을 나타낸다. 예를 들어, 만약 10개의 프로세싱 스테이지들이 존재하고 그 각각이 각각의 스테이지에 대한 5x5 영역의 입력을 요구한다면, 데이터경로 뱅크들 각각은 각각의 스테이지에서 수평 방향으로 유효 콘텍스트의 4개의 픽셀들을 손실하게 된다. 이것은 마지막 스테이지에서의 출력이 입력보다 40개의 픽셀들만큼 더 좁은 폭을 갖게 하는데, 이는 64개의 데이터경로들의 경우, 출력이 단지 24개 픽셀들의 폭만을 갖게 하는바, 이것의 의미는 병렬 데이터경로들의 단지 38%(24/64)만이 유효함을 의미한다. 이러한 것을 피하기 위해, 데이터경로 뱅크들(510-516) 간에 메모리들(500-506) 내의 데이터를 공유하는 것이 요구된다. 그러나, 이것은 가능하지않는데, 왜냐하면 데이터경로들이 동일한 동작들을 동시에 수행함이 보장될 수 없기 때문이며, 따라서 데이터경로에 로컬(local)이 아닌 다른 메모리로부터의 데이터경로에 의해 액세스될 때 데이터가 유효함을 보장하는 것이 가능하지 않다.

[0013] 종래 기술은 모든 스테이지들을 직렬로 처리하는 것이 아니라 도 6에서 예시되는 바와 같이 단일 프로세싱 스테이지만을 한번에 수행함으로써 그 분할된 데이터경로들에서의 출력 데이터 감소의 앞서 언급된 문제에 대처한다. 임의의 주어진 스테이지 N 이후에, 스테이지 N의 출력은 시스템 데이터-이동 동작에서 글로벌 공유 메모리(global shared memory)(600)에 기입된다. 그 후에, 데이터는 시스템 데이터-이동 동작에서 데이터경로들에 로컬인 메모리들(도 5에서의 500-506)에 다시 판독된다. 이러한 다시-판독되는 동작은 데이터 뱅크들에 판독되는 스캔 라인의 일부분들을 오버랩핑(overlapping)시킴으로써 내부 데이터 뱅크들에 대한 데이터 손실을 최소화시키기 위해 사용될 수 있다. 하나의 데이터 뱅크의 끝에 가까이 있는 데이터 요소들을 인접하는 뱅크의 시작에서 반복하는 것은 각각의 데이터 뱅크에 대해 필요한 에이프런을 제공하고 경계(boundary)에서의 데이터 손실을 제거한다. 프로세싱 스테이지당 데이터의 손실은 단지 스캔 라인의 바깥쪽 끝들에 의해서만 일어나는 것이다(예를 들어, 4개의 픽셀들). 이러한 손실은 모든 프로세싱 스테이지들이 실행되는 경우보다 데이터경로 폭에 있어 훨씬 더 적은 비율을 차지하는바, 이에 따라 병렬 실행의 유효성은 60/64=94%가 되게 된다. 그러나, 스테이지 N의 출력들 및 스테이지 N+1의 입력들을 카피(copy)하기 위해 추가적인 싸이클들이 요구되고, 이러한 싸이클들은 유효 실행 시간에 부가되며, 또한 처리량을 제한한다.

[0014] 방금 제시된 이와 같은 예들 모두에서, 프로그래밍가능 비주얼-컴퓨팅 해법에 대한 근본적인 처리량 한계가 존재한다. 이것은, 에이프런 요건들을 충족시키기 위해 데이터를 공유할 수 없으므로 인해 발생되는 병렬 동작들의 비유효성으로 인한 것이거나, 혹은 글로벌 메모리 내에서 공유 콘텍스트를 다시-형성하기 위해 요구된 추가적인 싸이클들로 인한 것이다.

[0015] 앞에서 논의된 것은, 데이터 세트의 하나의 요소에 관해 수행될 동작이 데이터 세트의 하나 이상의 다른 요소들에 대한 액세스를 요구하는 경우(즉, 데이터 세트 내의 요소들이, 적어도 임의의 주어진 동작에 관하여, 상호관련되어 있는 경우), 효율적인 소프트웨어 프로세싱의 어려움을 예시한다. 동작의 하드웨어 구현들은 요구된 입력 요소들을 스캐닝 프로세스를 통해 반복적으로 획득함으로써 이러한 상황을 효율적으로 처리할 수 있다. 2-차원 데이터 세트들 내의 상호관련된 요소들에 관한 동작들의 소프트웨어 구현에 대한 효율적인 접근법을 갖는 것이 바람직하다.

발명의 내용

해결하려는 과제

과제의 해결 수단

[0016] 앞에서 언급된 문제들은, 단일 프로그램을 실행시켜 임의의 많은 수의 동작들을 병렬로 수행하기 위해 프로세싱 데이터경로들을 체계화시키기 위한 시스템들, 프로세서들, 및 방법들에 의해 성공적으로 처리될 수 있다. 이러한 데이터경로들은 임의의 명령-세트 아키텍처에 근거할 수 있다. 각각의 데이터경로는 신규한 명령 시퀀싱 방법(instruction sequencing method)을 사용하여 명령들의 동일한 시퀀스를 실행하고, 데이터경로들은 그 개수에 상관없이 동시에 실행될 수 있다. 본 시스템은 수평 차원(horizontal dimension) 및 수직 차원(vertical dimension) 모두에서의 데이터의 완전 공유(full sharing)를 가능하게 하는 상호연결 네트워크(interconnection network)를 포함하며, 이 경우 공통 사용(common usage)에서 프로세싱 싸이클(processing cycle)들을 추가함이 없이 임의의 데이터경로가 임의의 다른 경로의 메모리에 결합되게 된다. 이러한 접근법은 하드와이어링 해법(hardwired solution)들의 처리량에 가까운 그러한 처리량을 갖는 프로그래밍가능 비주얼 컴퓨팅(programmable visual computing)을 가능하게 한다.

[0017] 본 명세서에서 설명되는 바와 같은 데이터 프로세싱 시스템(data processing system)의 실시예에서, 적어도 두 개의 데이터 메모리들이 제 1 방향을 따라 나란히 정렬되며, 각각의 데이터 메모리는 제 1 방향과 실질적으로 직교하는 제 2 방향을 따라 동일한 크기의 적어도 두 개의 영역들로 분할된다. 본 시스템은 또한, 2차원 어레이(two-dimensional array)의 요소(element)들을 나타내는 디지털 방식으로 코딩된 데이터(digitally-coded data)를 제 1 데이터 메모리의 제 1 영역 내의 연속적으로 인접하는 메모리 위치들에 저장하도록 구성된 데이터 할당 회로(data allocation circuitry)를 포함한다. 제 1 영역에 저장된 데이터는 어레이의 제 1 차원을 따라 있는 미리결정된 개수의 인접하는 요소들의 제 1 시퀀스(sequence)를 나타날 수 있다.

[0018] 데이터 할당 회로는 또한, 어레이의 제 2 차원을 따라 있는 제 1 시퀀스에 인접하는 시퀀스들을 나타내는 데이터를, 제 1 영역에 인접하는 데이터 메모리들의 각각의 영역들의 연속적으로 인접하는 메모리 위치들에 저장하도록 구성된다. 제 1 데이터 메모리의 영역들에 저장된 데이터가 나타내는 인접하는 시퀀스들의 개수가 제 1 데이터 메모리 내의 영역들의 개수와 동일하도록 데이터가 저장된다. 추가적으로, 제 2 데이터 메모리의 영역들에 저장된 데이터는 동일한 개수의 인접하는 시퀀스들을 나타내고, 그리고 인접하는 데이터 메모리들의 대응하는 영역들에 저장된 데이터가 나타내는 시퀀스들은 제 1 데이터 메모리 내의 영역들의 개수와 동일한 개수의 요소들의 수만큼 어레이의 제 2 차원을 따라 변위(displace)되어 있다.

[0019] 일 실시예에서, 데이터 프로세싱 시스템은 또한, 인접하는 데이터 메모리들 내의 대응하는 어드레스(address)들에서의 데이터에 관해 동일한 명령들의 실행을 지시하도록 구성되어 있는 명령 라우팅 회로(instruction routing circuitry)를 포함하고, 여기서 동일한 명령의 실행은 제 1 방향을 따라 있는 각각의 인접하는 데이터 메모리에 대해 일 클럭 싸이클(one clock cycle)만큼 오프셋(offset)되어 있다. 시스템은 또한, 적어도 두 개의 데이터 메모리들 각각에 결합된 적어도 두 개의 명령 실행 유닛(instruction execution unit)들을 포함할 수 있다. 명령 실행 유닛들은 각각의 데이터 메모리에 저장된 데이터가 나타내는 요소들에 관한 동작들을 위한 명령들을 수신 및 실행하도록 구성되어 있다. 데이터 프로세싱 시스템은 또한, 명령 실행 유닛들 모두에 결합된 마스터 프로세서 유닛(master processor unit)을 포함할 수 있다. 마스터 프로세서 유닛은 데이터 메모리들에 저

장된 데이터가 나타내는 요소들에 관한 동작들을 위한 명령들을 저장(store), 페치(fetch), 및 분배(distribute)하도록 구성되어 있다.

[0020] 데이터 프로세싱 시스템의 또 다른 실시예는 적어도 두 개의 동일한 데이터 프로세서들을 포함하고, 여기서 각각의 데이터 프로세서는, 적어도 두 개의 동일한 영역들로 분할된 데이터 메모리와, 그리고 데이터 프로세서들 간의 데이터 액세스 요청(data access request)들을 처리하기 위한 원격 액세스 로직(remote access logic)을 포함한다. 본 시스템은 또한, 적어도 두 개의 동일한 데이터 프로세서들 중 제 1 데이터 프로세서에 프로그램 명령을 전달하도록 구성되어 있는 마스터 프로세서(master processor)를 포함하고, 여기서 명령은 제 1 데이터 프로세서의 데이터 메모리의 영역들 중 하나의 영역에 있는 데이터를 사용하는 실행을 위해 전달된다. 본 시스템은 또한, 마스터 프로세서와 데이터 프로세서들을 직렬로 연결하는 명령 파이프라인(instruction pipeline)과, 그리고 동일한 데이터 프로세서들 각각에 대응하는 멀티플렉서(multiplexer)를 갖는 데이터 프로세서 상호연결 구조를 포함한다. 각각의 멀티플렉서의 출력은 데이터 프로세서들 각각에 대한 원격 액세스 로직의 입력에 결합되며, 데이터 프로세서들의 로컬 그룹(local group) 각각의 원격 액세스 로직으로부터의 출력들은 각각의 멀티플렉서에 대한 입력들을 형성한다. 이러한 데이터 프로세싱 시스템의 추가적인 실시예에서, 로컬 그룹 내의 데이터 프로세서들의 개수는 멀티플렉서들 각각에 의해 처리되는 입력들의 개수와 동일하고, 멀티플렉서들 각각은 단일 출력을 갖는다. 이러한 시스템의 다른 실시예에서, 상호연결 구조는 또한, 데이터 프로세서들의 각각의 로컬 그룹에 대해 추가의 원격 액세스 멀티플렉서를 포함하고, 여기서 원격 액세스 멀티플렉서는 로컬 그룹 내의 데이터 프로세서들의 개수와 동일한 개수의 입력들과, 그리고 단일 출력을 갖는다. 로컬 그룹 내의 데이터 프로세서들 각각의 원격 액세스 로직으로부터의 출력들이 원격 액세스 멀티플렉서의 입력들에 연결된다.

[0021] "타일 프로세서(tile processor)"로서 지칭될 수 있는 본 명세서에서 설명되는 데이터 프로세서의 실시예는, 동일한 크기의 적어도 두 개의 영역들로 분할된 데이터 메모리와, 여기서 영역들의 개수는 2를 거듭제곱한 값이고; 프로그램 명령들을 수신하기 위한 명령 버퍼(instruction buffer)와; 데이터 메모리의 영역들에 저장된 데이터를 사용하여 프로그램 명령들을 실행하도록 구성되어 있는 프로세서 로직(processor logic)과; 그리고 데이터 프로세서와 추가의 상호연결된 데이터 프로세서들의 그룹의 임의의 데이터 프로세서 간의 데이터 액세스 요청의 경로(route)를 특정하기 위해 수평 어드레스 성분(horizontal address component)을 사용하도록 구성되어 있는 원격 액세스 회로(remote access circuitry)를 포함한다. 추가적인 실시예에서, 데이터 프로세서는 데이터 프로세서에 의해 실행되는 프로그램 명령들의 서브그룹(subgroup)의 식별자를 저장하기 위한 영역 상태 로직(region state logic)을 포함할 수 있다. 추가적으로, 영역 상태 로직은, 데이터 메모리의 어떤 영역들이 프로그램 명령들의 서브그룹의 실행시 사용되는 데이터를 포함하고 있는지의 표시를 저장할 수 있다. 프로그램 명령들의 이러한 서브그룹은 데이터 메모리와 데이터 메모리 외부에 있는 메모리 위치 간에 데이터의 이동(movement)을 요구하는 명령으로 끝날 수 있으며, 이것은 본 명세서에서 "태스크 구간(task interval)"으로 지칭될 수 있다. 또 다른 실시예에서, 데이터 프로세서는 명령 버퍼로부터 페치된 명령을 디코딩(decoding)하고 동시에 명령을 인접하는 데이터 프로세서의 명령 버퍼에 전달하도록 구성되어 있는 명령 라우팅 회로를 포함할 수 있다.

[0022] 본 명세서에서 "마스터 타일 프로세서(master tile processor)"로서 지칭될 수 있는 데이터 프로세서의 다른 실시예는, 실행될 프로그램 명령들을 저장하기 위한 명령 메모리와; 명령 메모리로부터 페치된 명령을 디코딩하고 동시에 명령을 인접하는 데이터 프로세서의 명령 버퍼에 전달하도록 구성되어 있는 명령 라우팅 회로와; 그리고 인접하는 데이터 프로세서 내의 분할된 데이터 메모리의 각각의 연속하는 영역들에 저장된 데이터를 사용하는 실행을 위해, 명령들의 서브그룹을 인접하는 데이터 프로세서에 반복적으로 전달하도록 구성되어 있는 실행 제어 회로를 포함한다. 추가적인 실시예에서, 이러한 데이터 프로세서는 또한, 분할된 데이터 메모리의 영역들 중 하나의 영역에 기입(write)된 각각의 데이터 요소에 대해, 명령들의 어떤 서브그룹이 데이터 요소의 기입을 실행했는지에 관한 표시자(indicator)를 저장하도록 구성되어 있는 제어 메모리를 포함할 수 있다.

[0023] 데이터 프로세서들 및 데이터 프로세싱 시스템들에 추가하여, 데이터 프로세싱 방법들이 본 명세서에서 고려된다. 본 방법들은 본 발명이 속하는 기술분야에서 통상의 기술을 가진 자들에게 알려진 기법들을 사용하여 프로그램 명령들에 의해 구현될 수 있다. 마스터 프로세서에 의해 프로그램을 실행시키는 방법의 실시예는, 마스터 프로세서 내의 명령 메모리로부터 제 1 명령을 페치하는 것과, 그리고 제 1 명령을 마스터 프로세서에 결합된 제 1 데이터 프로세서의 명령 버퍼에 전달하는 것을 포함하고, 여기서 제 1 데이터 프로세서는 마스터 프로세서에 결합된 적어도 두 개의 데이터 프로세서들 중 하나이고, 각각의 데이터 프로세서는 복수의 동일한 영역들로 분할된 데이터 메모리를 포함한다. 본 방법은 또한, 제 1 데이터 프로세서에 의한 제 1 명령의 실행이 제 1 데

이터 프로세서의 데이터 메모리와 임의의 다른 데이터 프로세서의 데이터 메모리 간에 데이터의 이동을 요구하는지 여부를 결정하는 것을 포함한다. 만약 제 1 명령의 실행이 제 1 데이터 프로세서와 임의의 다른 데이터 프로세서 간에 데이터의 이동을 요구하지 않는다면, 본 방법은 전달된 명령이 데이터 프로세서들 간에 데이터의 이동을 요구한다고 결정될 때까지 후속 명령들을 계속해서 페치하여 명령 버퍼에 전달하는 것을 포함한다.

[0024] 마스터 프로세서에 의해 프로그램을 실행시키는 방법의 추가적인 실시예는, 전달된 명령의 실행이 제 1 데이터 프로세서와 임의의 상이한 데이터 프로세서 간에 데이터의 이동을 요구한다고 결정된 경우, 제 1 데이터 프로세서의 데이터 메모리의 연속적으로 인접하는 영역에 있는 데이터를 사용하는 실행을 위해 제 1 명령을 검색하고 제 1 명령을 제 1 데이터 프로세서의 명령 버퍼에 전달하는 것을 포함한다. 제 1 데이터 프로세서 내의 데이터 메모리의 각각의 인접하는 영역에 대해, 본 방법은 각각의 영역에 저장된 데이터를 사용하는 실행을 위해 제 1 명령으로부터 데이터 프로세서들 간에 데이터의 이동을 요구하는 명령에 이르는 범위를 갖는 명령들의 시퀀스의 전달을 계속한다.

[0025] 데이터 프로세서에 의해 프로그램을 실행시키는 방법의 실시예는, 명령 버퍼 내에 프로그램 명령을 수신하는 것과, 프로그램 명령을 디코딩하고 동시에 명령을 인접하는 동일한 데이터 프로세서의 명령 버퍼에 전달하는 것과, 그리고 데이터 메모리의 제 1 영역에 있는 데이터를 사용하여 프로그램 명령을 실행하는 것을 포함한다. 본 방법은 또한, 명령의 실행이 데이터 프로세서의 데이터 메모리와 임의의 다른 데이터 프로세서의 데이터 메모리 간에 데이터의 이동을 요구하는지 여부를 결정하는 것을 포함한다. 만약 명령의 실행이 데이터 프로세서와 임의의 다른 데이터 프로세서 간에 데이터의 이동을 요구하지 않는다면, 본 방법은 실행된 명령이 데이터 프로세서들 간에 데이터의 이동을 요구한다고 결정될 때까지 데이터 메모리의 동일 영역에 있는 데이터를 사용하여 명령 버퍼 내에 수신된 후속 명령들을 계속해서 실행하는 것을 포함한다.

[0026] 데이터 프로세서에 의해 프로그램을 실행시키는 방법의 추가적인 실시예는, 실행된 명령이 데이터 프로세서와 임의의 다른 데이터 프로세서 간에 데이터의 이동을 요구한다고 결정된 경우, 데이터 메모리의 다음 인접하는 영역에 저장된 데이터에 수신된 다음 명령의 실행을 적용하는 것을 포함한다. 데이터 메모리의 각각의 인접하는 영역에 대해, 본 방법은 데이터 프로세서들 간에 데이터의 이동을 요구하는 명령들로 끝나는 명령들의 시퀀스를 실행시키는 것을 포함하며, 여기서 실행은 각각의 영역에 저장된 데이터를 사용한다.

[0027] 데이터 메모리의 분할된 영역들에 데이터를 저장하기 위한 방법이 또한 본 명세서에서 고려되며, 여기서 데이터는 2-차원 어레이의 요소들을 나타낸다. 본 방법의 실시예는, 어레이의 제 1 차원을 따라 있는 미리결정된 개수의 인접하는 요소들의 시퀀스를 나타내는 데이터를 데이터 메모리의 제 1 영역의 대응하는 연속적으로 인접하는 메모리 위치들에 저장하는 것을 포함한다. 본 방법은 또한, 어레이의 제 2 차원을 따라 있는 미리결정된 개수의 요소들의 인접하는 시퀀스들을 나타내는 데이터를 데이터 메모리의 연속적으로 인접하는 영역들의 대응하는 연속적으로 인접하는 메모리 위치들에 저장하는 것을 포함한다. 제 1 데이터 메모리의 영역들 내에 저장된 데이터가 나타내는, 제 1 시퀀스가 포함된, 인접하는 시퀀스들의 개수는 제 1 데이터 메모리 내의 영역들의 개수와 동일하다. 일 실시예에서, 제 1 시퀀스 및 인접하는 시퀀스들은 2-차원 어레이의 각각의 컬럼(column)들의 일부분들이다. 추가적인 실시예에서, 제 1 시퀀스 및 인접하는 시퀀스들은 어레이의 각각의 컬럼들 전체이다. 추가적인 실시예에서, 본 방법은 또한, 추가의 인접하는 시퀀스들을 추가의 분할된 데이터 메모리들 내의 연속적으로 인접하는 영역들의 대응하는 연속적으로 인접하는 메모리 위치들에 저장하는 것을 포함한다. 이러한 실시예에서, 데이터 메모리들은 모두 데이터 메모리들이 영역들로 분할되는 방향에 실질적으로 직교하는 방향을 따라 나란히 정렬된다.

