(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2019년03월22일
(11) 등록번호 10-1961396
(24) 등록일자 2019년03월18일

(51) 국제특허분류(Int. Cl.)
G06F 9/38 (2006.01) *G06F 9/46* (2006.01)
(21) 출원번호 10-2013-7017356
(22) 출원일자(국제) 2011년12월09일
심사청구일자 2016년12월08일
(85) 번역문제출일자 2013년07월02일
(65) 공개번호 10-2013-0127480
(43) 공개일자 2013년11월22일
(86) 국제출원번호 PCT/US2011/064172
(87) 국제공개번호 WO 2012/082557
국제공개일자 2012년06월21일
(30) 우선권주장
13/287,418 2011년11월02일 미국(US)
61/423,465 2010년12월15일 미국(US)
(56) 선행기술조사문헌
Long Chen et al, "Dynamic Load Balancing on Single- and Multi-GPU Systems", 2010 IEEE International Symposium on Parallel & Distributed Processing(2010.04.)*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
어드밴스드 마이크로 디바이시즈, 인코포레이티드
미국 캘리포니아 95054 산타 클라라 어거스틴 드라이브 2485
(72) 발명자
샌더 벤자민 토마스
미국 텍사스 78735 오스틴 메디신 크리크 5701
호우스톤 마이클
미국 캘리포니아 95014 쿠퍼티노 콜롬버스 애비뉴 21330
(뒷면에 계속)
(74) 대리인
박장원

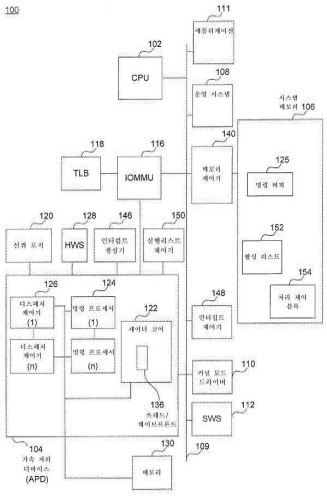
전체 청구항 수 : 총 22 항 심사관 : 정성훈

(54) 발명의 명칭 **이종 처리 디바이스의 동적 작업 분할**

(57) 요 약

본 발명은 이종 처리 디바이스에서 작업부하를 균형 잡는 방법, 시스템, 및 제조 물품에 관한 것이다. 본 방법은 상이한 유형의 프로세서와 연관된 디큐잉 개체에 의해 하나의 유형의 프로세서의 메모리 저장매체에 액세스하는 단계, 상기 상이한 유형의 프로세서에 의해 처리될 수 있는 메모리에 있는 복수의 작업으로부터 작업을 식별하는 단계, 상기 메모리 저장매체에 액세스할 수 있는 복수의 디큐잉 개체를 동기화하는 단계, 및 메모리 저장매체로부터 작업을 디큐잉하는 단계를 포함한다.

대 표 도 - 도1a



(72) 발명자

체웅 뉴톤

미국 캘리포니아 94088 서니베일 원 에이엠디 플레이스

로웨리 케이스

미국 워싱톤 98011 보셀 노스이스트 197 스트리트 10910

명 세 서

청구범위

청구항 1

이종 처리 디바이스(heterogeneous processing device)에서 작업 부하를 균형 잡는 방법으로서,

제1 유형의 하나 이상의 제1 프로세서들 및 제2 유형의 하나 이상의 제2 프로세서들에 통신 가능하게 연결되는 공유 큐 내에 복수의 작업들을 저장하는 단계 - 상기 복수의 작업들 각각은 상기 하나 이상의 제1 프로세서들 상에서의 실행을 위한 마이크로 코드에 대한 제1 포인터 및 상기 하나 이상의 제2 프로세서들 상에서의 실행을 위한 컴파일된 코드에 대한 제2 포인터를 포함하며,

상기 하나 이상의 제1 프로세서들 각각은:

제1 디큐잉 모듈(dequeuing module), 및

하나 이상의 제1 프로세싱 코어들을 포함하고,

상기 하나 이상의 제2 프로세서들 각각은:

동기화 모듈,

제2 디큐잉 모듈, 및

하나 이상의 제2 프로세싱 코어들을 포함하며 - 와;