[0028] 본 명세서에서 설명되는 시스템들, 프로세서들, 및 방법들은 프로그래밍가능 이미지 및 비젼 프로세싱(programmable image and vision processing)에 적용가능한바, 병렬로 동작하는 최대 4096개의 데이터경로들을 효율적으로 사용한다. 프로세서들은 바람직하게는 하이-레벨 언어(high-level language)로 기입된 순차적 프로그램을 실행한다. 데이터경로들은 애플리케이션 요건들에 따라 단일 데이터경로의 세분화(granularity)된 단위에서 할당될 수 있다. 일 실시예의 경우, 본 명세서에서 설명되는 기법들은 임의의 기존 명령 세트 및 C++ 컴파일러(compiler)에 맞게 조정될 수 있다. 데이터경로들은 이미지 혹은 비디오 프레임의 임의의 범위에 걸쳐 비주얼 데이터의 완전 공유를 구현할 수 있으며, 이 경우 제로-싸이클 레이턴시(zero-cycle latency) 및 완전 코히어런시(full coherency)를 지원하는 완전히 상호연결된 포인트-투-포인트 링크(point-to-point link)들과 유사한 효과를 갖게 된다. 계산(computation)에 있어서, 또한 모든 데이터경로들에 걸쳐 공유된, 글로벌 공유 데이터(global shared data), 룩업 데이블(lookup table)들, 및 히스토그램(histogram)들이 포함될 수 있다.

도면의 간단한 설명

[0029] 개시되는 다양한 실시예들에 관한 다음의 상세한 설명은 첨부되는 도면들을 참조한다.

도 1은 전형적인 정지-이미지 프로세싱 파이프라인을 보여준다.

도 2는 이미지 프로세싱을 위한 상대적인 입력 및 출력 콘텍스트를 보여준다.

도 3은 전형적인 하드웨어 프로세싱 파이프라인을 보여준다.

도 4는 이상적인 하지만 실행 불가능한 프로그래밍가능 해법을 보여준다.

도 5는 병렬 비주얼 프로세싱의 어려움을 예시한다.

도 6은 병렬 비주얼 프로세싱의 종래-기술의 접근법들을 보여준다.

도 7은 2-차원 어레이 프로세서 혹은 "타일 프로세서"의 선택된 컴포넌트들을 보여준다.

도 8은 타일 프로세서의 로컬 그룹을 보여준다.

도 9는 타일 상호연결 라우팅 유닛의 체계를 보여준다.

도 10은 라우팅 유닛들이 라우팅 계층들로 체계화된 것을 보여준다.

도 11은 타일 상호연결 라우팅 계층을 보여준다.

도 12는 타일 프로세서들 내의 데이터 메모리들의 분할 구성들을 보여준다.

도 13a는 픽셀 데이터의 스캔-라인을 4개의 영역들로 분할된 데이터 메모리들에 맵핑시킨 것을 보여준다.

도 13b는 픽셀 데이터의 2-차원 어레이를 도 13a의 데이터 메모리들에 맵핑시킨 것을 보여준다.

도 14는 픽셀 데이터의 스캔-라인을 8개의 영역들로 분할된 데이터 메모리들에 맵핑시킨 것을 보여준다.

도 15는 픽셀 데이터의 스캔-라인을 16개의 영역들로 분할된 데이터 메모리들에 맵핑시킨 것을 보여준다.

도 16a, 도 16b, 도 16c, 도 16d, 및 도 16e는 타일 상호연결 라우팅이 어떻게 결정되는지를 보여준다.

도 17a은 마스터 타일 프로세서의 체계를 보여준다.

도 17b는 본 명세서에서 설명되는 프로세서에 의해 실행되는 예시적인 태스크 구간을 보여준다.

도 17c 및 도 17d는 마스터 타일 프로세서에 의한 프로그램 실행 방법을 예시한다.

도 17e는 타일 프로세서에 의한 프로그램 실행 방법을 예시한다.

도 18a 및 도 18b는 명령 시퀀싱이 어떻게 레이턴시로 인한 지연들을 피하는지를 보여준다.

도 19a, 도 19b, 도 19c, 및 도 19d는 상호연결 라우팅들의 타이밍을 보여준다.

도 20은 의존성 그래프의 예를 보여준다.

도 21은 지연들을 피하는 의존성 결정의 타이밍을 보여준다.

발명을 실시하기 위한 구체적인 내용

[0030] 특정 시스템 컴포넌트들을 지칭하는 특정 용어들이 다음의 설명 및 특허청구범위에 걸쳐 사용된다. 본 발명의 기술분야에서 숙련된 자가 인식하는 바와 같이, 기업체들은 다른 명칭으로 컴포넌트를 지칭할 수 있다. 본 문서는 명칭에서는 상이하지만 그 기능에 있어서는 그렇지 않은 컴포넌트들을 구분하도록 의도되지 않았다. 다음의 논의에서 그리고 특허청구범위에서, 용어 "포함하는"는 개방형(open-ended fashion) 의미로 사용되는 것이며, 따라서 "...을 포함하는 하지만 이러한 것으로만 한정되는 것은 아닌"의 의미로 해석돼야 한다. 또한, 용어 "결합"은 간접적인 전기적 연결 혹은 직접적인 전기적 연결을 의미하도록 의도된 것이다. 따라서, 만약 제 1 디바이스가 제 2 디바이스에 결합된다고 하면, 이러한 연결은 직접적인 전기적 연결을 통한 것일 수 있거나 혹은 다른 디바이스들 및 연결들을 통한 간접적인 전기적 연결을 통한 것일 수 있다.

[0031] 다음의 논의는 본 명세서에서 설명되는 시스템들, 프로세서들, 및 방법들의 다양한 실시예들에 관한 것이다. 이러한 실시예들 중 하나 이상이 바람직할 수 있지만, 본 명세서에서 개시되는 실시예들은 특허청구범위를 포함하는 본 개시내용의 범위를 한정하는 것으로 해석되거나 혹은 사용돼서는 안 된다. 추가적으로, 본 발명의 기술분

야에서 숙련된 자는 다음의 설명이 광범위한 애플리케이션을 갖는다는 것을 이해할 것이며, 아울러 임의의 실시예의 설명이 단지 해당 실시예의 사례가 되도록 의도된 것일 뿐 특허청구범위를 포함하는 본 개시내용의 범위가 그 실시예로만 한정됨을 시사하도록 의도된 것이 아님을 이해할 것이다.

[0032] 도 7은 비주얼 타일에 대한 기본 프로세싱 요소의 선택된 컴포넌트들을 보여주는바, 여기서 용어 "타일(tile)"은 폭이 4개 혹은 8개 혹은 16개의 픽셀들이고 높이가 최대 256개의 픽셀들인, 프레임 내의 픽셀들의 직선적 영역(rectilinear region) 혹은 2-차원 어레이를 말한다. 각각의 프로세싱 요소 혹은 타일 프로세서는 픽셀들의 고유한 타일에 관해 동작하고, 이 경우 인접하는 타일들은 인접하는 타일 프로세서들로 맵핑된다. 이러한 맵핑은 기본 프로세서 체계에 관한 설명 다음에 아래에서 더 기술된다. 타일 프로세싱이 본 명세서에서는 주로 이미지 프로세싱 애플리케이션들에 대해 설명되지만, 개시되는 실시예들은 2-차원 데이터 세트들(특히, 2-차원의 상호관련된 데이터 세트들)의 데이터 프로세싱을 수반하는 임의의 애플리케이션들에 대해서도 적합한 것으로 고려된다는 것을 이해해야 한다.

[0033] 타일 프로세서(701)는 종래의 프로세서와 공통되는 많은 컴포넌트들을 가지고 있지만, 주목할 만한 예외사항은 명령 메모리가 없다는 점 그리고 명령 페치 로직이 없는 점이다. 도 7의 설명은 타일 프로세서(701)의 종래와 다른 컴포넌트들을 예시하도록 의도된 것이며, 아울러 복수의 타일 프로세서들(701)의 그룹화에 관한 도 8에서의 설명을 용이하게 하기 위해 형성된 것이다. 타일 프로세서(701)의 컴포넌트들의 배치가 컴포넌트들 간의 모든 상호연결들을 반영하는 것은 아닐 수 있다. 본 명세서에서 달리 기재되는 바와 같은 것을 제외하고, 타일 프로세서(701)는 본 발명의 기술분야에서 알려진 바와 같은 종래의 프로세서의 방식으로 상호연결되는 (예를 들어, 파워 서플라이들을 포함하는) 종래의 프로세서의 컴포넌트들을 포함한다.

[0034] 종래의 명령 메모리 및 명령 페치 로직을 사용하는 대신에, 타일 프로세서(701)는 아래에서 설명되는 마스터 타일 프로세서에 의해 페치되어 (모든 타일 프로세서들을 명령들의 동일한 시퀀스에 결합시키는 직렬 명령 버퍼 혹은 I-버퍼 레지스터들(700)로 구성되는) 명령 파이프라인을 사용하여 분배되는 명령들을 사용한다. I-버퍼 레지스터들은 더블-버퍼링(double-bufferung)되는바, 이에 따라 로컬 타일 프로세서에서의 프로세싱을 인터럽트(interrupt)하는 임의의 스톨(stall)은 단지 다음 명령을 제공하는 타일 프로세서로만 전파될 필요가 있게 된다. 복수의 스톨 싸이클들에 대해서, 스톨은 인접하는 프로세서에 의해 다음 프로세서로 전파되는 등 이렇게 일어난다. 명령 페치의 스타일은 스톨링(stalling)을 제어하기 위한 글로벌 신호의 사용을 피하는바(이것은 많은 수의 타일 프로세서들을 높은 주파수에서 동작시키는 것을 허용하지 않음), 이는 이러한 신호를 전파시킴에 있어서의 지연들 때문이다.

[0035] I-버퍼(700)에 수신된 명령들은 디코딩 로직(702)에 의해 디코딩되고, 그리고 오퍼랜드(operand)들을 페치하고 결과들을 분할된 데이터 메모리(DMEM)(710)에 기입하기 위해 로드(load)들 및 스토어(store)들을 사용하여 기능 유닛(functional unit)들(706)에 의해 뱅크드 레지스터-파일(banked register-file)(708) 내의 레지스터들의 콘텐츠들에 관한 동작들이 수행된다. 이러한 것들은 종래의 프로세서 특징들인바, 이러한 특징들은 본 발명의 기술분야에서 숙련된 자에게 친숙할 것이다. 타일 프로세서의 신규한 특징들은, 영역 상태 로직(region state logic)(704), 레지스터-파일 뱅킹 메커니즘(register-file banking mechanism)(708), 데이터-메모리 분할 메커니즘(data-memory partitioning mechanism)(710), 원격 액세스 로직(remote access logic)(712), 타일 상호연결 네트워크(tile interconnect network)(714), 및 이들의 조합들을 포함한다. 네트워크(714)가 별개의 하드웨어 블록으로서 제시되지 않았는데, 왜냐하면 이것은 모든 타일 프로세서들을 다른 타일 프로세서들 모두의 DMEM에 결합시키는 완전히 연결된 네트워크를 나타내기 때문이다. 이러한 컴포넌트들은 시스템 체계의 맥락에서 아래에서 상세히 설명된다.

[0036] 도 8은 4개의 타일 프로세서들의 로컬 그룹에 대한 타일 상호연결을 나타낸 도면으로, 4개의 타일 프로세서들 각각은 도 7에 제시된 바와 같이 체계화되어 있다. 각각의 타일 프로세서의 원격 액세스 로직(712)은 로컬 액세스 상호연결(local access interconnect)(800)에 결합되며, 로컬 액세스 상호연결(800)은 단일 싸이클에서 4개의 타일 프로세서들 중 임의의 타일 프로세서로부터의 요청들을 다른 타일 프로세서들 중 임의의 타일 프로세서로 라우팅시킨다. 이와 동시에 원격 액세스 로직(802)은 임의의 요청이 로컬 그룹 내의 프로세서로 디렉트(direct)되지 않는지 여부를 결정하고 임의의 이러한 요청을 비-로컬 타일 상호연결(non-local tile interconnect)(804)에 제공하는바, 비-로컬 타일 상호연결(804)은 이러한 비-로컬 요청(non-local request)을 궁극적인 목적지에 결합시킨다. 로컬 액세스 상호연결(800), 원격 액세스 로직(802) 및 비-로컬 타일 상호연결(804)은 도 7에 제시된 상호연결 네트워크(714)의 특정 레벨들이다.

[0037] 로컬 액세스 상호연결(800) 및 원격-액세스 로직(802)의 실시예들의 구조가 도 9에서 상세히 제시된다. 명료한

설명을 위해, 도 9에서는 상호연결된 타일 프로세서들 각각의 원격 액세스 로직(712)만이 제시된다. 로컬 액세스 상호연결은 멀티플레서(MUX)들(900-906)에 의해 구현되는바, 특히 각각의 MUX에 대해 도면번호 800으로 라벨링되어 있는 처음 4개의 입력들에 의해 구현된다. 도면에서 알 수 있는 바와 같이, 이것은 임의의 타일 프로세서로부터의 요청 혹은 응답을 그룹 내의 임의의 다른 타일 프로세서에 완전히 결합시킨다. 원격 액세스 상호연결(802)은 MUX(908)에 의해 구현되며(여기서 MUX(908)는 밖으로 나가는 비-로컬 요청 혹은 응답을 비-로컬 타일 상호연결에 결합시킴), 아울러 MUX들(900-906)에 대한 다섯 번째 입력에 의해 구현된다(여기서, MUX들(900-906)에 대한 다섯 번째 입력은 비-로컬 타일 상호연결로부터 안으로 들어오는 비-로컬 요청 혹은 응답을 로컬 그룹 내의 목적지 프로세서에 결합시킴). 멀티플렉서들(900-908)은 모두 함께 라우팅 유닛(routing unit)(910)을 구성하는바, 이것은 임의의 싸이클에서 4개의 로컬 요청들 및 1개의 비-로컬 요청을 라우팅시킬 수 있다. 도 8 및 도 9의 실시예에서, 도 8의 로컬 그룹 내에 있는 4개의 타일 프로세서들은 도 9에서의 원격 액세스 멀티플렉서(908)의 4개의 입력들에 대응한다.

[0038] 도 8 및 도 9의 실시예와 함께 도 10을 계속 참조하면, 라우팅 유닛들(910)은 라우팅 계층(100) 내에 계층적으로 결합된다. 라우팅 계층은 라우팅 유닛(910)의 동일한 인스턴스(instance)들을 5개 포함하며, 여기서 4개의 인스턴스들은 도 9에서 제시된 바와 같이 로컬 및 비-로컬 액세스들(요청들 혹은 응답들)을 라우팅시키며(이것은 라우팅 레벨로 지칭됨), 그리고 다섯 번째 인스턴스는 처음 4개의 인스턴스들로부터의 4개의 비-로컬 액세스들을 라우팅시킨다(이것은 다음 라우팅 레벨로 지칭됨). 이러한 다음 레벨은 또한, 제 1 레벨로부터의 액세스들을 로컬 요청과 유사하게 제 1 레벨에서의 라우팅 유닛들 중 다른 유닛에 결합시킬 수 있거나, 또는 액세스가 어떤 상위 라우팅 계층에 결합돼야한다고 결정할 수 있다. 다음 계층에서의 이러한 비-로컬 액세스들은 액세스 버퍼들(1000 및 1002) 내에 배치되는바, 버퍼(1000)는 밖으로 나가는 액세스들을 버퍼링하고, 버퍼(1002)는 안으로 들어오는 액세스들을 버퍼링한다. 레지스터들(1000 및 1002)은 도 7에서 제시된 I-버퍼 레지스터들(700)과 유사하게 더블-버퍼링되는바, 이에 따라 스톨들은 글로벌하게 전파될 필요가 있는 것이 아니라 단지 한 레벨 더 전파될 필요만이 있게 된다.

[0039] 도 10에 의해 예시되는 바와 같이, 라우팅 계층(1004)의 구성은, 라우팅 계층에 로컬인 16개의 액세스들, 그리고 라우팅 계층에 비-로컬인 1개의 액세스를 단일 싸이클에서 라우팅시킬 수 있다. 가능한 로컬 액세스들이 너무 많이 주어지는 경우 비-로컬 대역폭이 불충분할 것처럼 보여도, I-버퍼 레지스터들(700) 때문에 타일 프로세서들은 일 싸이클 만큼 오프셋된 대응하는 액세스 명령들을 실행하여 이들의 액세스들을 한번에 하나씩 라우팅 레벨에 제공한다는 것을 이해해야 한다. 따라서, 액세스 라우팅들에 대한 요구는 시간적으로 균등하게 분배된다.

[0040] 도 11은 도 8 내지 도 10의 실시예에서의 라우팅 유닛들(910)이 임의 쌍의 타일 프로세서들 간의 완전-결합된 액세스를 구현하기 위해 어떻게 구조화되는지를 예시한다. 로컬 상호연결(1100), 레벨 1 상호연결(1102), 및 레벨 1 액세스 버퍼(1104)의 결합은 라우팅 계층(1004)의 인스턴스들을 256개 포함한다. 이것은 로컬 라우팅을 위한 라우팅 유닛(910)의 인스턴스들을 1024개("1024x") 포함하고, 레벨 1 상호연결을 위한 라우팅 유닛(910)의 인스턴스들을 256개("256x") 포함하고, 그리고 레벨 1 상호연결로의 비-로컬 액세스들 그리고 레벨 1 상호연결로부터의 비-로컬 액세스들을 위한 액세스 버퍼들(1000 및 1002)의 인스턴스들을 256개 포함한다. 이에 따라, 레벨 2 상호연결(1106), 레벨 3 상호연결(1108), 및 레벨 3 액세스 버퍼(1110)의 결합은 라우팅 계층(1004)의 인스턴스들을 16개 포함하는바, 도면에서는 (버퍼(1110)를 위한) 액세스 버퍼들(1000 및 1002) 및 (상호연결들(1106 및 1108)을 위한) 라우팅 유닛(910)의 인스턴스들의 개수가 제시되어 있다. 레벨 4 상호연결(1112), 레벨 5 상호연결(1114), 및 레벨 5 액세스 버퍼(1116)를 결합시키는 상부 레벨은 라우팅 유닛(1004)의 단일 인스턴스를 포함한다. 레벨 5에서의 비-로컬 요청은 데이터를 시스템의 나머지에 결합시키기 위한 시스템 인터페이스(1120)로 디렉트되거나, 혹은 모든 타일 프로세서들에 의해 공유되는 테이블들 내의 데이터를 결합시키기 위한 글로벌 테이블 랜덤-액세스 메모리(RAM)(1118)로 디렉트된다. 이러한 RAM(1118)은 다양한 데이터 값들의 빈도를 기록하기 위한, 예를 들어, 이미지의 전체 휘도를 기록하기 위한, 글로벌 파라미터들, 룩업 테이블들, 및 히스토그램들을 저장한다.

[0041] 도 8 내지 도 11은 본 명세서에서 설명되는 2-차원 데이터 프로세싱 시스템들이 사용하는 계층적 상호연결 라우팅 구성의 단지 일 실시예를 예시한다. 이러한 라우팅 구성은 계층적이고 멀티플렉서-기반인바, 이에 따라 아래에서 더 논의되는 바와 같이 적절한 개수의 비트들을 갖는 타일 프로세서 인덱스(tile processor index)를 사용하여 특정 타일 프로세서가 선택될 수 있다. 그러나, 멀티플렉서들의 크기(즉, 멀티플렉서 입력들의 개수) 및 사용된 상호연결 레벨들의 개수와 같은 세부사항들은 특정 애플리케이션의 비용 및/또는 성능 요건들에 근거하여 본 발명의 기술분야에서 숙련된 자에 의해 변경될 수 있다. 본 발명의 기술분야에서 통상의 기술을 가진 자

에 의해 다른 수정 및 개선이 또한 사용될 수 있다. 예를 들어, 데이터경로들의 폭을 넓히고 경합(contention)을 감소시키기 위해 라우팅 유닛들 및/또는 버퍼들의 복수의 카피들이 라우팅 구성에서의 선택된 포인트들에서 포함될 수 있다.