상기 하나 이상의 제2 프로세서들 중 적어도 하나의 동기화 모듈에 의해, 동기화를 위한 원자 동작(atomic operation)을 이용하여 상기 하나 이상의 제1 프로세서들 각각의 제1 디큐잉 모듈 및 상기 하나 이상의 제2 프로세서들 각각의 제2 디큐잉 모듈에 의해 상기 공유 큐로의 액세스를 동기화하는 단계와;

상기 하나 이상의 제2 프로세서들 중 적어도 하나의 제2 디큐잉 모듈에 의해, 상기 복수의 작업들 중 특정 작업을 판독하는 단계와;

상기 하나 이상의 제2 프로세서들 중 상기 적어도 하나의 제2 디큐잉 모듈에 의해, 상기 특정 작업이 상기 하나 이상의 제2 프로세서들 상에서 실행하기에 적합한 조건에서 상기 특정 작업을 상기 공유 큐로부터 디큐잉하는 단계와; 그리고

상기 하나 이상의 제2 프로세서들 중 상기 적어도 하나의 제2 디큐잉 모듈에 의해, 상기 제2 포인터에 의해 지시된 상기 컴파일된 코드를 실행하도록 상기 하나 이상의 제2 프로세싱 코어들에 명령하는 단계를 포함하고,

상기 제2 포인터는 상기 특정 작업의 포인터인 것을 특징으로 하는

작업 부하를 균형 잡는 방법.

청구항 2

제1항에 있어서,

상기 하나 이상의 제2 프로세서들은 중앙 처리 유닛(CPU: central processing unit)를 포함하고, 그리고 상기 하나 이상의 제1 프로세서들은 가속 처리 디바이스(APD: accelerated processing device)를 포함하는 것을 특징으로 하는

작업 부하를 균형 잡는 방법.

청구항 3

제2항에 있어서,

상기 하나 이상의 제1 프로세서들의 상기 제1 디큐잉 모듈들은 하드웨어 디바이스들인 것을 특징으로 하는

작업 부하를 균형 잡는 방법.

청구항 4

제2항에 있어서,

상기 하나 이상의 제2 프로세서들의 상기 제2 디큐잉 모듈들은 소프트웨어 모듈들인 것을 특징으로 하는

작업 부하를 균형 잡는 방법.

청구항 5

제1항에 있어서,

상기 특정 작업은 상기 공유 큐에 저장된 작업 파라미터에 기초하여 상기 하나 이상의 제2 프로세서들 상에서 실행하기에 적합하다고 결정되는 것을 특징으로 하는

작업 부하를 균형 잡는 방법.

청구항 6

제1항에 있어서,

상기 하나 이상의 제2 프로세서들의 상기 제2 디큐잉 모듈들이 디큐잉하는 작업들의 수는 상기 하나 이상의 제2 프로세서들의 유형에 기초하는 것을 특징으로 하는

작업 부하를 균형 잡는 방법.

청구항 7

제1항에 있어서,

상기 하나 이상의 제1 프로세서들 중 적어도 하나의 제1 디큐잉 모듈에 의해, 상기 복수의 작업들 중 다른 특정 작업을 판독하는 단계와;

상기 적어도 하나의 제1 디큐잉 모듈에 의해, 상기 다른 특정 작업이 상기 하나 이상의 제1 프로세서들 상에서 실행하기에 적합한 조건에서 상기 다른 특정 작업을 상기 공유 큐로부터 디큐잉하는 단계와; 그리고

상기 적어도 하나의 제1 디큐잉 모듈에 의해, 상기 제1 포인터에 의해 지시된 상기 마이크로 코드를 실행하도록 상기 하나 이상의 제1 프로세싱 코어들에 명령하는 단계를 더 포함하는 것을 특징으로 하는

작업 부하를 균형 잡는 방법.

청구항 8

제1항에 있어서,

상기 복수의 작업들 각각은, 미리 결정된 이벤트의 발생 후에 실행 가능하게 되는 종속 정보를 포함하는 중간 언어 표현에 대한 제3 포인터를 더 포함하는 것을 특징으로 하는

작업 부하를 균형 잡는 방법.