[0042] 도 11의 예시적 실시예로부터 이해돼야 하는 것으로서, 라우팅 및 버퍼링 유닛들(1100-1116)로 구성된 타일 상호연결은 임의의 타일 프로세서로부터의 액세스들을 임의의 다른 타일 프로세서의 DMEM으로 완전히 결합시키는 것을 구현한다. 각각의 라우팅 계층은 액세스 버퍼들 간의 멀티플렉싱의 두 개의 레벨들로 구성되기 때문에, 이러한 상호연결을 통한 전달들은 일반적으로 타이밍-임계적(timing-critical)이지 않다. 이러한 상호연결 타이밍 톨로런스(interconnect timing tolerance)는 다수의 타일들에 걸친 분배된 비주얼 프로세싱을 지원하는 구현예의 하나의 컴포넌트이다. 다른 컴포넌트들은: 1) 상호연결에 걸쳐 액세스들을 어드레싱하고 라우팅하기 위한 메커니즘; 2) 타일 프로그램들의 실행을 느리게 하는 것으로부터 다수의 싸이클들에서 상호연결의 레이턴시를 피하기 위한 메커니즘; 그리고 3) 액세스된 데이터가 올바른 것임을 보장하기 위한 코히어런시 메커니즘(이것은 값이 유효하게 컴퓨팅되지 않았다면 값이 액세스되지 않는 것, 그리고 값을 요구하는 모든 프로세서들에 의해 값이 액세스되기 전에 값이 오버-라이팅(over-wirting)되지 않는 것을 모두 요구함). 이러한 세 개의 컴포넌트들이 또한 아래에서 논의된다.

[0043] 타일 상호연결을 통한 어드레싱 및 라우팅에 대한 기반은 도 7에서 제시된 DMEM(710)의 분할이다. 이러한 분할의 세 개의 예시적 구성들이 도 12에서 개념적으로 제시된다. DMEM(710)은 예를 들어, 임의의 주어진 영역에 대한 액세스들을 위한 베이스 어드레스 레지스터 세팅(base address register setting)을 사용하여, 동일한 크기의 영역들로 분할된다. 각각의 영역은 타일 내에 픽셀들의 수직 세트(즉, 2-차원 어레이 내의 픽셀들의 컬럼)를 포함하고, 인접하는 영역들은 (2-차원 어레이의 로우를 따라) 타일의 수평 차원을 포함한다. 프로그램들은 단일 영역 내에서 한번에 동작하지만 영역들을 통해 순차적으로 반복되고, 따라서 타일에 걸쳐 수평으로 반복된다. 영역들의 개수는 2를 거듭제곱한 값이고, 이는 타일의 폭에 대응하며, 도 12의 실시예들에서 영역들은 16진수로 순차적으로 번호가 부여되어 있는데, 구성(1200)은 0에서 3까지 번호가 부여된 4개의 영역들을 가지고 있으며, 구성(1202)은 0에서 7까지 번호가 부여된 8개의 영역들을 가지고 있으며, 그리고 구성(1204)은 0에서 F까지 번호가 부여된 16개의 영역들을 가지고 있다. 프로그램의 메모리 요건들에 따라, 영역들에 할당될 필요가 없는 추가적인 메모리가 존재할 수 있다. 이러한 메모리는 임시 값들을 위해 모든 프로그램들에 의해 사용되는 공통의 스필(spill)/필(fill) 영역(1206)을 포함할 수 있다(이것은 모든 영역들에서 이러한 메모리의 복제를 피함).

[0044] 픽셀들의 영역들로의 맵핑을 명확히 보여주기 위해, 도 13a, 도 14 및 도 15는 구성들(1200, 1202 및 1204)에 대해서 스캔-라인 내의 인접하는 픽셀들을 인접하는 타일들 내의 대응하는 영역들로 맵핑시킨 것을 각각 제시하고 있다. 타일 내의 다수의 픽셀들은 대응하는 개수의 영역들로 맵핑되고, 이에 따라 수평 차원은 각각의 타일 프로세서의 DMEM(710) 내에서 부분적으로 수직 맵핑되게 된다. 프레임의 수직 차원은 각각의 영역 내에 직접 맵핑되고 이에 따라 예컨대, 동일한 수평 위치에 있지만 수직 방향으로 있는 픽셀들은 동일한 영역 내에 있게 됨을 이해해야 한다. 2-차원 픽셀 데이터의 이러한 맵핑이 도 13b에서 제시된다. 실행은 픽셀 타일들 간에 병렬로 이루어지지만, 각각의 타일 내에서는 직렬로 이루어지는바, 이는 아래에서 더 논의되는 바와 같다.

[0045] 영역들 간의 액세스를 생성하는 것은 두 개의 어드레싱 단계들을 수반하는바, 하나는 픽셀의 수평 위치를 특정하기 위한 인덱스의 사용이고, 두 번째는 수직 위치를 특정하기 위한 인덱스의 사용이다. 본 발명의 기술분야에서 숙련된 자는 어드레싱의 이러한 형태가 단일 프로세서들 내의 2-차원 어레이들에 액세스하기 위해 사용되는 어드레싱과 동일하게 수행된다는 것을 인식할 것이다. 종래에는, 이러한 두 개의 인덱스 차원들은, 하나의 인덱스에 두 개의 어레이 차원들 중 하나(로우들의 개수 혹은 컬럼들의 개수)를 곱함으로써, 그리고 두 번째 인덱스를 더함으로써, 1-차원 어드레스로 결합된다. 최종 결과가 어레이에 대한 베이스 어드레스에 더해진다. 이렇게 하는 이유는 2-차원 어레이가 선형 메모리 내의 어드레스들의 순차적 세트에 할당되기 때문이다. 대조적으로, 본 명세서에서 설명되는 프로세서들에서, 이러한 어드레스 컴포넌트들은 분리된 상태로 유지된다. 수평 인덱스는 타일의 위치를 정함과 아울러 영역에 대응하는 타일 내의 컬럼 위치를 정하며, 수직 인덱스는 영역의 베이스 어드레스에 대한 영역 내의 메모리 위치를 정함과 아울러 영역 내의 픽셀 값들의 위치를 정한다.

[0046] 타일 상호연결을 통한 액세스들의 라우팅을 위해 수평 인덱스만이 사용된다. 도 16a, 도 16b, 도 16c, 도 16d 및 도 16e는 이러한 인덱스가 도 8 내지 도 11의 라우팅 구성 실시예의 경우 이와 같은 라우팅을 수행하기 위해 어떻게 해석되는지의 예들을 제공한다. 도 16a에서, 수평 혹은 컬럼 인덱스는 16-비트 값(1606)으로 나타나 있다. 이러한 값은 영역 구성이 도 12에서의 1200인지, 혹은 1202인지, 혹은 1204인지에 따라, 도 16b에서의 인덱스(1600)로, 혹은 도 16c에서의 인덱스(1602)로, 혹은 도 16d에서의 인덱스(1604)로 제시되는 바와 같이 각각 해석된다. 각각의 경우에 있어, 최하위 비트들의 개수는 영역 개수를 선택하기 위해 사용되는데, 여기서 비트들

의 개수는 구성된 모든 영역들을 선택하기에 충분하다(4개, 8개, 혹은 16개의 영역들에 대해 2개, 3개, 혹은 4개의 비트들이 각각 있음). 따라서, 도 16b에서의 영역 인덱스(1610)는 2개의 비트들을 가지며, 도 16c에서의 영역 인덱스(1612)는 3개의 비트들을 가지며, 도 16d에서의 영역 인덱스(1614)는 4개의 비트들을 갖는다.

[0047] 각각의 수평 인덱스 내의 그 다음 12개의 더 상위 비트들은 타겟 프로세서 인덱스(target processor index)(1616)를 형성한다. 타겟 프로세서 인덱스(1616)는 도 8 내지 도 11의 라우팅 구성에 의해 연결된 4096개의 타일 프로세서들 중 하나를 식별시키는바, 이는 인접하는 타일 프로세서들이 픽셀들의 인접하는 타일들에 대응하기 때문이다(이것은 결과적으로, 수평 인덱스들(1600 및 1602)에 대해 제시되는 바와 같이, 수평 인덱스 내의 한 개 혹은 두 개의 비트들이 사용되지 않게 할 수 있음). 예를 들어, 프레임 내에서 가장좌측에 있는 타일은 타일 프로세서 0에 대응하고, 그 다음 타일은 타일 프로세서 1에 대응하는 등등이다. 이러한 12개의 비트들은 타겟 타일 프로세서 번호를 형성하며, 타일 상호연결을 통한 액세스의 라우팅을 직접 결정하는데, 이는 도 16e에서 수평 인덱스(1608)에 대한 이러한 필드(field)의 분해(breakdown)에 의해 제시되는 바와 같다. 타겟 프로세서 인덱스(1616)는 라우팅 레벨들에 대응하는 6개의 2-비트 라우팅 레벨 필드들(1618)을 포함하는바, 더 높은 상위 비트들은 도면에서 "로컬", "L1" 등으로 라벨링된 더 높은 라우팅 레벨들에 대응한다. 임의의 레벨에서, 2-비트 필드(1618)는, 도 9에서 MUX들(900-906)에 대한 MUX 입력들(800)에 대응하는, 로컬 액세스 상호연결에 대한 MUX 선택들을 형성한다. 2-비트 필드들은 MUX들(900-906)의 4개의 입력들을 어드레싱하기에 충분하지만, 더 많은 멀티플렉서들을 사용하는 실시예들에 대해서는 더 많은 비트들이 요구될 것이다. 액세스가 로컬인지 아니면 비-로컬인지에 상관없이 도 8 및 도 9에 제시된 원격 액세스 로직(802)에서는 임의의 2-비트 필드(1618)에 비해 더 많은 상위 비트들이 디코딩을 위해 사용된다. 예를 들어, 가장좌측 위치로부터 처음 4개의 타일들은 필드들(L1-L5) 모두에서 제로를 갖는다. 만약 이러한 필드들 중 임의의 필드가 비-제로라면, 액세스는 이러한 그룹에 비-로컬이며, 이러한 경우, 제로 값들을 갖는 필드들(L2-L5)에 의해 결정되는 바와 같이, 만약 액세스가 계층 내에 있다면, L1 필드는 로컬 액세스 상호연결에 대한 레벨 1 라우팅에 의해 사용된다. 만약 필드들(L2-L5) 중 임의의 필드가 비-제로라면, 액세스는 레벨 1에 비-로컬이고, 이것은 도 10에 제시된 레벨 1 액세스 버퍼(1000)에 제공된다.

[0048] 이러한 프로세스는 타겟 타일 프로세서 번호의 상위-순번의 필드들을 연속적으로 사용하여 레벨 5 상호연결(1114)을 포함하는 최대 레벨 5 상호연결(1114)까지 도 11에 제시된 계층 전체에 걸쳐 액세스들의 라우팅을 계속 행함을 이해해야 한다. 이러한 레벨에서, 액세스를 개시한 명령의 타입은, (임의의 다른 타일 프로세서에 대한 액세스에 의해 식별되는) 레벨 5에 로컬인 액세스 또는 (글로벌 테이블 RAM(1118) 혹은 시스템 인터페이스(1120)에 대한 액세스에 의해 식별되는) 레벨 5에 비-로컬인 액세스를 구분하기 위해 사용된다. 명령 타입의 이러한 표시는 프로세서 설계에 관한 기술분야에서 통상의 기술을 가진자에게 알려진 시그널링 방법들 중 임의의 방법을 사용하여 수평 인덱스로부터의 개별 신호를 통해 구현된다.

[0049] 타일 프로세서 액세스 요청들의 경우로 되돌아가서, 타겟 타일 프로세서 인덱스가 그 타겟 타일 프로세서에 액세스할 수 있는 라우팅 유닛을 갖는 상호연결 레벨로 라우팅되는 경우, 액세스 요청에 대한 더 이상의 디코딩은 요구되지 않으며, 라우팅 레벨 필드들에서의 비트들은 그 대응하는 라우팅 레벨에서의 멀티플렉서에 대한 입력들로서 기능한다. 이러한 방식으로, 액세스 요청들은 계층의 상위 레벨들로부터 하위 레벨들로 라우팅된다. 예를 들어, 레벨 5에서의 로컬 라우팅은 도 9에서의 MUX들(900-906)에 대한 MUX 입력들(800)에 대응하는 MUX 선택들을 형성하기 위해 수평 인덱스(1608) 내의 "L5"의 비트들을 사용한다(이러한 라우팅 유닛(910)은 도면에서 제시되는 바와 같이 로컬 그룹에 대한 것이 아니라 레벨 5에 대한 라우팅 유닛임을 이해해야함). 이것은 원격 액세스 입력으로서 레벨 4 라우팅 유닛에 대한 액세스를 결합시키며, 그 포인트에서 1608 내의 "L4"의 비트들은 도 9에서의 원격 액세스 로직(802) 내의 원격 액세스 입력들에 대응하는 MUX 입력들 중 어떤 하나를 선택하며 레벨 3에 대해 선택된다. 이것은 도 11에서의 액세스 버퍼(1110)의 16개의 인스턴스들 중 하나인 레벨 3에 대한 도 10에서 제시된 그 선택된 안으로 들어오는 액세스 버퍼(1002)에 대한 액세스를 결합시킨다. 그 포인트에서, 프로세스는 타겟 타일 프로세서 번호의 하위-순번의 필드들을 연속적으로 사용하여 궁극적 목적지 타일 프로세서까지 계층 전체에 걸쳐 액세스들의 라우팅을 계속 행한다. 그 타일 프로세서에서, 영역 번호는 액세스에 대한 베이스 어드레스를 선택하는바, 이것은 DMEM 내의 데이터에 액세스하기 위한 액세스의 수직 인덱스에 더해진다. 스토어 액세스(store access)들은 액세스에 동반된 데이터를 DMEM에 기입한다. 로드 액세스(load access)들은 액세스를 수행한 타일 프로세서 및 영역을 식별시키는 리턴 인덱스(return index)를 동반한다. 어드레싱된 데이터가 DMEM으로부터 판독되고 액세스의 소스로 리턴되는바, 이것은 방금 설명된 라우팅 프로세스를 사용하여 행해진다. 액세스의 소스에서 수신된 데이터는, 도 7에 제시된 뱅크드 레지스터 파일(708)에 기입되는바, 구체적으로는, 본래 로드 명령에 의해 식별된 레지스터에 기입되고, 그리고 소스 영역과 관련된 특정 레지스터 뱅크에 기입된다. 뱅크드 레지스터 파일은 영역에 의해 독립적으로 사용된 각각의 영역에 대한 레지스터들의 동일한 세

트를 포함하고, 이에 따라 로드 데이터는 해당 영역에 고유한 레지스터에 기입되게 된다.

[0050] 타일 상호연결 어드레싱 및 라우팅 메커니즘이 설명되었고, 그리고 이제 이러한 상호연결을 통한 로드 액세스들의 레이턴시에 의해 일어나는 성능 문제들에 대해 살펴보면, 타일 상호연결을 트래버스(traverse)하기 위해 액세스 요청에 대해 취해진 싸이클들의 개수는, 타일 상호연결을 트래버스하기 위해 데이터 응답에 대해 취해진 싸이클들의 개수에 추가하여, 전형적으로 하나의 싸이클인 종래의 로드의 싸이클 개수보다 훨씬 더 크다는 것이 본 발명의 기술분야에서 숙련된 자에게는 명백할 것이다. 이러한 싸이클들은 로드에 의해 액세스된 데이터를 요구하는 임의의 명령의 실행을 대략 동일한 개수의 싸이클들 만큼 지연시킨다. 이러한 로드들은 상대적으로 빈번하기 때문에(모든 명령들의 대략 20%일 수 있음), 추가적인 5개의 싸이클들은 프로그램에 의해 취해진 싸이클의 개수를 두 배로 늘릴 것이다. 이것은 상당한 성능저하(degradation)를 나타내고, 이것은 본 명세서에서 설명되는 시스템들에서 로드 명령과, 로드 동작에 의해 액세스되는 데이터에 따라 달라지는 임의의 후속 명령 사이에 시간적으로 넓은 폭의 분리를 배치하는 명령 시퀀싱의 신규한 형태를 도입함으로써 피해진다.

[0051] 이러한 명령 시퀀싱은 도 17a에 제시된 마스터 타일 프로세서(1701)에 의해 구현되는바, 마스터 타일 프로세서(1701)는 타일 프로세서(701)의 제 1 인스턴스에 연결된다. 타일 프로세서들(701) 중 첫 번째 타일 프로세서는 또한, 도 8에서 제시되는 방식으로 그 다음 타일 프로세서에 연결되고, 이것은 그 상호연결되는 타일 프로세서 그룹 전체에 걸쳐 계속된다. 도 8 내지 도 11에 제시된 라우팅 구성에 있어서, 이것은 모두 합쳐 최대 4096개의 타일 프로세서들을 포함할 것이다. 마스터 타일 프로세서(1701)에 의해 실행되는 명령들의 시퀀스가 도 17b에 제시되는바, 이것은 태스크 구간으로 지칭된다. 타일 프로세서(701)의 번호가 부여되지 않은 컴포넌트들은 도 7에 제시된 다른 타일 프로세서들에 대한 것과 동일한 것이며, 번호가 부여된 컴포넌트들은 마스터 타일 프로세서에 대해 고유한 것이다. 이러한 것들은, 명령 메모리(IMEM)(1700), 명령 페치 로직(1702), 프로그램 카운터(Program Counter, PC) MUX(1704), 명령 MUX(1706), 제 1 태스크 명령을 저장하기 위한 레지스터(1710), 및 제 2 태스크 명령의 PC를 저장하기 위한 레지스터(1712)를 포함한다.

[0052] 도 17b의 태스크 구간 명령 시퀀스에서, 제 1 태스크 명령(1714)은 제 1 태스크 명령 레지스터(1710)에 로딩되게 된다. 제 2 태스크 명령(1716)에 대한 프로그램 카운터 값은 제 2 PC 태스크 명령 레지스터(1712)로의 로딩을 위한 것이다. 마스터 타일 프로세서(1701) 및 그 관련된 타일 프로세서 어레이에 의한 명령 실행은 도 17c 및 도 17d의 흐름도들과 연계되어 더 상세히 설명된다.

[0053] 도 17c 및 도 17d의 흐름도는 마스터 타일 프로세서(1701)에 의한 프로그램 실행의 프로세스의 실시예를 예시한다. 프로그램을 시작하기 위해, 명령 페치 로직(1702)은 프로그램의 제 1 명령을 페치하는바(도 17c에서의 단계(1720)), MUX(1704)에 대한 "분기 PC(Branch PC)" 입력이 선택되고, 제 1 명령의 어드레스가 적용된다(이것은 임의의 프로세서에 대한 종래의 동작임). 명령 MUX(1706)는 IMEM(1700)으로부터 제 1 타일 프로세서의 I-버퍼 레지스터로 액세스되는 명령을 선택하고(도 17c의 단계(1722)), 여기서 명령이 디코딩 및 실행된다. 타일 프로세서들(701)에 의한 실행이 도 17e의 흐름도와 연계되어 아래에서 더 상세히 설명된다. 명령이 디코딩되는 것과 동시에, 이 명령은 또한 다음 인접한 타일 프로세서의 I-버퍼로 운반되는바, 다음 인접한 타일 프로세서도 명령을 디코딩하고, 이 명령을 또한 다음 인접한 프로세서로 운반한다. 이러한 방식으로, 모든 타일 프로세서들은 동일한 명령 시퀀스를 실행하지만, 각각의 연속하는 타일 프로세서에서 일 싸이클만큼 오프셋되어 있다.

[0054] 제 1 명령이 I-버퍼에 배치되는 경우, 제 1 명령은 이와 동시에 제 1 태스크 명령 레지스터(1710)에 배치된다(도 17c의 단계(1724)). 다음 싸이클시, 명령 페치 로직(1702)은 PC MUX(1704)에서 다음 증분된 PC "PC+1"을 선택하고, 다음 순차적 명령이 액세스되며(도 17c의 단계(1726)) 처음의 것은 디코딩되고 인접하는 타일 프로세서로 운반된다. 이러한 싸이클의 끝에서, "PC+1"에 대한 값은 제 2 태스크 명령 PC 레지스터(1712)(도 17c의 단계(1728))에 배치된다. 제 2 명령은 또한 실행을 위해 제 1 타일 프로세서의 I-버퍼 레지스터로 전송된다(도 17c의 단계(1730)). 레지스터(1710)에 저장된 명령과 레지스터(1712)에 저장된 PC의 결합은 명령 페치 로직(1702)으로 하여금, I-버퍼에 대해 MUX(1706)에서 레지스터(1710)를 선택함으로써 그리고 다음 싸이클시, 페치될 다음 명령에 대한 PC로서 레지스터(1712)를 선택함으로써, 태스크 구간의 실행을 다시 시작하도록 허용한다.