청구항 9

이종 처리 디바이스들에서 작업 부하를 균형 잡는 시스템으로서,

하나 이상의 제1 유형의 제1 프로세서들과;

하나 이상의 제2 유형의 제 2 프로세서들과; 그리고

상기 하나 이상의 제1 프로세서들 및 상기 하나 이상의 제2 프로세서들에 통신 가능하게 연결되는 공유 큐를 포함하고,

상기 하나 이상의 제1 프로세서들 각각은,

제1 디큐잉 모듈, 및

하나 이상의 제1 프로세싱 코어들을 포함하고,

상기 하나 이상의 제2 프로세서들 각각은,

동기화 모듈,

제2 디큐잉 모듈, 및

하나 이상의 제2 프로세싱 코어들을 포함하며,

상기 공유 큐는 복수의 작업들을 저장하고, 상기 복수의 작업들 각각은 상기 하나 이상의 제1 프로세서들 상에서의 실행을 위한 마이크로 코드에 대한 제1 포인터 및 상기 하나 이상의 제2 프로세서들 상에서의 실행을 위한 컴파일된 코드에 대한 제2 포인터를 포함하고,

상기 하나 이상의 제2 프로세서들의 각 동기화 모듈은, 상기 공유 큐로부터 특정 작업을 제거하기 전에 동기화를 위한 원자 동작을 이용하여, 상기 하나 이상의 제1 프로세서들 각각의 제1 디큐잉 모듈 및 상기 하나 이상의 제2 프로세서들 각각의 제2 디큐잉 모듈에 의해 상기 공유 큐로의 액세스를 동기화하며,

상기 하나 이상의 제2 프로세서들의 각 제2 디큐잉 모듈은,

상기 복수의 작업들 중 상기 특정 작업을 판독하고,

상기 특정 작업이 상기 하나 이상의 제2 프로세서들 상에서 실행하기에 적합한 조건에서 상기 특정 작업을 상기 공유 큐로부터 디큐잉하며, 그리고

상기 제2 포인터에 의해 지시된 상기 컴파일된 코드를 실행하도록 상기 하나 이상의 제2 프로세싱 코어들에 명령하며,

상기 제2 포인터는 상기 특정 작업의 포인터인 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 10

제9항에 있어서,

상기 하나 이상의 제2 프로세서들은 중앙 처리 유닛(CPU)를 포함하고, 그리고 상기 하나 이상의 제1 프로세서들은 가속 처리 디바이스(APD)를 포함하는 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 11

제9항에 있어서,

상기 하나 이상의 제1 프로세서들의 상기 제1 디큐잉 모듈들은 하드웨어 디바이스들인 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 12

제9항에 있어서,

상기 특정 작업은 상기 공유 큐에 저장된 작업 파라미터에 기초하여 상기 하나 이상의 제2 프로세서들 상에서 실행하기에 적합하다고 결정되는 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 13

제9항에 있어서,

적어도 하나의 상기 제2 디큐잉 모듈이 디큐잉하는 작업들의 수는 상기 하나 이상의 제2 프로세서들의 유형에 기초하는 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 14

제9항에 있어서,

상기 하나 이상의 제1 프로세서들의 각 제1 디큐잉 모듈은,

상기 복수의 작업들 중 다른 특정 작업을 판독하고,

상기 다른 특정 작업이 상기 하나 이상의 제1 프로세서들 상에서 실행하기에 적합한 조건에서 상기 다른 특정 작업을 상기 공유 큐로부터 디큐잉하며, 그리고

상기 제1 포인터에 의해 지시된 상기 마이크로 코드를 실행하도록 상기 하나 이상의 제1 프로세싱 코어들에 명령하는 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 15

제9항에 있어서,

상기 복수의 작업들 각각은, 미리 결정된 이벤트의 발생 후에 실행 가능하게 되는 종속 정보를 포함하는 중간 언어 표현에 대한 제3 포인터를 더 포함하는 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 16

인스트럭션들이 기록된 비 일시적인 컴퓨터 판독가능한 매체로서, 상기 인스트럭션들은, 컴퓨팅 디바이스에 의해 실행될 때, 상기 컴퓨팅 디바이스로 하여금,

제1 유형의 하나 이상의 제1 프로세서들 및 제2 유형의 하나 이상의 제2 프로세서들에 통신 가능하게 연결되는 공유 큐 내에 복수의 작업들을 저장하는 단계 - 상기 복수의 작업들 각각은 상기 하나 이상의 제1 프로세서들 상에서의 실행을 위한 마이크로 코드에 대한 제1 포인터 및 상기 하나 이상의 제2 프로세서들 상에서의 실행을 위한 컴파일된 코드에 대한 제2 포인터를 포함하며,