[0055] 태스크 구간의 끝은, 영역들 간의 액세스들, 그리고 레벨 5에서의 시스템 및 테이블 액세스들을 포함하는 타일 상호작용을 사용하여 수행되도록 의도된 원격 액세스를 요구하는 명령에 의해 정의된다. 이것은 도 17b에서 "인스턴스 M(Inst M)"으로 표시된다. 마스터 타일 프로세서는 디코딩 싸이클 동안 이러한 명령들을 검출하고(도 17c의 결정 블록(1732)), 그리고 만약 이러한 명령이 검출되면(블록(1732)의 "예"의 결과) 이것은 명령 페치 로직에 표시되는바, 이는 명령 페치 로직으로 하여금 다음 명령으로서 레지스터(1710)를 선택하도록 하고, 이것은 제 1 명령으로 태스크 구간을 다시 시작하게 한다. 이것은 도 17d에 제시된 흐름도의 일부의 단계들(1744 및

1746)에 대응한다. 다음 싸이클시, 레지스터(1712)는 태스크 구간에서 제 2 명령에 액세스하기 위해 사용되며(도 17d의 단계(1748)), 이 포인트로부터 실행은 순차적으로 행해지는바 이것은 실행이 해당 구간에서 최종 명령에 다시 도달할 때까지 행해지며, 이러한 포인트에서 태스크 구간이 다시 실행된다. 이러한 실행은 도 17c에서 결정 블록(1732)의 "아니오" 분기에 대응하며 도 17d의 단계들(1734, 1736, 및 1738)이 포함된다.

[0056] 각각의 반복된 태스크 구간은, 데이터의 상이한 세트를 사용하여 실행됨으로써 구분되며, 레지스터 파일 내의 레지스터들의 고유한 뱅크 및 DMEM의 고유한 영역이 사용되고, 영역 0에서 시작하여 순차적으로 번호가 부여된 영역들로 진행된다. 도 17c 및 도 17d의 실시예에서, 순차적 영역들을 통한 이러한 실행은 태스크 구간의 끝에 도달하는 경우 영역 카운터를 증분시킴으로써 행해진다(도 17c의 단계(1740)). 이것은 타일에 걸쳐 수평으로 태스크 구간을 반복하는바, 이것은 올바른 동작을 위해 요구되는 것인데 왜냐하면 프로그램은 모든 수평 위치에서 실행돼야만 하기 때문이다. 이러한 반복은 태스크 구간이 구성 내에서 가장 높은 번호가 부여된 영역에서 완료될 때까지 계속된다(도 17c에서의 결정 블록(1742)). 이러한 포인트에서, "PC+1"이 MUX(1704)에서 선택되며(도 17c의 단계(1752)), 명령 실행이 명령 "다음1(Next1)"에서 순차적으로 진행된다. 이것은 다음 태스크 구간의 시작이고, 레지스터들(1710 및 1712)을 세팅하는 프로세스는 구간의 끝이 검출되는 경우 이러한 태스크 구간이 또한 다시 시작될 수 있도록 반복된다. 명령 시퀀싱의 이러한 스타일은 프로그램의 끝까지 계속된다(도 17c의 결정 블록(1750)의 "예" 분기). 도 17c에서의 단계(1754) 내지 단계(1762)는 아래에서 더 설명되는 코히어런시 메커니즘의 실시예를 구현한다.

[0057] 마스터 타일 프로세서가 모든 타일 프로세서들에 대한 명령 시퀀스를 결정하기 때문에, 모든 타일 프로세서들은 영역들에 걸쳐 동일한 반복을 실행한다. 영역들을 통한 이러한 반복은 프로그램 실행 시간을 연장하지만, 처리량을 감소시키는 방식으로 일어나지는 않는데, 왜냐하면 추가적인 싸이클들이 추가적인 결과들을 통해 분배되기 때문이다. 예를 들어, 4개의 영역들을 통한 반복은 중간 결과들의 4개의 세트들을 생성하고, 이것은 프로그램의 단일 반복의 처리량과 동일한 처리량을 갖는다(4/4=1).

[0058] 그러나, 이러한 반복은, 도 18a에서 예시되는 바와 같이, 로드에 의해 액세스된 데이터의 사용으로부터 타일 상호연결을 통해 로드를 넓게 분리시키는 유익한 효과를 갖는다. 이러한 도면은 2개의 태스크 구간들(즉 태스크 구간(1800)("a") 및 태스크 구간(1802)("b"))의 실행을 나타낸다. 모든 구간들(1800)은 명령들의 동일 시퀀스이고, 구간들(1802)도 그러한바, 이에 따라 동일하게 번호가 부여되어 있다. 이미 설명된 바와 같이, 이러한 태스크 구간들은 모든 영역들(본 예에서는 4개의 영역들)에 걸쳐 반복되고, 아울러 2개의 인접하는 타일 프로세서들("N" 및 "N+1")에 걸쳐 반복되는바, 도면에서 시간은 수직으로 진행한다. 상이한 영역들에서의 태스크 구간의 실행은 "a-0" 내지 "a-4" 및 "b-0" 내지 "b-3"으로서 식별되는바, 이는 각각의 타일 프로세서 내의 영역들(영역 0 내지 영역 3)에서의 실행을 나타낸다. 인접하는 타일 프로세서에서의 동일한 태스크 구간의 실행은 도 7의 I-버퍼들(700)을 통한 명령들의 분배로 인해 일 싸이클만큼 오프셋된다. 태스크 구간(1800)은 태스크 구간의 끝을 정의하는 원격 로드("로드(load)")로 끝난다. 태스크 구간(1802)은 로드된 데이터를 사용하는 명령("사용(use)")으로 시작한다.

[0059] 도 18a는 인접하는 타일 프로세서들 및 이들 각각의 데이터 메모리 영역들의 공간적 표현을 제공한다. 도 18b에서 제시되는 바와 같이 동일 명령 실행들의 시간-기반 표현이 또한 도움을 줄 수 있다. 도 18b에서, y-축은 명령들이 실행되고 있는 타일 프로세서의 데이터 메모리의 영역을 나타내고, x-축은 클럭 싸이클들을 나타낸다. 위쪽 도면은 타일 프로세서 N에 대한 것이고, 아래쪽 도면은 인접하는 타일 프로세서 N+1에 대한 것이다. 도시된 대각선은 4개의 명령들의 태스크 구간(1800)을 통해 진행하는 것을 나타내며, 이 경우 타일 프로세서의 데이터 메모리의 영역 0에서의 적절한 데이터에 대한 작용이 일어나고 그 다음에 데이터 메모리의 영역 1, 영역 2, 그리고 영역 3에서 동일 명령 세트가 순차적으로 실행된다. 태스크 구간(1800)의 끝에서 "로드" 명령이 4개의 영역들 모두에서 실행되는 경우, 명령들의 다음 세트(태스크 구간(1802))가 영역 0에서 다시 실행을 시작한다. 동일한 시퀀스가 타일 프로세서 N+1에 대해 일어나는데, 다만 이것은 타일 프로세서 N에서의 실행 이후 일 클럭 싸이클 뒤에 일어난다는 점에서 차이가 있다. 도 18b는 이러한 실시예에 대해 각각의 명령이 하나의 클럭 싸이클을 요구한다고 가정한 것이며, 데이터 메모리의 새로운 영역에서의 실행 시작과 관련된 혹은 명령들의 새로운 세트의 실행을 시작하는 것과 관련된 어떠한 클럭 싸이클도 존재하지 않는다고 가정한 것이다.

[0060] 도 18a 및 도18b의 실시예에서의 태스크 구간들의 길이는 4개의 명령들이다. 실제 태스크 구간들은 전형적으로 더 길지만, 소프트웨어 최적화는 태스크 구간들이 그 길이가 4개의 명령들처럼 최소의 길이를 갖도록 보장할 수 있다. 이것은 가능한데 왜냐하면 컴파일러(compiler)는 태스크 구간들을 끝내는 명령들의 타입들을 결정할 수 있고, 그리고 코드 모션 최적화(code motion optimizations)(이것은 태스크 구간으로 하여금 원하는 개수의 명령들을 포함하도록 함)를 수행할 수 있기 때문이다. 도면으로부터 알 수 있는 바와 같이, 그 길이가 적어도 4개

의 명령들인 태스크 구간들의 사용은 원격 로드를 그 사용으로부터 적어도 12개의 싸이클들(3개의 다른 영역들 각각에서 4개의 싸이클들)만큼 분리시킨다. 이것은 도 18b에서 구간(1804)으로 예시되어 있는바, 이는 타일 프로세서 N의 영역 0에서의 로드 명령의 실행과 동일 영역에서의 후속 명령 실행 동안 그 로드된 데이터의 사용 사이의 시간을 나타낸다. 따라서, 만약 로드 동작이 12 싸이클들에서 완료된다면, 이러한 데이터를 사용하는 명령에 대한 명령 실행에서의 지연은 존재하지 않는다. 이용가능한 시간은 영역들의 개수와 함께 증가하고 이에 따라 8개의 영역들을 갖는 경우 로드는 28개의 싸이클들(4x7)에서 완료할 수 있고 16개의 영역들을 갖는 경우에는 60개의 싸이클들(4x15)에서 완료할 수 있음이 또한 명백해 진다.

[0061] 태스크 구간들의 길이가 4개의 명령들인 그러한 태스크 구간들의 실행에 대해 그 요구된 액세스 레이턴시에 있어서 최소한 가장 나쁜 경우를 나타내는 도 18에 제시된 타이밍이 원격 액세스들에 의해 일어나는 프로그램 실행 지연을 방지하기에 충분함을 예시하기 위해, 도 19는 타일 상호연결을 통한 다양한 라우팅 경로들에 대해 요구된 싸이클들의 개수를 보여준다. 요구된 싸이클들(여기에는 타겟 타일 프로세서에 대한 요청을 라우팅하기 위한 싸이클, 그 타겟에서 DMEM에 액세스하기 위한 싸이클, 그리고 데이터 응답을 라우팅하기 위한 싸이클들이 포함됨)의 개수는 라우팅을 수행하기 위해 요구된 상호연결의 레벨에 따라 달라진다. 도 19a는 로컬 및 레벨 1 상호연결을 통해 라우팅될 수 있는 액세스에 대해 필요한 싸이클들의 시퀀스(1900)를 보여주는바, 로드가 싸이클 1에서 실행되는 때로부터, 다음과 같은 것들 각각에 대해, 즉, 로컬 및 레벨 1 요청 라우팅, DMEM 액세스, 로컬 및 레벨 1 응답 라우팅, 및 레지스터 기입 각각에 대해, 하나의 싸이클이 취해진다. 레지스터가 기입되기 때문에, 데이터는 또한 레지스터 파일을 바이패스(bypass)할 수 있고 실행을 위해 사용될 수 있는바, 이에 따라 프로그램 실행 지연을 피하기 위해 싸이클 2와 싸이클 4를 포함하는 그러한 싸이클 2 내지 싸이클 4에서 단지 3개의 중간개입 명령(intervening instruction)들만이 요구된다. 도 19b에서의 시퀀스(1902)는 레벨 2 및 레벨 3 상호연결을 통해 라우팅될 수 있는 액세스에 대한 것이며, 이 경우 지연을 피하기 위해 요구된 총 7개의 중간개입 명령들에 대한 추가적인 레벨들을 트래버스하기 위해 시퀀스(1900)에 의해 요구된 것보다 4개의 싸이클들이 더 부가된다.

[0062] 도 19c에서의 시퀀스(1904)는 글로벌 RAM 액세스에 대한 것으로, 이것은 레벨 4 및 레벨 5를 통한 라우팅을 포함하고, 이 경우에 있어서도 시퀀스(1900)에서의 개수보다 4개의 싸이클들이 더 부가되어 있고 7개의 중간개입 명령들이 요구된다. 마지막으로, 도 19d에서의 시퀀스(1906)는 요청과 응답 모두를 라우팅시키기 위해 모든 레벨들을 필요로 하는 가장 나쁜 경우에 대한 것이다. 여기에는 시퀀스(1902)에 의해 요구된 것들보다 4개의 싸이클들이 더 부가되고, 지연을 피하기 위해 11개의 중간개입 명령들이 요구된다. 타일 프로세서 데이터 메모리 영역들에 걸친 태스크 구간 실행의 반복이 도 18로부터 알 수 있는 바와 같이 적어도 12개의 중간개입 명령들을 제공하기 때문에, 그리고 도 19d의 동작에 대해 11개가 요구되기 때문에, 액세스는 가장 나쁜 경우에도 레이턴시로 인한 프로그램 실행 지연 없이 타일 상호연결을 사용하여 수행될 수 있다. 만약 더 많은 영역들이 사용된다면, 타이밍은 덜 임계적이게 되는데, 왜냐하면 이것은 훨씬 더 많은 중간개입 명령들을 제공하기 때문이다(8개의 영역들에 대해서는 28개, 그리고 16개의 영역들에 대해서는 60개).

[0063] 이제 코히어런시의 문제로 돌아가서, 타일 상호연결을 통해 액세스들을 라우팅시킬 수 있는 것 그리고 로드 액세스의 레이턴시가 프로그램 실행을 지연시키는 것을 방지할 수 있는 것으로는 충분하지 않다. 타일 프로세서들의 실행은 I-버퍼들에 의해 그리고 영역들에 걸친 반복에 의해 오프셋되어 있다. 로드에 의해 액세스되는 데이터는 직렬 명령 실행에서의 그 로드에 앞서 가장 최근에 스토어에 의해 기입된 데이터일 것, 그리고 로드 이후의 스토어 혹은 좀 더 이른 스토어에 의해 기입된 데이터가 아닐 것을 보장하기 위한 어떤 메커니즘이 존재해야만 한다. 달리 말하면, 요청된 데이터는 그 요청 명령에서 사용되기 위해서 제시간에 도착해야만 할 뿐만 아니라 올바른 데이터이어야만 한다. 스토어들 및 로드들의 이러한 가상적 직렬 시퀀스는 타일 실행이 직렬이 아닌 경우에도 코히어런시 메커니즘에 의해 효과적으로 재구성돼야만 한다.

[0064] 코히어런시 메커니즘의 동작은 종속성 그래프에 의해 설명될 수 있는바, 이것의 예가 도 20에서 제시된다. 이러한 그래프에서, 메모리 내의 데이터는 블록들(2000-2004)로 나타나 있는바, 블록들(2000 및 2002)은 시스템 메모리 내의 데이터를 나타내고, 블록(2004)은 타일 프로세서 DMEM 내에 보유된 데이터를 나타낸다. 그래프 노드들(원형의 것들)(2006-2016)은 명령 동작들을 나타낸다(이것은 종속성 그래프들에 대한 종래의 표기법임). 명령 입력들은 노드의 상부에서 안으로 들어오는 화살표들로 나타나 있으며 "사용(use)"으로 라벨링되어 있고, 출력은 노드의 하부에서 밖으로 나가는 화살표로 정의되어 있으며 ("값을 정의함(define a value)"을 나타내는) "def"로 라벨링되어 있다. 하나의 노드로부터 다음 노드로의 화살표는 아크(arc)이고, 이것은 밖으로 나가는 화살표에 의해 정의된 결과 값이, 안으로 들어오는 화살표에 대한 입력 오퍼랜드 값(input operand value)으로서 사용됨을 표시하는바, 아크는 결과와 오퍼랜드 값 간의 종속성을 정의한다.

[0065] 모든 타일 프로세서들은 동일한 명령 시퀀스를 실행하고, 이에 따라 종속성 그래프는 모든 프로세서들에 대해 동일하며, 단지 데이터 값만이 다르다. 도 20은 3개의 타일 프로세서들을 보여주며, N, M 및 P로 라벨링되어 있고, 이들은 반드시 인접해 있을 필요는 없으며, 이미지 프레임(혹은 다른 2-차원 어레이)의 수평 차원을 따라 어디든 위치하는 데이터를 포함할 수 있다. 이것은 종속성 그래프가 타일 프로세서들의 임의의 범위를 가로지를 수 있음을 강조하고 있으며, 코히어런시 메커니즘에 의해 다루어질 문제들의 본질을 예시하고 있다. 일부 종속성 아크들은 타일 프로세서에 로컬이다(예를 들어, 타일 프로세서 N에서 노드(2006)로부터 노드(2012)로의 아크, 노드(2010)로부터 노드(2012)로의 아크, 그리고 노드(2012)로부터 노드(2016)로의 아크). 다른 아크들은 비-로컬이다(예를 들어, 타일 프로세서 M에 의해 실행 중에 있는 노드(2014)로부터 타일 프로세서 N에 의해 실행 중에 있는 노드(2016)로의 아크, 그리고 타일 프로세서 P에 의해 실행 중에 있는 노드(2008)로부터 타일 프로세서 N에 의해 실행 중에 있는 노드(2010)로의 아크). 본 예는 명확한 설명을 위해 타일 프로세서 N의 종속성들에 초점을 맞추고 있지만 도 20의 실시예에서 모든 타일 프로세서들은 번호가 부여된 것들에 대응하는 종속성들을 가짐을 이해해야 한다.

[0066] 코히어런시 메커니즘을 이해함에 있어서 본질적으로 주시해야 하는 것은, 원격으로-액세스되는 값에 대한 임의의 정의하는 명령은 모든 타일 프로세서 및 영역에 대해 동일한 태스크 구간 내에서 일어난다는 것을 인식하는 것인데, 왜냐하면 이러한 명령은 구간의 끝을 정의하고, 모든 영역은 동일한 명령 시퀀스를 실행하기 때문이다. 더욱이, 모든 태스크 구간은 이전의 태스크 구간을 순차적으로 따르기 때문에, 태스크 구간들은 태스크 구간이 모든 영역들에서 실행될 때 증가하는 (프로그램의 시작에서 값 0으로 시작하는) 카운터를 사용하여 식별될 수 있다. 이것은 태스크 구간에 대한 태스크 ID로서 지칭된다.

[0067] 코히어런시 메커니즘을 이해함에 있어서 본질적으로 주시해야 하는 또 하나의 다른 것은, 비주얼 데이터가 공유되기 때문에 비주얼 데이터는 프로세서와 주변 디바이스 간에 공유되는 데이터의 (단일-프로세서 시스템에서의) 간단한 경우에 대해서도 임의의 공유된 데이터에 적용되는 두 가지 속성들을 갖는다는 것이다. 이러한 속성들 중 첫 번째는, 주변기기로의 직렬 출력 동안 데이터가 단지 한 번만 기입되는 것과 유사하게, 프로그램의 반복 동안 데이터가 단지 한 번만 기입("한번-기입(write-once)")된다는 것이다. 이러한 속성이 없다면 반복들 간에 데이터를 공유하는 것은 불가능하게 되는바, 이것은 수직 차원에서 데이터를 공유하기 위해 요구된다(왜냐하면 프로그램 반복은 수직 차원에 있기 때문). 예를 들어, 만약 도 20에서 그 보유된 데이터(2004)를 포함하는 메모리 위치가 두 번 기입된다면, 두 번째 기입의 값만이 후속 반복들을 위해 보전될 것이며, 첫 번째 것은 후속의 반복들과 올바르게 공유될 수 없다. 이것은 전형적인 프로세싱 파이프라인들(예를 들어, 도 1의 프로세스 참조)에 의해 지원되는데, 왜냐하면 데이터가 프로세싱 스테이지들 사이에 버퍼링되어 하나의 스테이지에 의해서는 출력으로서 기입되고 또 하나의 다른 스테이지에 의해서는 입력으로서 판독되기 때문이다. 두 번째 속성은 공유된 데이터가 휘발성이라는 것인데, 이것이 의미하는 바는, 주변기기에 대한 출력 데이터는 주변기기에 기입돼야 하는 요건과 유사하게, 데이터 값들은 DMEM에서 반영돼야만 하고, 프로세서 레지스터들만을 통해서는 명령들 간에 전달될 수 없다는 것이다.

[0068] 한번-기입 속성은 임의 개수의 사용들에 대해 종속성 그래프에서 정의되는 단지 하나의 값만이 존재함을 보장한다. 더욱이, 휘발성 속성은 그 정의된 값들이 서로 다를지라도 각각의 정의된 값에 대응하는 모든 영역 및 모든 타일 프로세서 내에 스토어가 존재함을 보장한다. 이러한 스토어들이 임의의 고유하게 번호가 부여된 태스크 구간에서 수행된다는 사실을 함께 고려하는 경우, 다음과 같은 것이 적용돼야 하는데, 즉 종속성 그래프 내의 임의의 아크는 태스크 ID에 의해 고유하게 식별될 수 있는 것, 이러한 태스크 ID를 갖는 하나의 그리고 단지 하나의 스토어 명령이 존재한다는 것, 그리고 모든 프로세서는 동일한 대응하는 스토어 명령에 대해 (비록 이것이 타일 프로세서들 및 영역들에 걸쳐 여러 번 실행될지라도) 동일한 태스크 ID를 갖는다는 것이 적용돼야 한다.

[0069] 도 17로 다시 돌아가서, 마스터 타일 프로세서는 스칼라 제어 RAM(1708)(이것은 이전에 설명되지 않은 것임)을 포함한다. 또한, 도 7로 다시 돌아가서, 각각의 타일 프로세서는 영역 상태 로직(704)(이것은 이전에 설명되지 않은 것임)을 포함한다. 스칼라 제어 RAM(1708)(이것은 관련된 제어 로직(1714)을 가짐) 및 영역 상태 로직(704)은 함께 코히어런시 프로토콜을 구현한다. 스칼라 제어 RAM은 비주얼 데이터(혹은 다른 2-차원 데이터 요소들)와 관련된 스칼라 데이터를 포함한다. 본 명세서에서 설명되는 시스템의 실시예에서, 비주얼 데이터는 C++ 프로그래밍 언어에서 오브젝트들로 나타내어지며, 이러한 오브젝트들은 스칼라 정보(예를 들어, 오브젝트의 차원들 및 어드레싱 변수들)뿐만 아니라 벡터 정보(이것은 픽셀들 및 관련된 데이터를 나타내는 비주얼 데이터임)를 모두 포함한다. 스칼라 정보는 마스터 타일 프로세서의 스칼라 제어 RAM 내에만 포함되며, 벡터 정보는 타일 프로세서들의 DMEM에 걸쳐 분배된다. 이 둘 간의 관계는 스칼라 데이터가 모든 영역들의 DMEM 내에 (각각의 영역의 베이스 어드레스로부터의 오프셋을 나타내는) 비주얼 데이터에 대한 어드레스 포인터(address

pointer)를 포함한다는 것이다.

[0070] 이것은 종속성 그래프 내의 값들을 정의하는 스토어들을 이러한 값들을 사용하는 로드들과 관련시키는 메커니즘을 제공한다. 비주얼 오브젝트가 스토어에 의해 기입되는 경우, 이러한 스토어는 먼저 제 1 타일 프로세서의 영역 0에서 수행된다. 비주얼 벡터 데이터(이것은 이러한 영역 내에서 스칼라임, 벡터 내의 많은 값들 중 하나)를 기입하는 것과 동시에, 마스터 타일 프로세서는 스칼라 제어 RAM(1708) 내에 오브젝트의 태스크 ID 변수(이러한 변수는 오브젝트의 각각의 인스턴스에 고유한 것임)를 기입하고, 그리고 오브젝트에 대한 스토어를 수행하는 태스크 구간의 태스크 ID를 기록한다. 스토어는 그 저장된 데이터를 사용하여 임의의 후속 명령의 실행 이전에 DMEM의 모든 영역들에서 실행될 것이기 때문에, 태스크 ID의 이러한 기입은, 스토어가 DMEM의 마지막 영역에서 실행된 이후, 단지 한 번만 수행될 수 있다. 이러한 프로세스가 도 17c의 실시예에서 단계(1760) 및 단계(1762)로 제시되어 있다. 비주얼 오브젝트가 로드에 의해 판독되는 경우, 마스터 타일 프로세서는 이러한 태스크 ID 변수를 판독하고 이것을 로드 명령과 관련시키는바, 로드 명령은 모든 타일 프로세서들에 분배되고 모든 영역들에서 실행된다. 로드 명령의 태스크 ID와의 이러한 관련은, 도 17c의 실시예에서의 단계(1754) 내지 단계(1758)에 의해 구현되는 바와 같이, 각각의 원격 로드 명령에 대해 수행된다.

[0071] 타일 프로세서들에서, 영역 상태 로직(704)은 두 개의 값들을 유지하는바, 하나는 실행되고 있는 명령들의 현재 태스크 ID를 식별시키는 것이고, 또 하나의 다른 하나는 어떤 영역들이 그 대응하는 태스크 구간을 완료했는지를 표시하는 바이너리 값(binary value)들을 포함하는 것이다. 타일 프로세서에 의해 프로그램을 실행하기 위한 프로세스의 실시예를 예시하는 흐름도가 도 17e에 제시된다. I-버퍼로부터 안으로 들어오는 명령을 디코딩하면서, 타일 프로세서는 또한 인접하는 타일 프로세서의 I-버퍼로 명령을 운반한다(도 17e의 단계(1782) 및 단계(1784)). 타일 프로세서는 안으로 들어오는 명령들을 실행하는바, 이것은 원격 액세스 명령이 태스크 구간의 끝을 일으킬 때까지 행해진다(단계(1768) 및 단계(1770)에서 제시됨). 태스크 구간이 DMEM의 영역 내에서 완료되는 경우, 태스크 구간이 어떤 영역들 내에서 완료되었지에 관한 기록이 유지된다(단계(1772)). 도 17e의 실시예에서는, 이러한 기록을 유지함에 있어서 태스크 ID 카운터가 사용된다. 그 다음에, 동일한 태스크 구간의 실행이 DMEM의 후속 영역들에서 반복되는데(단계들(1774, 1776, 1768, 및 1770)), 이것은 태스크 구간이 모든 영역들에서 실행될 때까지 행해진다(결정 블록(1776)). 이러한 포인트에서, 마스터 타일 프로세서로부터 수신된 다음 명령들은 다음 태스크 구간에 대한 것일 것이고, 이러한 다음 태스크 구간은 모든 영역들에 걸쳐 다시 실행된다.

[0072] 타일 프로세서들에서의 코히어런시 요건들은 원격 상호연결로부터의 원격 요청들을 처리하는 개별 프로세스에 의해 유지된다. 코히어런시 요건을 충족시키는 로드에 대한 기준은, 로드의 타겟인 타일 프로세서 및 영역에서, 해당 영역이 (로드와 관련된 태스크 ID에 대응하는) 태스크 구간을 완료했어야한다는 것이다. 이것은 로드의 태스크 ID를 영역에 의해 완료된 태스크 ID와 비교함으로써 표시되는바, 영역의 태스크 ID는 로드와 관련된 태스크 ID보다 크거나 같아야만 한다. 이것은 스토어를 수행한 태스크 구간의 태스크 ID이기 때문에, 이러한 기준은 타겟 영역이 스토어의 포인트를 넘어 실행되었음을 보장함과 아울러 영역이 올바른 값으로 기입되었음을 보장한다.

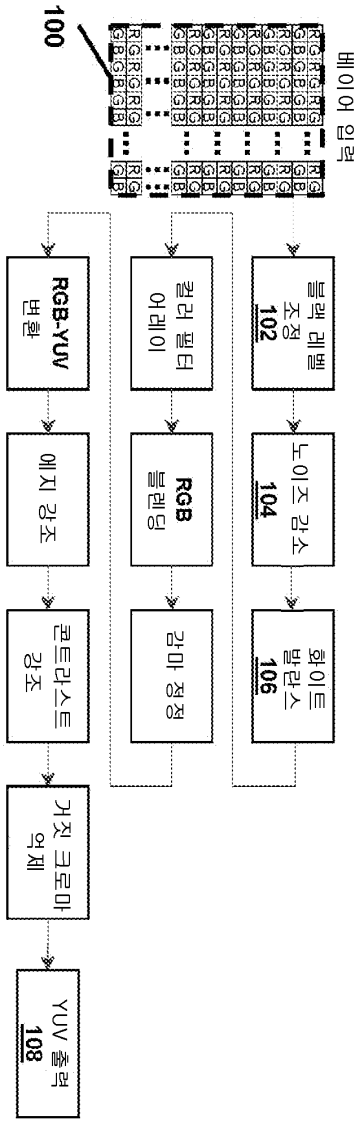
[0073] 만약 이러한 기준이 충족되지 않는다면, 액세스는 기준이 충족될 때까지 타겟 타일 프로세서 내에서 홀딩(holding)되는바, 이것은 궁극적으로 스톨을 생성할 수도 있다. 이러한 스톨은 실행을 지연시킬 수 있지만 이러한 것은 거의 일어나지 않는다. 이것은 도 21에서의 예에 의해 알 수 있다. 이 도면은 도 18a와 유사하지만 3개의 태스크들의 실행이 9개의 타일 프로세서들 및 4개의 영역들을 가로지르는 것을 보여준다. 태스크 구간(2100)(이것은 명확한 설명을 위해 타일 프로세서 N+8에 대해서만 제시된 것으로 모든 타일 프로세서들에 대해서 동일함)은 데이터 값들을 정의하는 스토어로 끝난다. 태스크 구간(2102)은 값들을 판독하는 로드에서 끝난다. 태스크 구간(2104)(본 예에서 이것은 영역 0에 대해서만 제시된 것임)은 로드된 값의 사용으로 시작한다. 태스크 구간들은 각각의 구간에서 4개의 명령들이 있는 것으로 가정한 스케일로 제시되어 있다. 인접하는 타일 프로세서들이 일 싸이클만큼 떨어져 동일한 명령들을 실행하기 때문에, 도 21의 실시예에서, 타일 프로세서 N+4는 타일 프로세서 N보다 4개의 싸이클들만큼 더 늦게 태스크 구간(2100)을 시작하고, 이에 따라 타일 프로세서 N+1이 자신의 영역 1에서 실행을 시작하는 것과 동시에 타일 프로세서 N+4가 자신의 영역 0에서 실행을 시작한다. "L4/L5" 및 "L2/L3"으로 라벨링된 화살표들은, 타일 상호연결의 레벨 4와 레벨 5를 트래버스하는 로드들 및 레벨 2와 레벨 3을 트래버스하는 로드들에 대한 종속성 아크들을 각각 나타낸다. (9개의 타일 프로세서들이 인접하고 있지만, 이들 간의 액세스들은, 만약 프로세서들의 그룹이 프로세서들의 로컬 그룹들 사이의 경계를 가로지른다면, 상호연결의 복수의 레벨들을 요구할 수 있다.) 아크들은 액세스의 최대 범위를 나타내는바, 이것은 결과적으로 실행에 있어 어떠한 지연도 발생시키지 않는다. 이러한 범위는 L4/L5 라우팅에 대해서는 22

개의 영역들이고, L2/L3 라우팅에 대해서는 34개의 영역들이다. 명확한 설명을 위해 도면에서는 제시되지 않았지만, 로컬 및 레벨 1 라우팅은 최대 64개의 영역들에 걸친 범위를 가질 수 있다. 이러한 범위들은 22개, 34개 및 64개의 픽셀들의 에이프런 액세스에 각각 대응하는바, 이것은 비주얼 프로세싱을 위해 전형적으로 요구된 것보다 훨씬 더 큰 것이다. 이러한 범위는 전형적으로 훨씬 더 큰데 왜냐하면 로드 데이터의 스토어들, 로드들 및 사용들이 도면에서 제시된 것보다 더 많은 명령들에 의해 분리되어 있기 때문임을 또한 이해해야만 한다. 더욱이, 이러한 범위는 8개 혹은 16개의 영역들을 갖는 구성들에 대해 훨씬 더 크다.

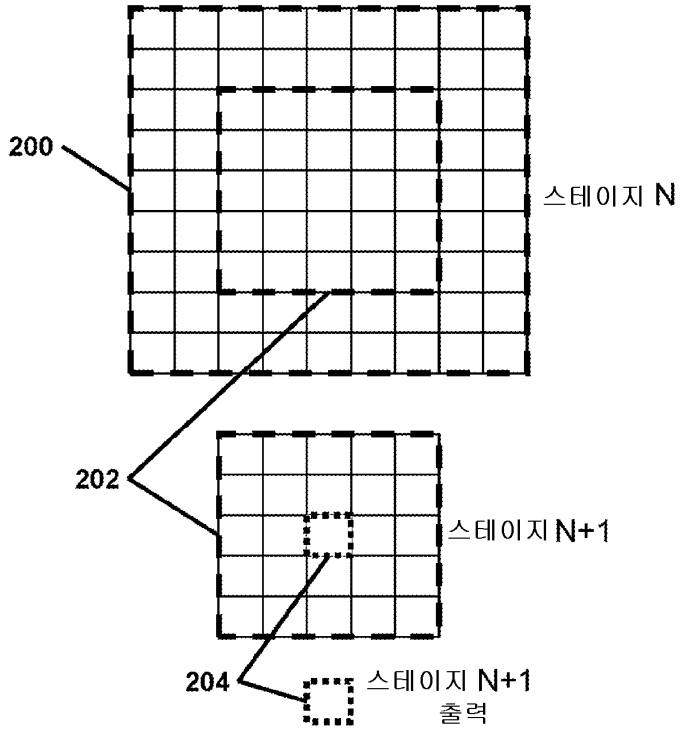
[0074] 본 명세서에서 설명되는 시스템들, 프로세서들, 및 방법들은 타일 프로세서들 간의 비주얼 데이터의 결합을 제공하고, 그리고 로드 액세스들의 레이턴시로 인한 성능 저하를 피하는 명령 시퀀싱을 제공하며, 그리고 전형적으로 데이터 종속성들로 인한 성능 저하를 또한 피하는 코히어런시 메커니즘을 제공한다. 본 명세서에서 제공되는 설명은 본 발명의 원리 및 다양한 실시예를 예시하기 위한 것이다. 본 발명의 기술분야에서 숙련된 자들에게는 앞에서 개시된 내용이 완전히 이해되는 경우 많은 변형들 및 수정들이 가능하다는 것이 명백하게 될 것이다. 다음의 청구항들은 이러한 변형들 및 수정들을 모두 포괄하는 것으로 해석되도록 의도된 것이다.