상기 하나 이상의 제1 프로세서들 각각은:

제1 디큐잉 모듈, 및

하나 이상의 제1 프로세싱 코어들을 포함하고,

상기 하나 이상의 제2 프로세서들 각각은:

동기화 모듈,

제2 디큐잉 모듈, 및

하나 이상의 제2 프로세싱 코어들을 포함하며 - 와;

적어도 하나의 동기화 모듈에 의해, 상기 하나 이상의 제1 프로세서들 각각의 제1 디큐잉 모듈 및 상기 하나 이상의 제2 프로세서들 각각의 제2 디큐잉 모듈에 의해 상기 공유 큐로의 액세스를 동기화하는 단계와;

상기 하나 이상의 제2 프로세서들 중 적어도 하나의 제2 디큐잉 모듈에 의해, 상기 복수의 작업들 중 특정 작업을 판독하는 단계와;

상기 하나 이상의 제2 프로세서들 중 상기 적어도 하나의 제2 디큐잉 모듈에 의해, 상기 특정 작업이 상기 하나 이상의 제2 프로세서들 상에서 실행하기에 적합한 조건에서 상기 특정 작업을 상기 공유 큐로부터 디큐잉하는 단계와; 그리고

상기 하나 이상의 제2 프로세서들 중 상기 적어도 하나의 제2 디큐잉 모듈에 의해, 상기 제2 포인터에 의해 지시된 상기 컴파일된 코드를 실행하도록 상기 하나 이상의 제2 프로세싱 코어들에 명령하는 단계 - 상기 제2 포인터는 상기 특정 작업의 포인터이며 - 를 포함하는 방법을 수행하게 하는 것을 특징으로 하는

비 일시적인 컴퓨터 판독가능한 매체.

청구항 17

제16항에 있어서,

상기 하나 이상의 제2 프로세서들은 중앙 처리 유닛(CPU)를 포함하고, 그리고 상기 하나 이상의 제1 프로세서들은 가속 처리 디바이스(APD)를 포함하는 것을 특징으로 하는

비 일시적인 컴퓨터 판독가능한 매체.

청구항 18

제16항에 있어서,

상기 하나 이상의 제1 프로세서들의 상기 제1 디큐잉 모듈들은 하드웨어 디바이스들인 것을 특징으로 하는

비 일시적인 컴퓨터 판독가능한 매체.

청구항 19

제16항에 있어서,

상기 하나 이상의 제1 프로세서들 중 적어도 하나의 제1 디큐잉 모듈에 의해, 상기 복수의 작업들 중 다른 특정 작업을 판독하는 단계와;

상기 적어도 하나의 제1 디큐잉 모듈에 의해, 상기 다른 특정 작업이 상기 하나 이상의 제1 프로세서들 상에서 실행하기에 적합한 조건에서 상기 다른 특정 작업을 상기 공유 큐로부터 디큐잉하는 단계와; 그리고

상기 적어도 하나의 제1 디큐잉 모듈에 의해, 상기 제1 포인터에 의해 지시된 상기 마이크로 코드를 실행하도록 상기 하나 이상의 제1 프로세싱 코어들에 명령하는 단계를 더 포함하는 것을 특징으로 하는

비 일시적인 컴퓨터 판독가능한 매체.

청구항 20

제16항에 있어서,

상기 복수의 작업들 각각은, 미리 결정된 이벤트의 발생 후에 실행 가능하게 되는 종속 정보를 포함하는 중간 언어 표현에 대한 제3 포인터를 더 포함하는 것을 특징으로 하는

비 일시적인 컴퓨터 판독가능한 매체.