도면

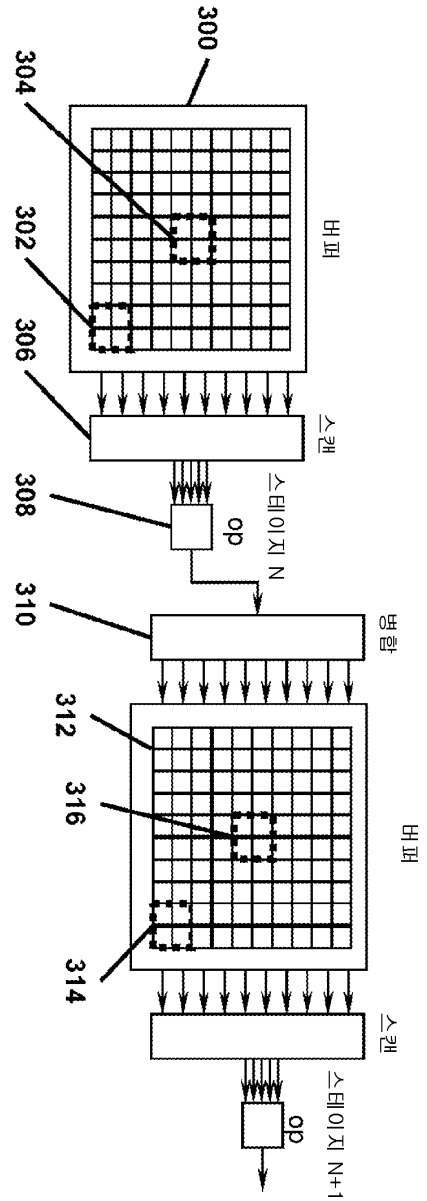
도면1



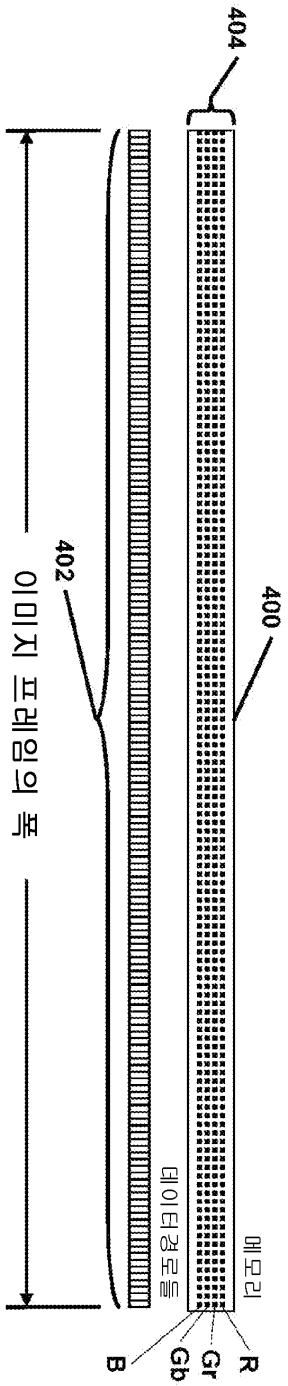
도면2



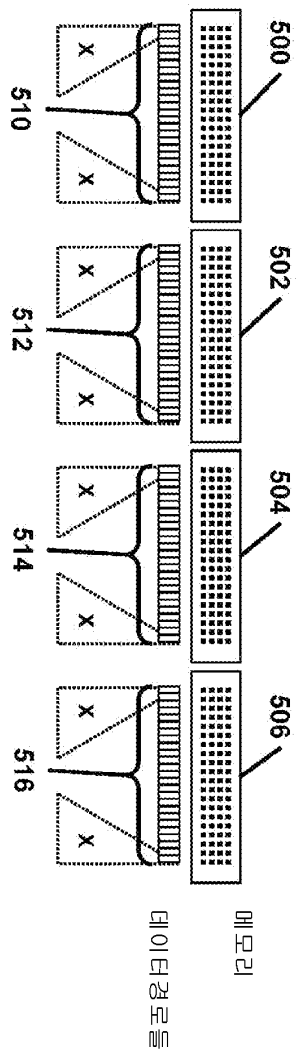
도면3


300
304
302
306
308
310
312
316
314
버퍼
스캔
스테이지 N
op
동기
버퍼
스캔
스테이지 N+1
op

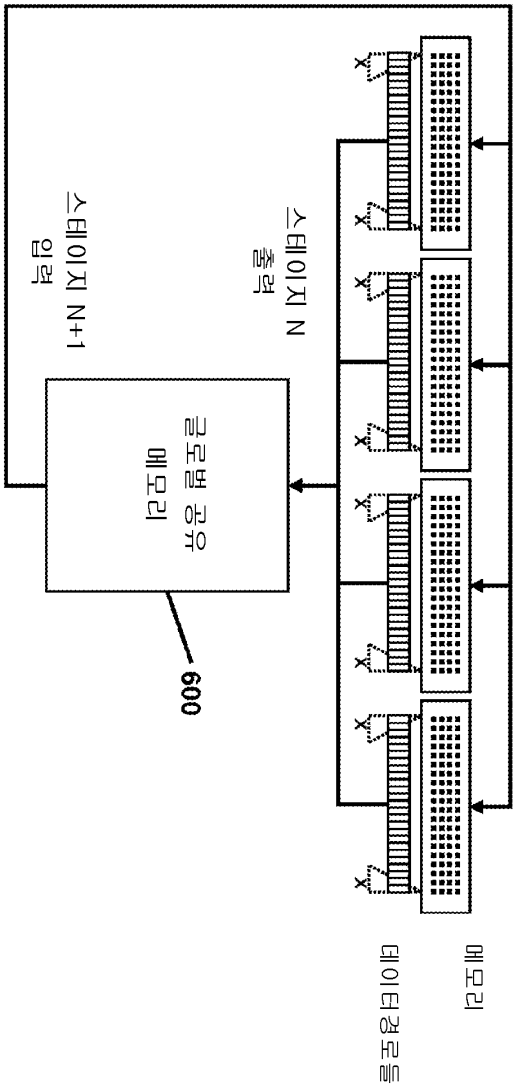
도면4



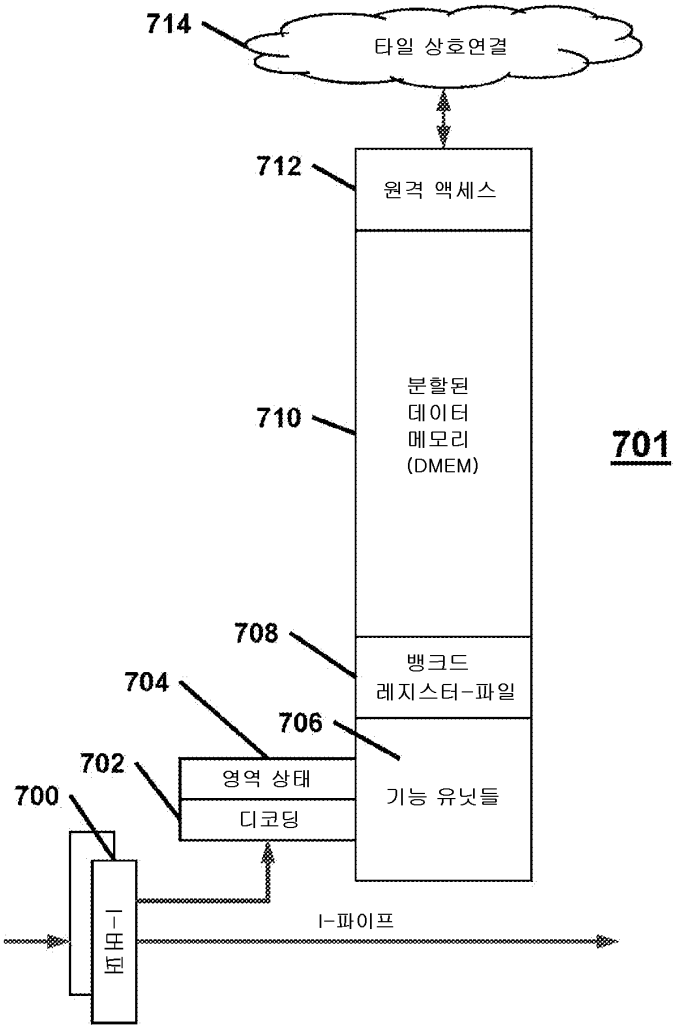
도면5



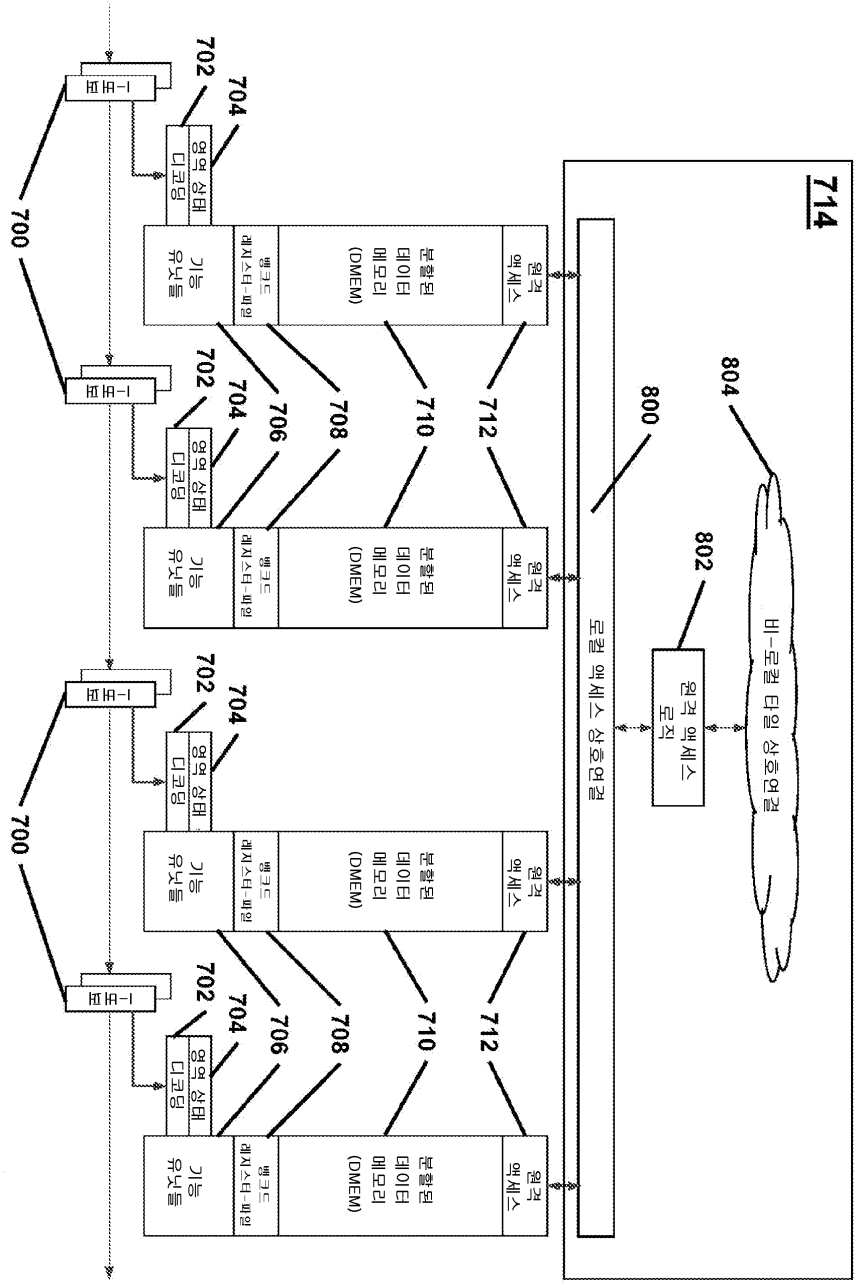
도면6



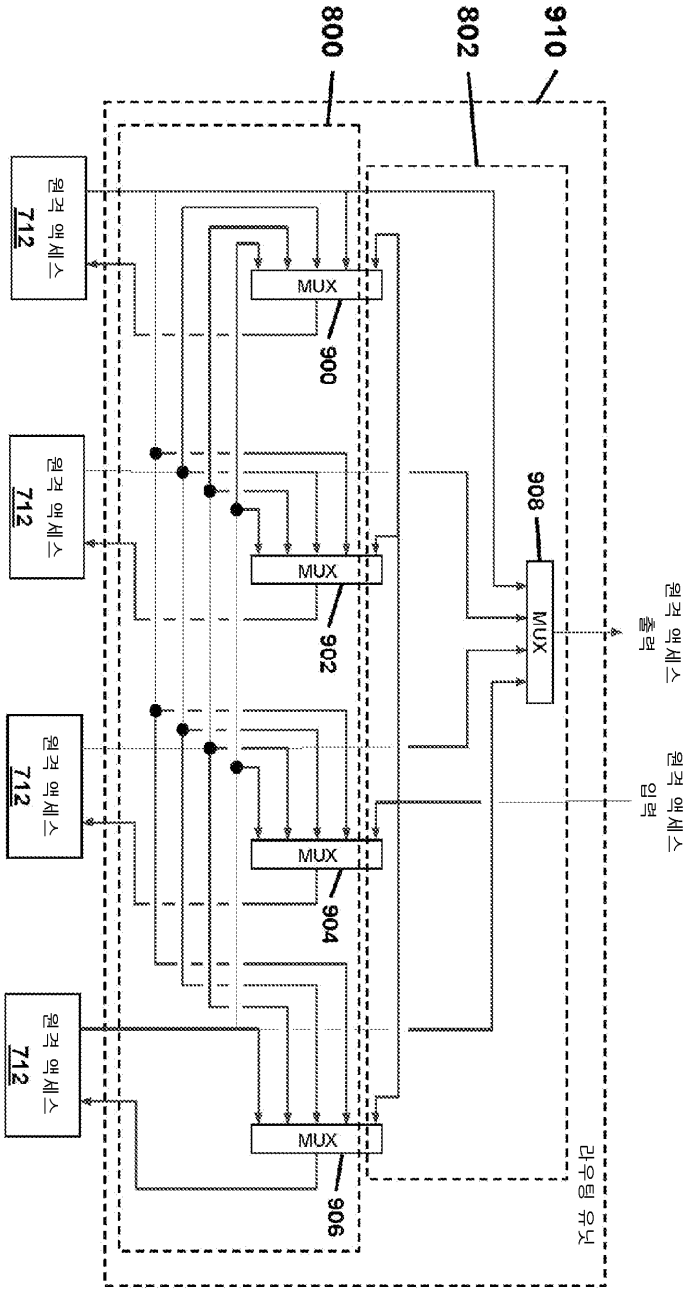
도면7


714
타일 상호연결
712
원격 액세스
710
분할된
데이터
메모리
(DMEM)
701
708
뱅크드
레지스터-파일
704
706
702
영역 상태
디코딩
기능 유닛들
700
I-버퍼
I-파이프

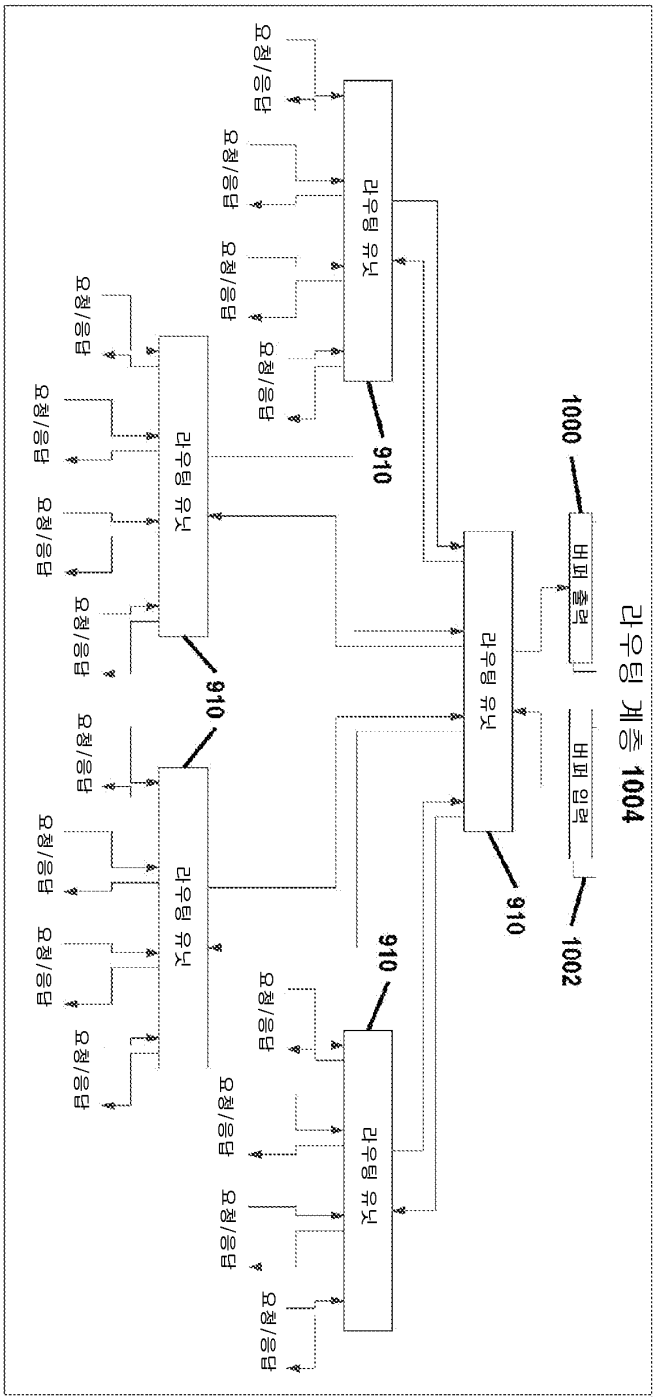
도면8


714
700
702
704
706
708
710
712
800
802
804
I-버퍼
영역 상태 디코딩
기능 유닛들
레지스터-버퍼 뱅크
로컬 데이터 메모리 (DMEM)
로컬 액세스 상호연결
원격 액세스 로직
비-로컬 타일 상호연결

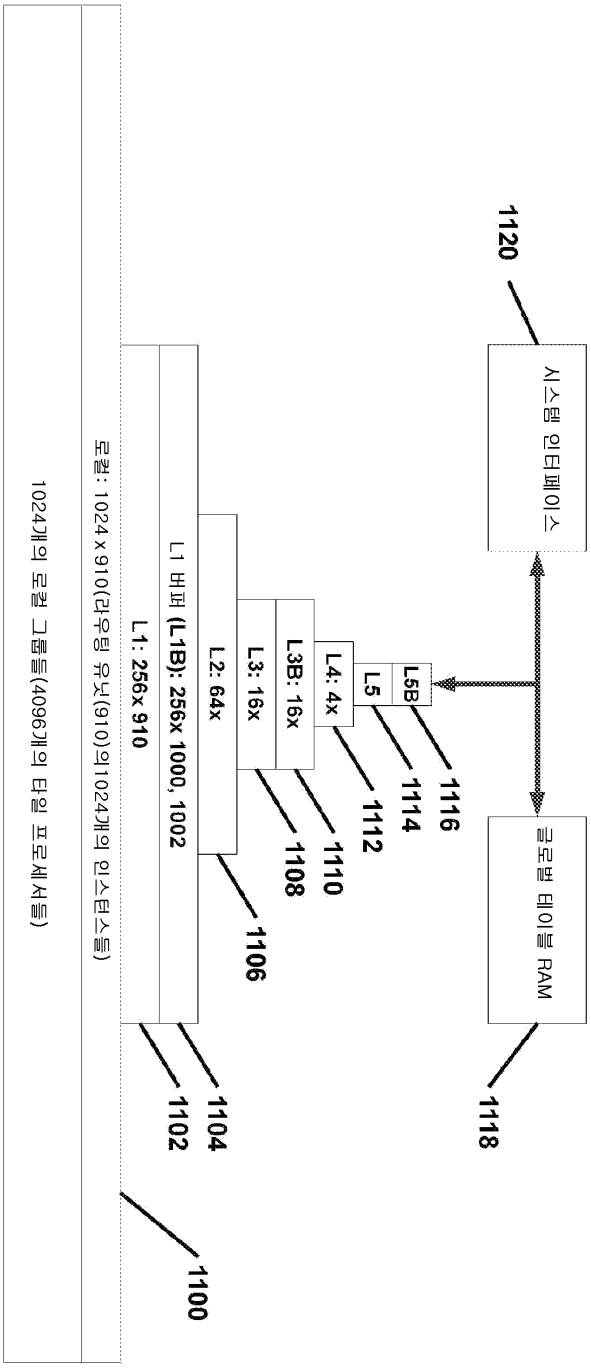
도면9


800
802
910
원격 액세스 712
원격 액세스 712
원격 액세스 712
원격 액세스 712
MUX
900
MUX
902
MUX
904
MUX
906
908
MUX
원격 액세스 출력
원격 액세스 입력
라우팅 유닛

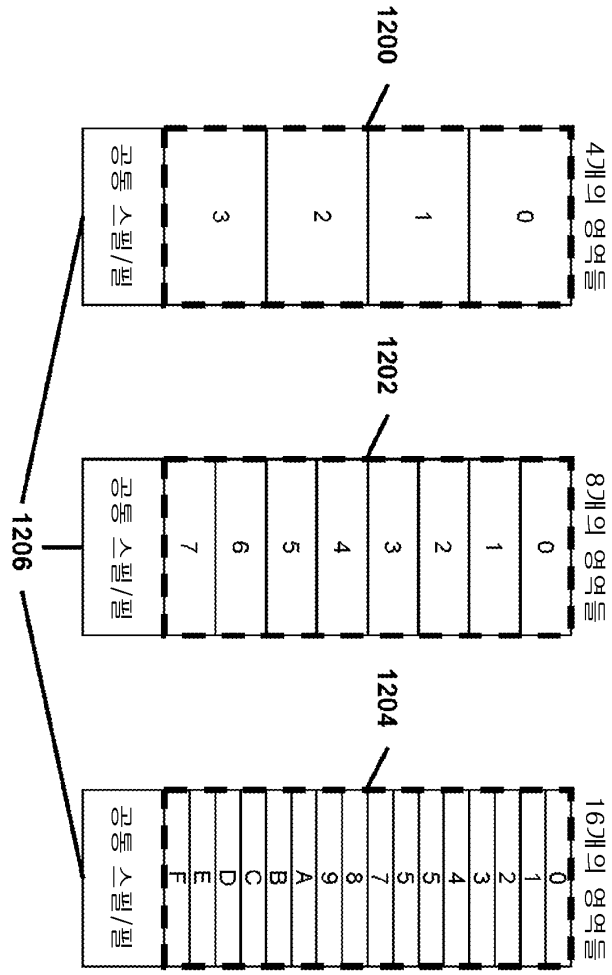
도면10


라우팅 케이블 1004
1000
버퍼 출력
버퍼 입력
1002
910
라우팅 유닛
910
라우팅 유닛
910
라우팅 유닛
910
라우팅 유닛
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력
입출력

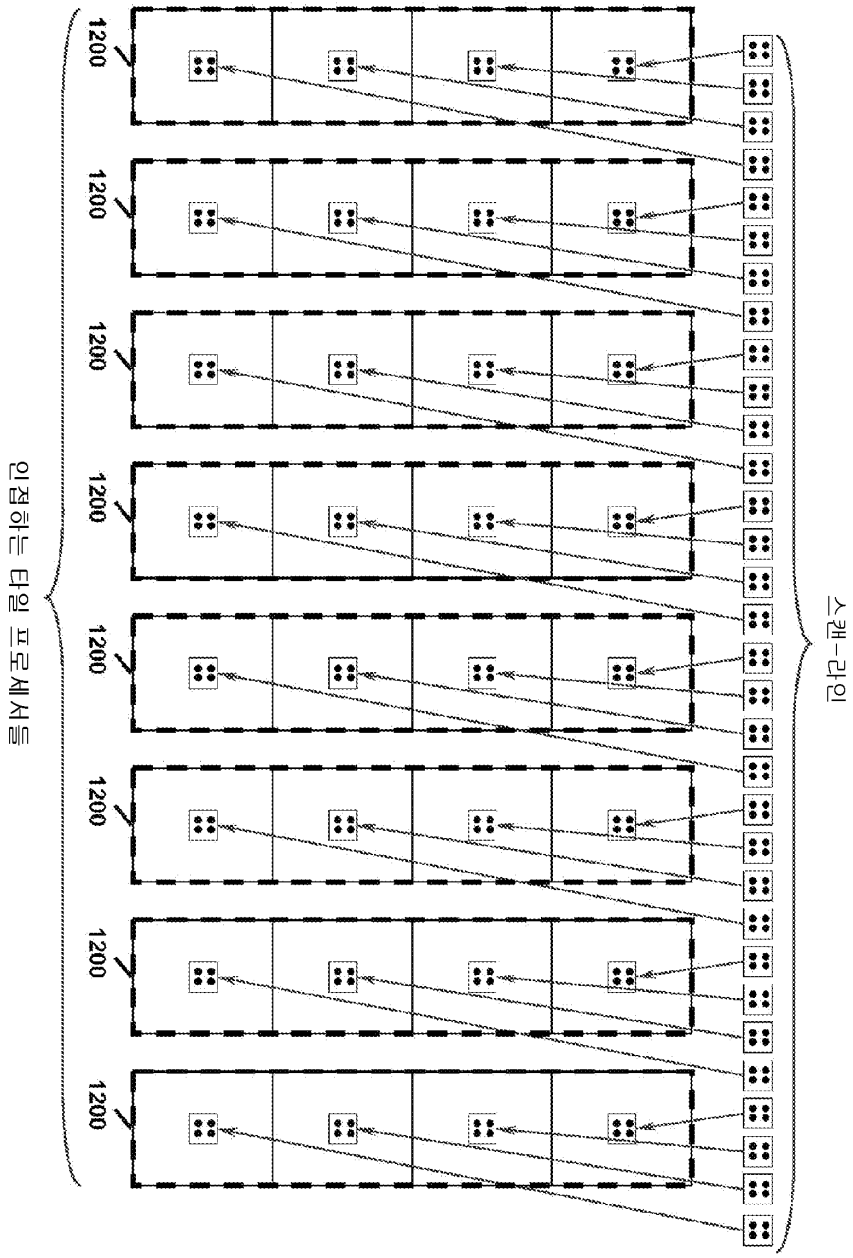
도면11


1120
시스템 인터페이스
글로벌 테이블 RAM
1118
L5B
L5
L4: 4x
L3B: 16x
L3: 16x
L2: 64x
L1 버퍼 (L1B): 256x 1000, 1002
L1: 256x 910
1116
1114
1112
1110
1108
1106
1104
1102
1100
로컬: 1024 x 910(라우팅 유닛(910)의1024개의 인스턴스들)
1024개의 로컬 그룹들(4096개의 타일 프로세서들)