청구항 21

이종 처리 디바이스들에서 작업 부하를 균형 잡는 시스템으로서,

하나 이상의 제1 유형의 제1 프로세서들과;

하나 이상의 제2 유형의 제 2 프로세서들과; 그리고

상기 하나 이상의 제1 프로세서들 및 상기 하나 이상의 제2 프로세서들에 통신 가능하게 연결되는 공유 큐를 포함하고,

상기 하나 이상의 제1 프로세서들 각각은,

제1 디큐잉 모듈, 및

하나 이상의 제1 프로세싱 코어들을 포함하고,

상기 하나 이상의 제2 프로세서들 각각은,

동기화 모듈,

제2 디큐잉 모듈, 및

하나 이상의 제2 프로세싱 코어들을 포함하며,

상기 공유 큐는 복수의 작업들을 저장하고, 상기 복수의 작업들 각각은 상기 하나 이상의 제1 프로세서들 상에서의 실행을 위한 마이크로 코드에 대한 제1 포인터 및 상기 하나 이상의 제2 프로세서들 상에서의 실행을 위한 컴파일된 코드에 대한 제2 포인터를 포함하고,

상기 하나 이상의 제2 프로세서들의 각 동기화 모듈은, 상기 공유 큐로부터 특정 작업을 제거하기 전에 동기화를 위한 원자 동작을 이용하여, 상기 하나 이상의 제1 프로세서들 각각의 제1 디큐잉 모듈 및 상기 하나 이상의 제2 프로세서들 각각의 제2 디큐잉 모듈에 의해 상기 공유 큐로의 액세스를 동기화하며,

상기 하나 이상의 제1 프로세서들의 각 제1 디큐잉 모듈은,

상기 복수의 작업들 중 특정 작업을 판독하고,

상기 특정 작업이 상기 하나 이상의 제1 프로세서들 상에서 실행하기에 적합한 조건에서 상기 특정 작업을 상기 공유 큐로부터 디큐잉하며, 그리고

상기 제1 포인터에 의해 지시된 상기 마이크로 코드를 실행하도록 상기 하나 이상의 제1 프로세싱 코어들에 명령하며,

상기 제1 포인터는 상기 특정 작업의 포인터인 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 22

제21항에 있어서,

상기 복수의 작업들 각각은, 미리 결정된 이벤트의 발생 후에 실행 가능하게 되는 종속 정보를 포함하는 중간 언어 표현에 대한 제3 포인터를 더 포함하는 것을 특징으로 하는

작업 부하를 균형 잡는 시스템.

청구항 23

삭제

발명의 설명

기 술 분 야

[0001] 본 발명은 일반적으로 컴퓨터 시스템(computer system)에 관한 것이다. 보다 상세하게는, 본 발명은 컴퓨터 시스템 내 연산 성분(computational component)을 단일화하는 아키텍처에 관한 것이다.

배 경 기 술

[0002] 일반적인 연산(computation)에 그래픽 처리 유닛(GPU: graphics processing unit)을 사용하려는 요구가 최근에 단위 전력 및/또는 비용당 GPU의 예시적인 성능으로 인해 훨씬 더 높아지고 있다. GPU의 연산 성능(computational capability)은 일반적으로 대응하는 중앙 처리 유닛(CPU: central processing unit) 플랫폼의 것을 초과하는 율(rate)로 성장하였다. 모바일 컴퓨팅 시장(예를 들어, 노트북, 모바일 스마트폰, 태블릿 등) 및 필요한 지원 서버/기업용 시스템의 폭발적 증가와 연결된 이러한 성장은 원하는 유저 경험의 특정된 품질을 제공하는데 사용되고 있다. 그 결과, 데이터와 병렬로 콘텐츠에 작업부하(workload)를 실행하기 위해 CPU와 GPU를 결합하여 사용하는 것은 볼륨 기술(volume technology)이 되고 있다.

[0003] 그러나, GPU는 전통적으로 주로 그래픽을 가속시키기 위하여 이용가능한 제약된 프로그래밍 환경에서 동작된다. 이들 제약은 GPU가 CPU만큼 풍부한 프로그래밍 에코시스템을 가지지 않는다는 것에 기인한다. 그리하여, 그 사용은 그래픽 및 비디오 애플리케이션 프로그래밍 인터페이스(API: application programming interface)로 처리하는 것에 이미 익숙해진, 대부분 2D와 3D 그래픽 및 일부 선도하는 멀티미디어 애플리케이션으로 제한된다.

[0004] 다수 벤더 지원 OpenCL(상표명)과 DirectCompute(등록상표), 표준 API 및 지원 툴의 도래로, 전통적인 애플리케이션에서 GPU의 제한은 전통적인 그래픽을 넘어 확장되었다. OpenCL 및 DirectCompute가 유망한 시작이라 하더

라도, CPU와 GPU의 조합이 대부분 프로그래밍 작업에 CPU만큼 유동적으로 사용되게 하는 환경 및 에코시스템을 생성하는 것에 많은 장애들이 남아있다.