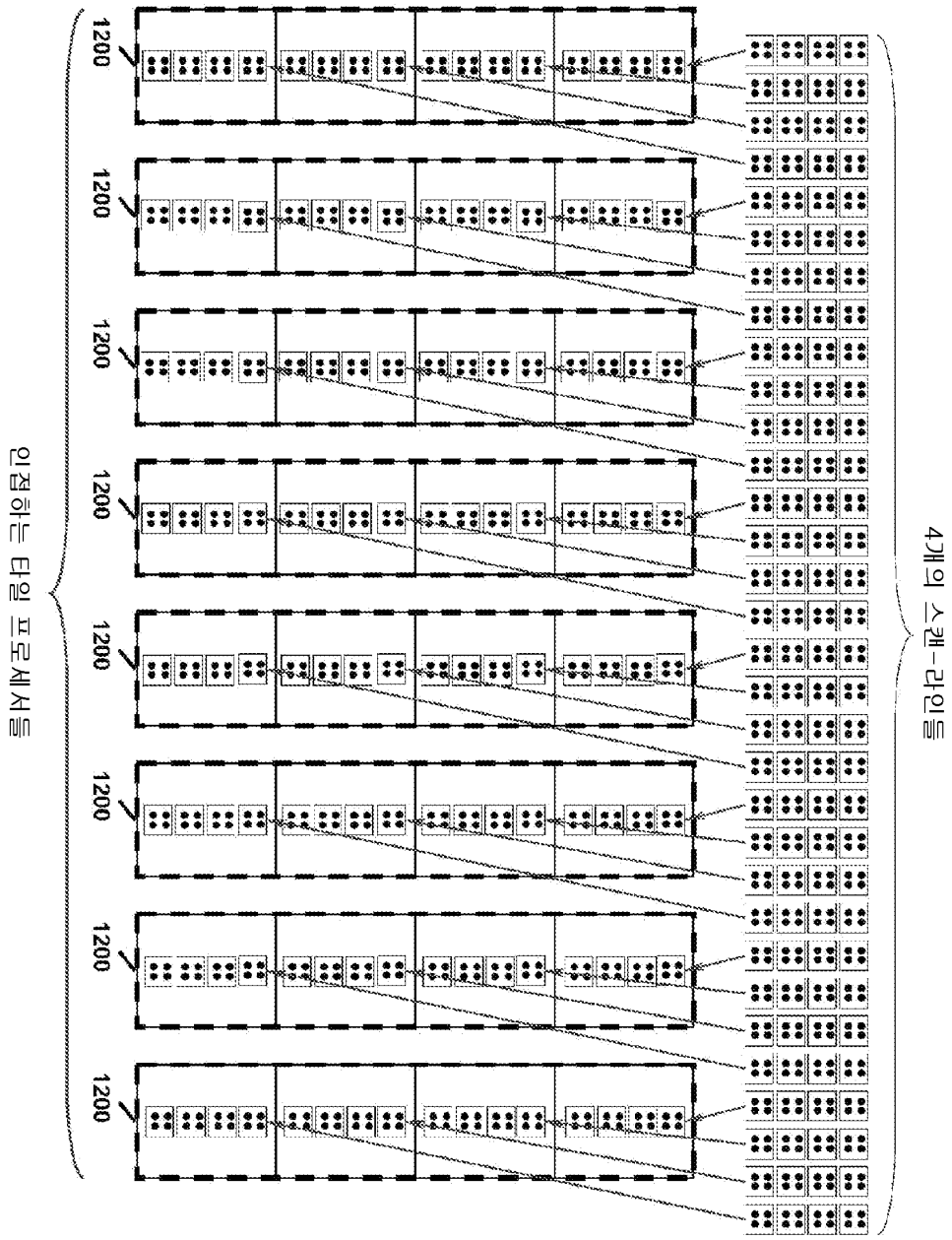
도면12


1200
4개의 영역들
0
1
2
3
공통 스필/필
1202
8개의 영역들
0
1
2
3
4
5
6
7
공통 스필/필
1204
16개의 영역들
0
1
2
3
4
5
5
7
8
9
A
B
C
D
E
F
공통 스필/필
1206

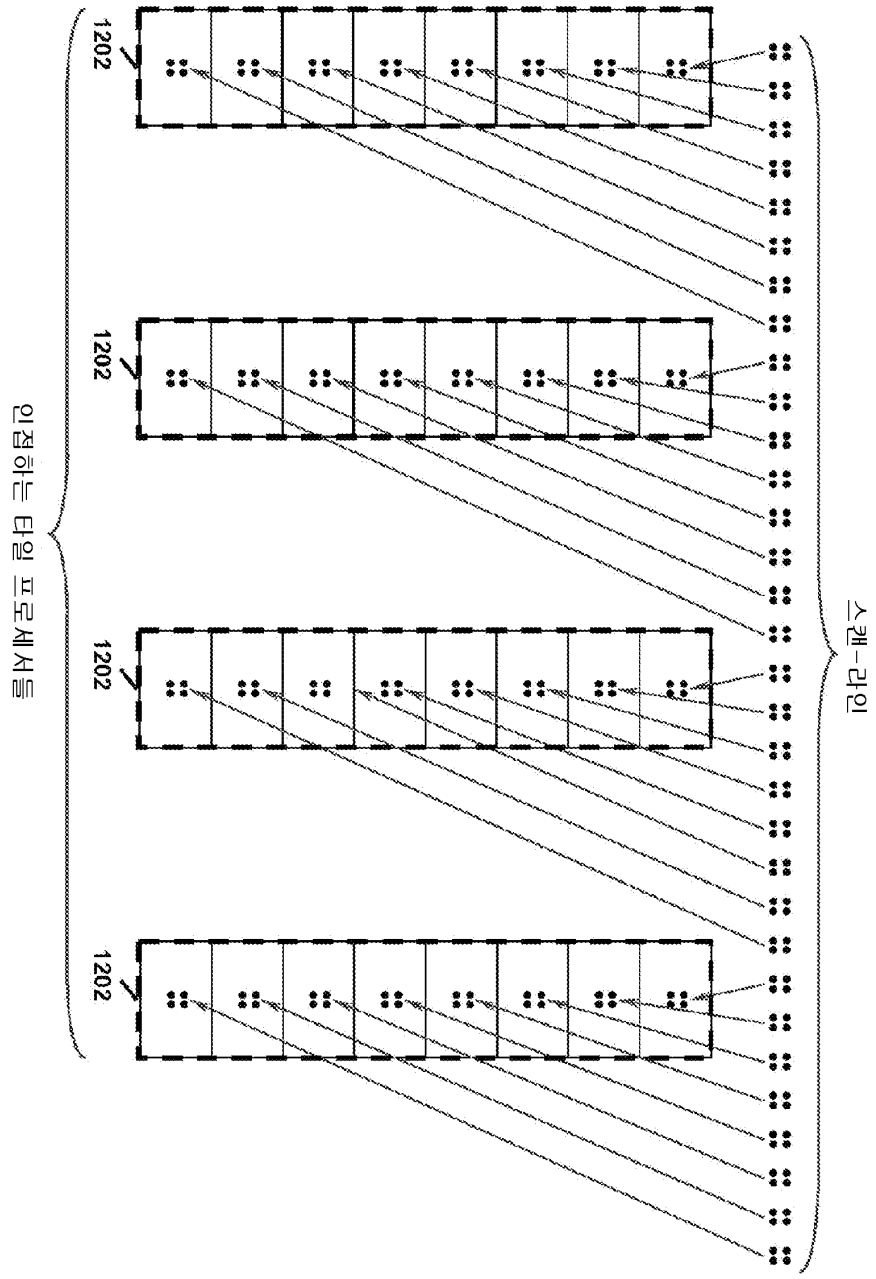
도면13a



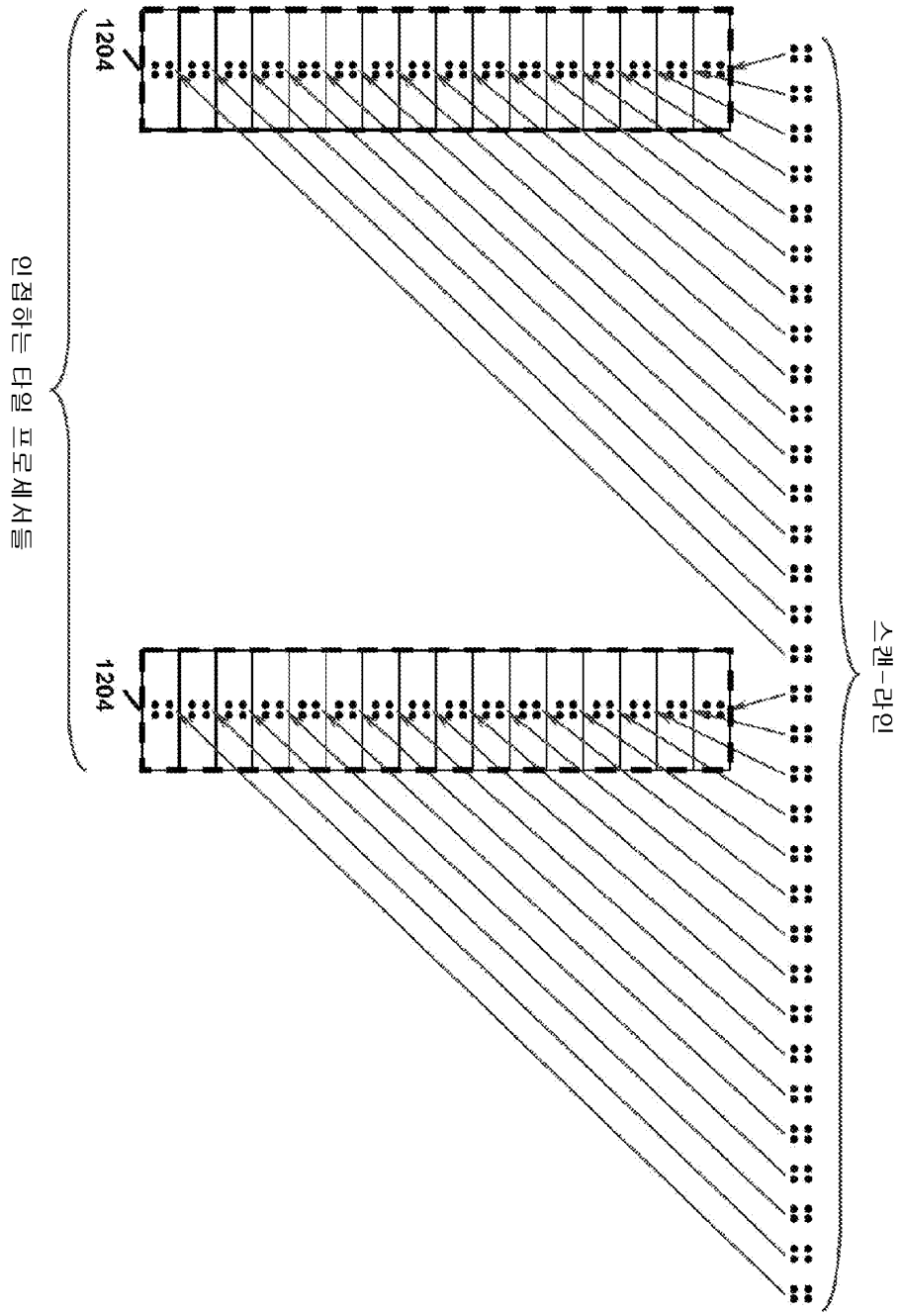
도면13b


1200
1200
1200
1200
1200
1200
1200
1200
인접하는 타일 프로세서들
4개의 스캔-라인들

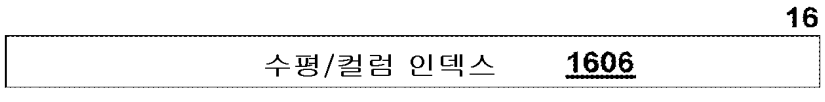
도면14



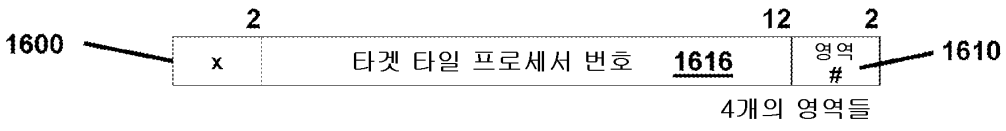
도면15



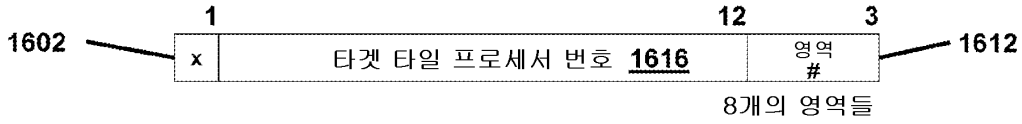
도면16a



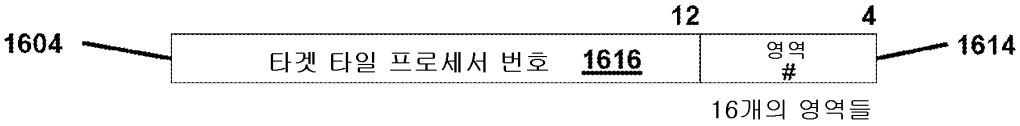
도면16b



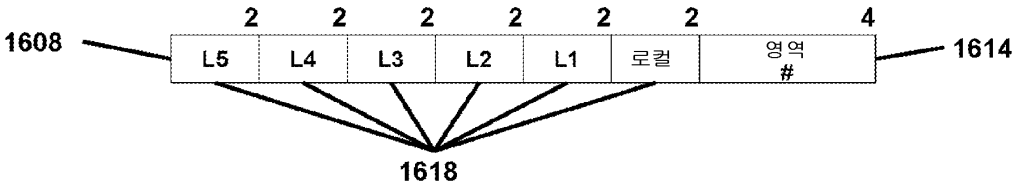
도면16c



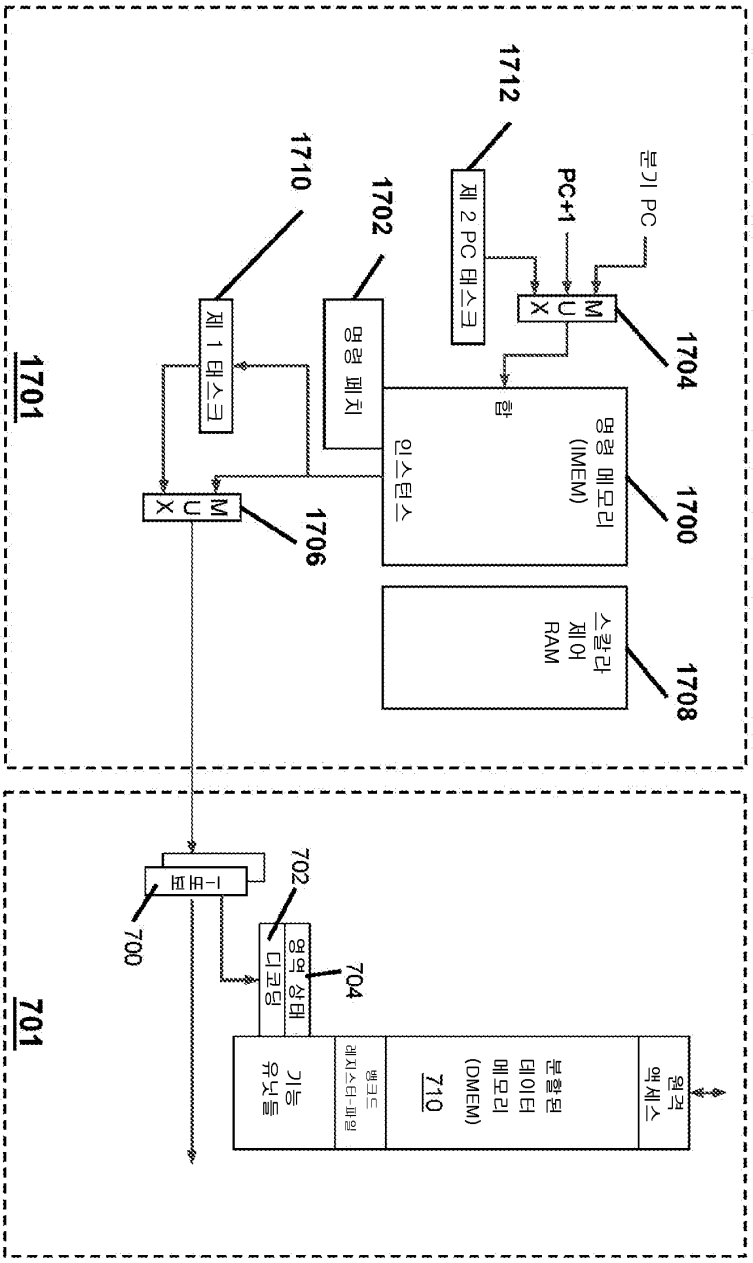
도면16d



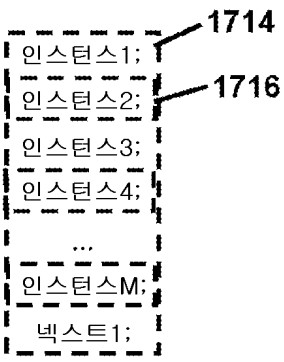
도면16e



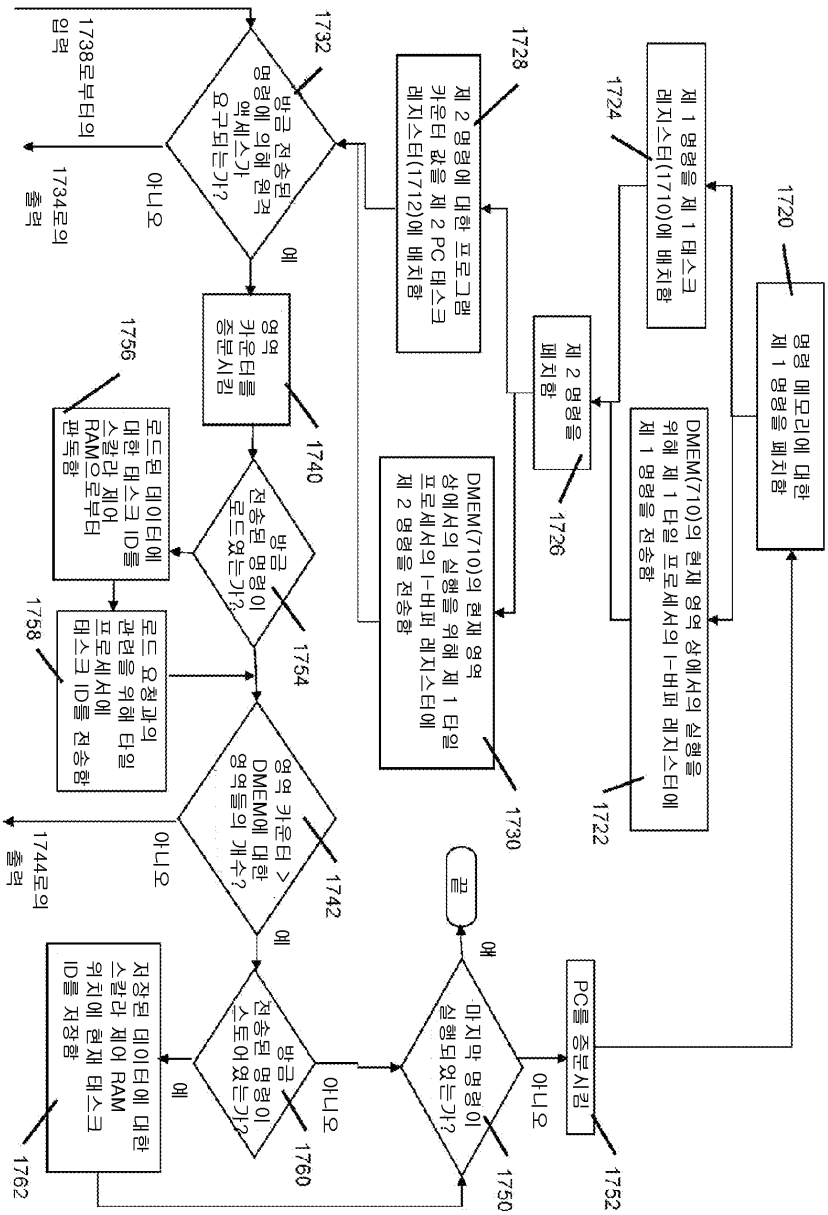
도면17a


1701
1712
제 2 PC 태스크
PC+1
분기 PC
M
U
X
1704
1710
제 1 태스크
1702
명령 페치
합
명령 메모리
(IMEM)
1700
인스턴스
M
U
X
1706
스칼라
제어
RAM
1708
701
1-또는
700
702
영역 상태
디코딩
704
기능
유닛들
뱅크드
레지스터-파일
분할된
데이터
메모리
(DMEM)
710
원격
액세스

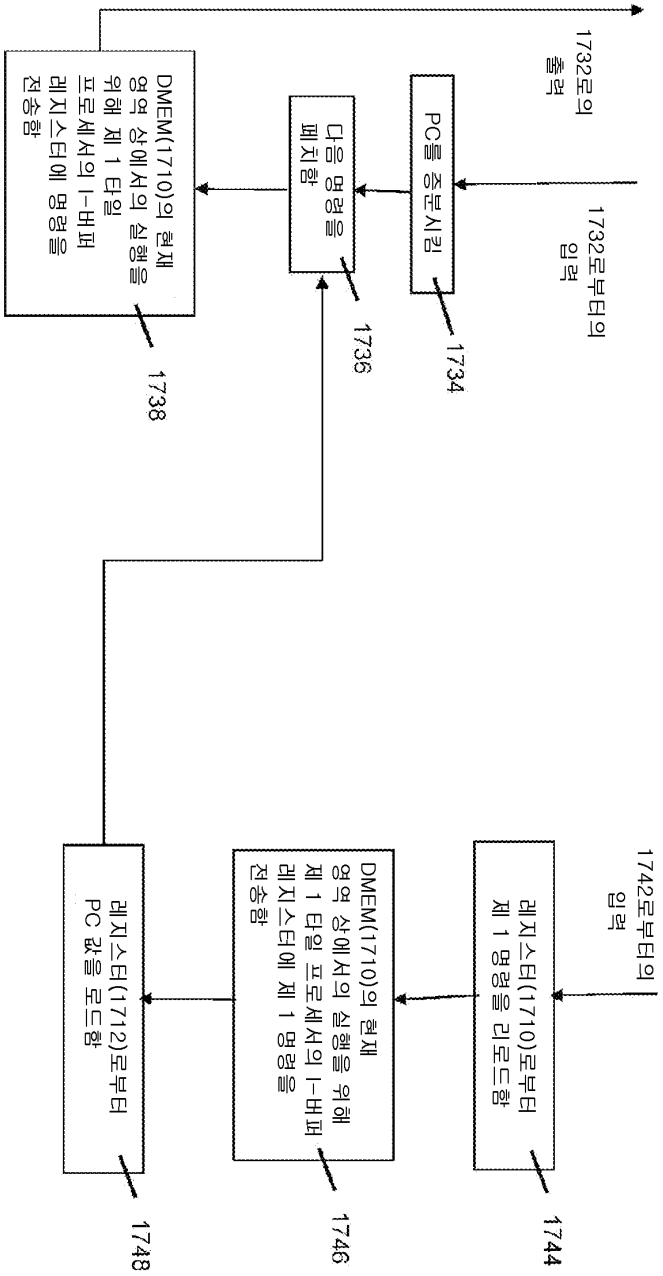
도면17b


1714
1716
인스턴스1;
인스턴스2;
인스턴스3;
인스턴스4;
...
인스턴스M;
넥스트1;

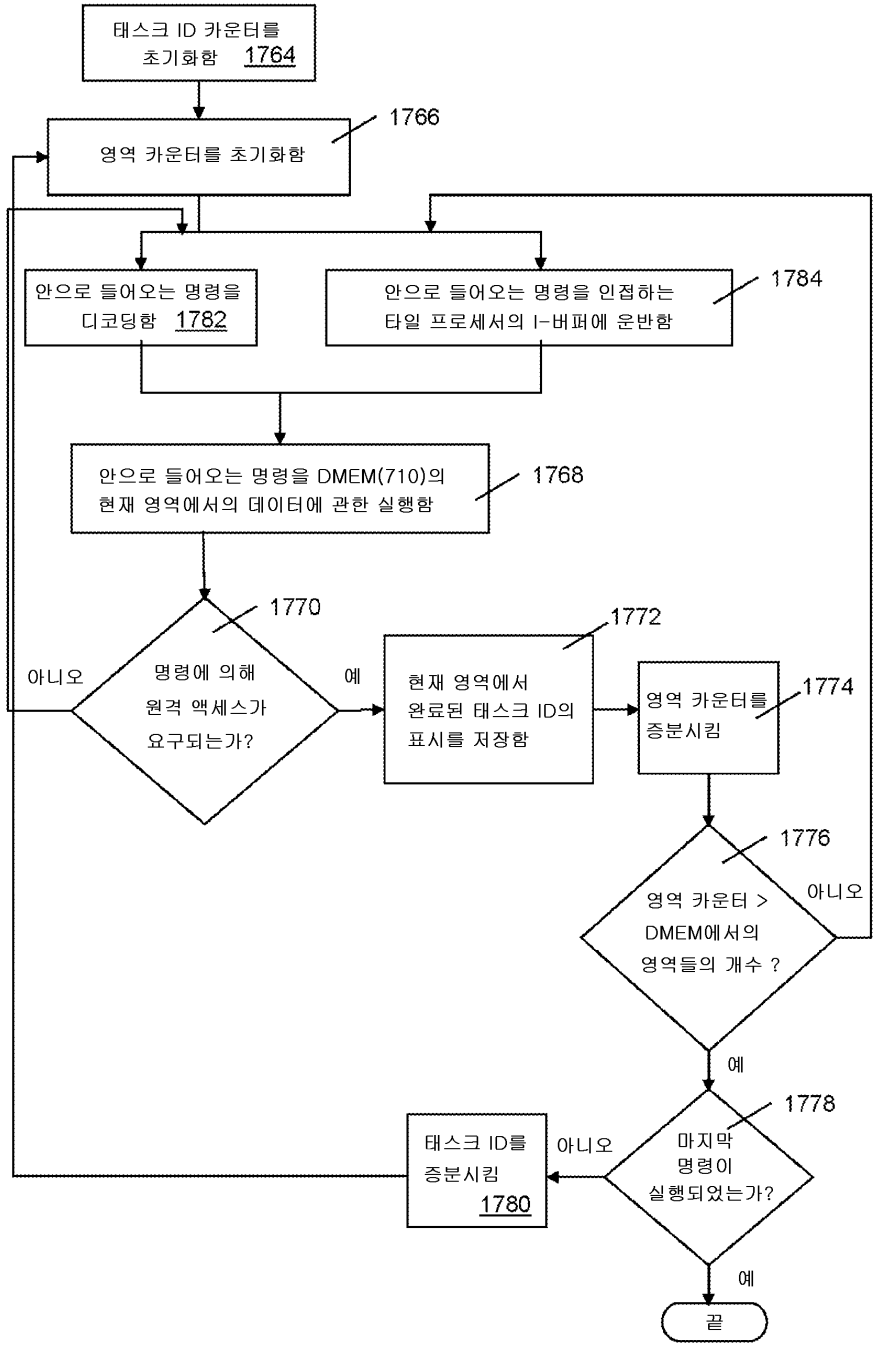
도면17c


1720
명령 메모리에 대한 제 1 명령을 페치함
1724
제 1 명령을 제 1 태스크 레지스터(1710)에 배치함
1722
DMEM(710)의 현재 영역 상에서의 실행을 위해 제 1 타일 프로세서의 I-버퍼 레지스터에 제 1 명령을 전송함
1726
제 2 명령을 페치함
1728
제 2 명령에 대한 프로그램 카운터 값을 제 2 PC 태스크 레지스터(712)에 배치함
1730
DMEM(710)의 현재 영역 상에서의 실행을 위해 제 1 타일 프로세서의 I-버퍼 레지스터에 제 2 명령을 전송함
1732
방금 전송된 명령에 의해 원격 엑세스가 요구되는가?
예
아니오
1734로의 출력
1738로부터의 입력
1740
영역 카운터를 증분시킴
1742
영역 카운터 > DMEM에 대한 영역들의 개수?
예
아니오
1744로의 출력
1750
마지막 명령이 실행되었는가?
예
끝
아니오
1752
PC를 증분시킴
1754
방금 전송된 명령이 로드였는가?
예
아니오
1756
로드된 데이터에 대한 태스크 ID를 스칼라 제어 RAM으로부터 판독함
1758
로드 요청과의 관련을 위해 타일 프로세서에 태스크 ID를 전송함
1760
방금 전송된 명령이 스토어였는가?
예
아니오
1762
저장된 데이터에 대한 스칼라 제어 RAM 위치에 현재 태스크 ID를 저장함

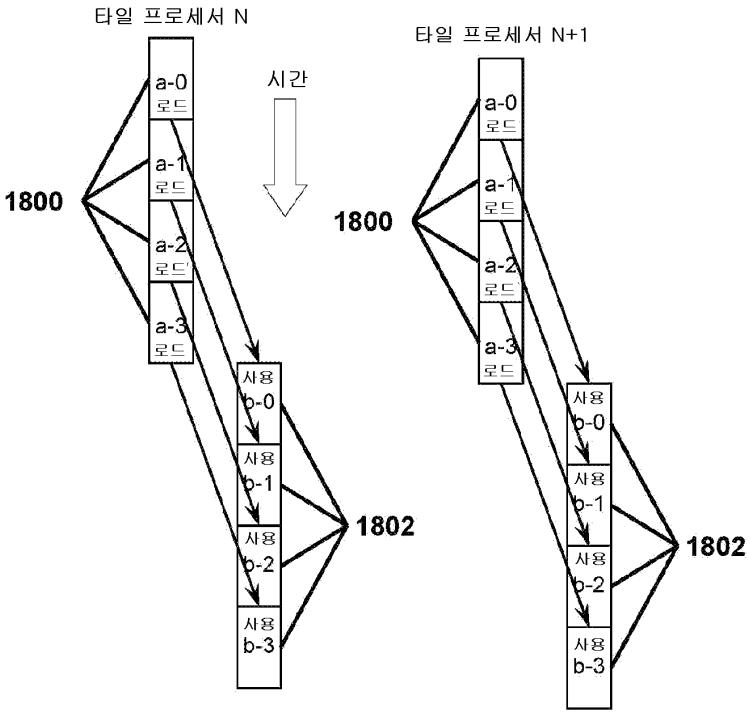
도면17d


1732로의 출력
1732로부터의 입력
PC를 증분시킴
1734
다음 명령을 페치함
1736
DMEM(1710)의 현재 영역 상에서의 실행을 위해 제 1 타일 프로세서의 I-버퍼 레지스터에 명령을 전송함
1738
1742로부터의 입력
레지스터(1710)로부터 제 1 명령을 리로드함
1744
DMEM(1710)의 현재 영역 상에서의 실행을 위해 제 1 타일 프로세서의 I-버퍼 레지스터에 제 1 명령을 전송함
1746
레지스터(1712)로부터 PC 값을 로드함
1748

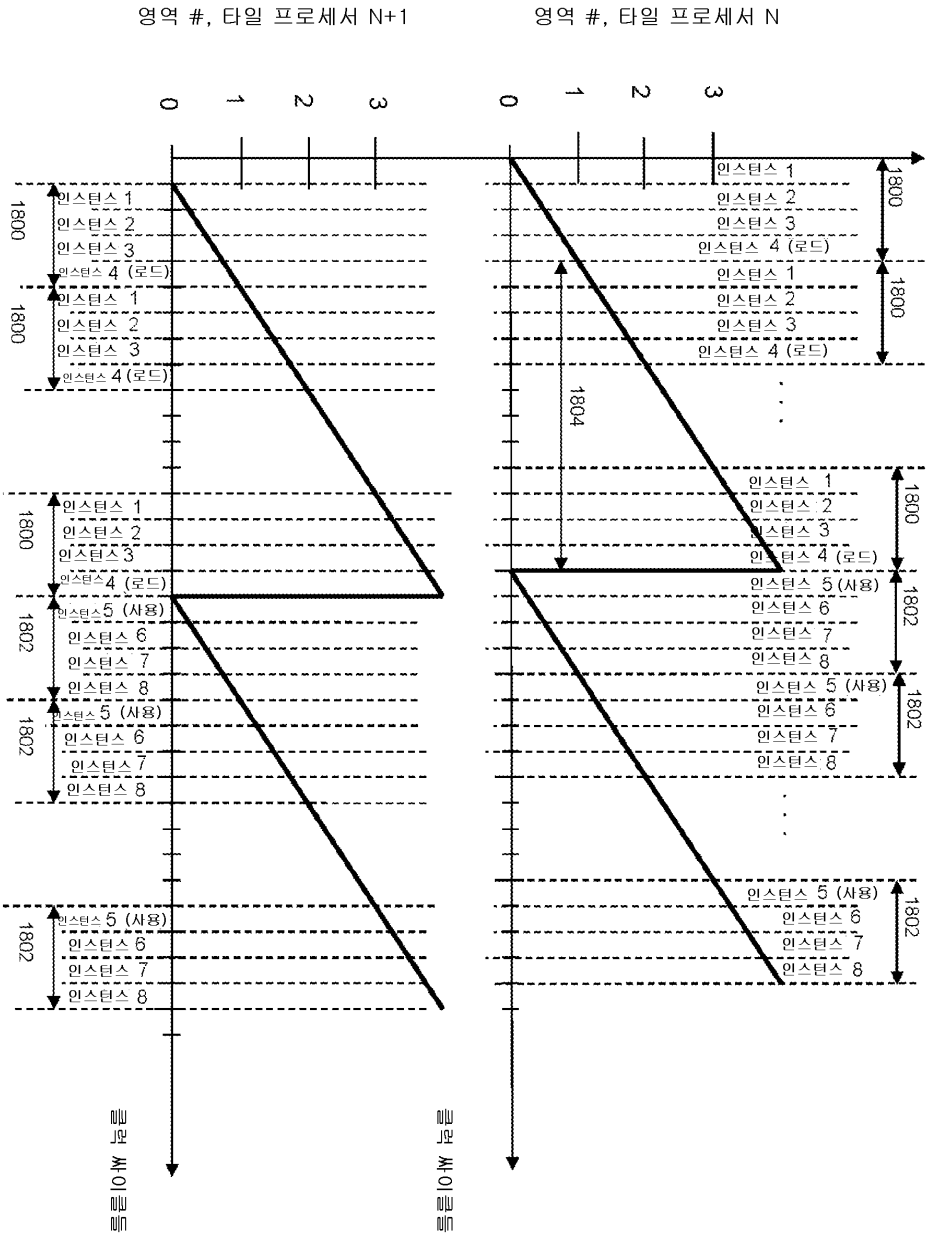
도면17e


태스크 ID 카운터를 초기화함 1764
1766
영역 카운터를 초기화함
안으로 들어오는 명령을 디코딩함 1782
안으로 들어오는 명령을 인접하는 타일 프로세서의 I-버퍼에 운반함
1784
안으로 들어오는 명령을 DMEM(710)의 현재 영역에서의 데이터에 관한 실행함
1768
1770
명령에 의해 원격 액세스가 요구되는가?
아니오
예
1772
현재 영역에서 완료된 태스크 ID의 표시를 저장함
영역 카운터를 증분시킴
1774
1776
영역 카운터 > DMEM에서의 영역들의 개수 ?
아니오
예
1778
마지막 명령이 실행되었는가?
아니오
태스크 ID를 증분시킴 1780
예
끝

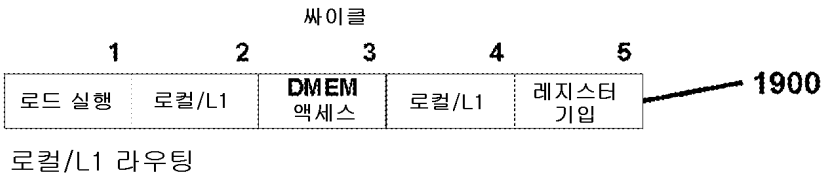
도면18a


타일 프로세서 N
타일 프로세서 N+1
시간
1800
1800
a-0 로드
a-1 로드
a-2 로드
a-3 로드
사용 b-0
사용 b-1
사용 b-2
사용 b-3
1802
1802

도면18b

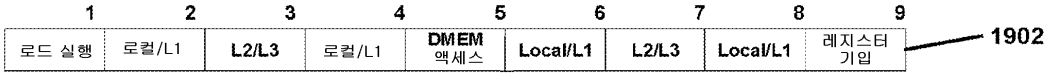


도면19a



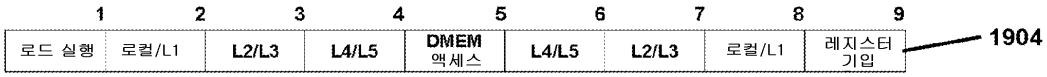
로컬/L1 라우팅

도면19b



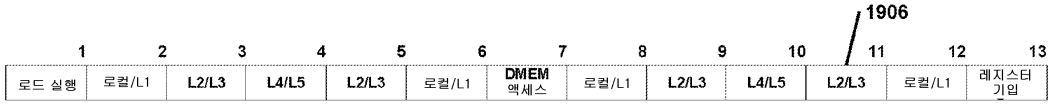
L2/L3 라우팅

도면19c



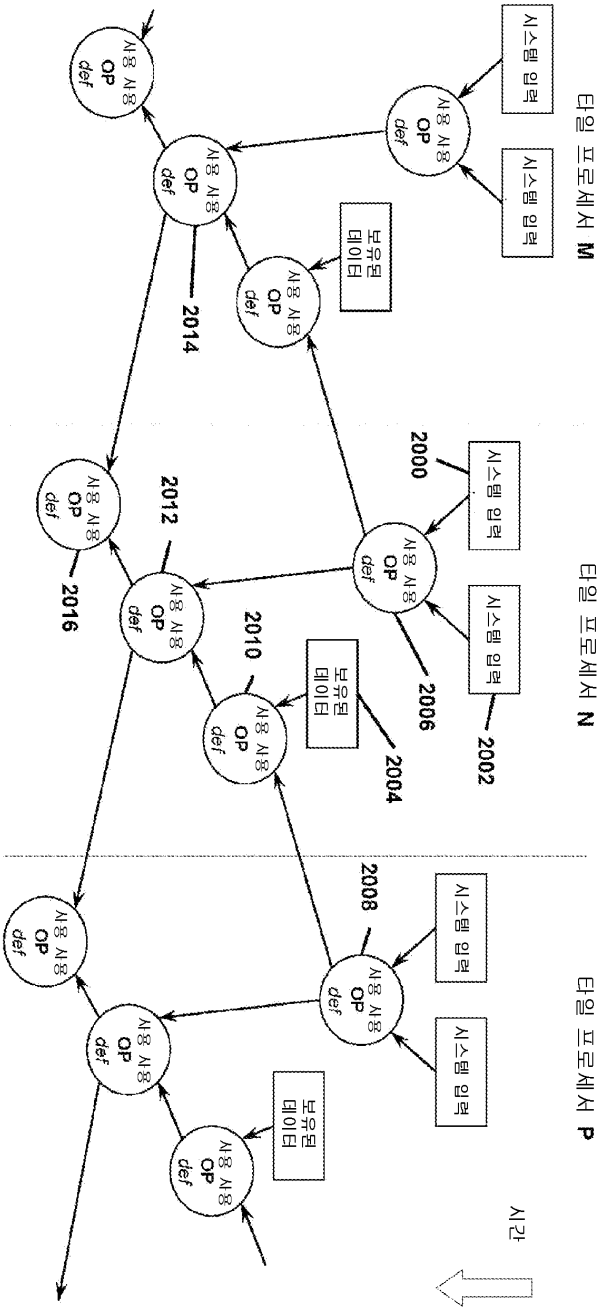
테이블 액세스

도면19d

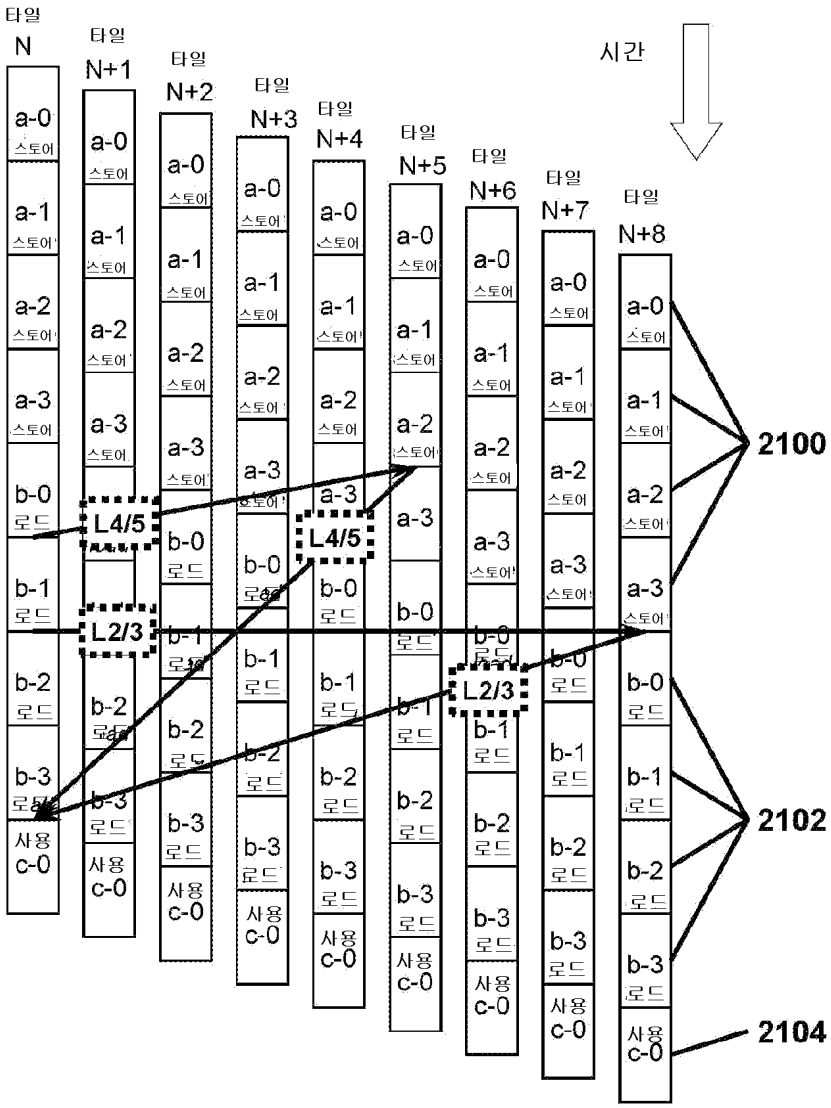


L4/L5 라우팅

도면20



도면21


타일 N
타일 N+1
타일 N+2
타일 N+3
타일 N+4
타일 N+5
타일 N+6
타일 N+7
타일 N+8
시간
a-0 스토어
a-1 스토어
a-2 스토어
a-3 스토어
b-0 로드
b-1 로드
b-2 로드
b-3 로드
사용 c-0
L4/5
L2/3
2100
2102
2104