[0005] 현존하는 컴퓨팅 시스템은 종종 다수의 처리 디바이스를 포함한다. 예를 들어, 일부 컴퓨팅 시스템은 별개의 칩에 CPU와 GPU를 포함하거나(예를 들어, CPU는 마더보드 상에 위치될 수 있고 GPU는 그래픽 카드 상에 위치될 수 있다) 단일 칩 패키지에 CPU와 GPU를 모두 포함한다. 그러나, 이들 두 배열은 전력 소비를 최소화하면서 (i) 별개의 메모리 시스템, (ii) 효과적인 스케줄링, (iii) 처리 사이에 서비스 품질(QoS: quality of service) 보장 제공, (iv) 모델 프로그래밍, 및 (v) 다수의 타깃 인스트럭션 세트 아키텍처(ISA: instruction set architecture)로 컴파일링하는 것과 연관된 상당한 문제를 여전히 포함한다.

[0006] 예를 들어, 이산 칩 배열은 각 프로세서가 메모리에 액세스하기 위한 칩 대 칩 인터페이스를 시스템과 소프트웨어 아키텍처가 이용할 수 있게 한다. 이들 외부 인터페이스(예를 들어, 칩 대 칩)는 이종 프로세서(heterogeneous processor)와 협력하기 위해 메모리 지체와 전력 소비에 부작용을 나타내지만, 별개의 메모리 시스템(즉, 별개의 어드레스 공간)과 드라이버로 관리되는 공유 메모리는 정밀 입도 오프로드(fine grain offload)에 허용가능하지 않는 오버헤드를 생성한다.

[0007] CPU와 GPU가 전통적으로 상이한 작업을 수행하였지만, 많은 유형의 작업부하가 CPU 또는 GPU를 사용하여 수행될 수 있다. CPU 또는 GPU가 자유로울 때, 작업 부하가 프로세서들 간에 재분배될 수 있다면 컴퓨팅 환경이 유리할 것이다.

[0008] 처리 전에, 작업부하는 많은 이산 작업으로 분할된다. 각 작업은 CPU 또는 GPU와 연관된 작업 큐(work queue)에 할당된다. CPU와 GPU를 포함하는 종래의 컴퓨팅 환경은 일단 작업이 처리를 위해 CPU 또는 GPU에 할당되면 상이한 유형의 처리 디바이스로 작업이 재분배되는 것을 허용하지 않는다. 종래의 시스템은 CPU가 다른 CPU에 작업을 재분배할 수 있게 하는 반면, GPU는 작업을 재분배하는 기능을 제공하지 않는다. 이것은 또한 GPU가 휴면 중인 동안 CPU는 작업 중이거나 또는 그 역으로 GPU가 작업 중인 동안 CPU는 휴면 중일 수 있으므로 처리를 방해한다. 특히 작업이 어느 처리 디바이스에서 처리될 수 있을 때 이러한 불균형 처리는 비효율성과 비 최적의 성능을 초래한다.

발명의 내용

해결하려는 과제

[0009] 그러므로, CPU와 GPU가 이들 간에 작업을 재분배하고 균형잡을 수 있게 하는 시스템 및 방법이 필요하다.

[0010] GPU, 가속 처리 유닛(APU: accelerated processing unit), 및 일반 목적 사용의 그래픽 처리 유닛(GPGPU: general purpose use of the graphics processing unit)이 이 분야에서 일반적으로 사용되는 용어이지만, "가속 처리 디바이스(APD: accelerated processing device)"라는 표현이 더 넓은 표현인 것으로 고려된다. 예를 들어, APD는 종래의 CPU, 종래의 GPU, 소프트웨어 및/또는 이들의 조합에 비해 가속된 방식으로 가속 그래픽 처리 작업, 데이터 병렬 작업, 또는 내포 데이터 병렬 작업과 연관된 기능(function)과 연산을 수행하는 하드웨어 및/또는 소프트웨어의 임의의 협력하는 집합을 말한다.

[0011] 본 발명의 실시예는 특정 실시예에서 이종 처리 디바이스(heterogeneous processing device)에서 작업 부하를 균형 잡는 방법, 시스템 및 제조 물품을 포함한다. 본 방법은 상이한 유형의 프로세서와 연관된 개체(entity)를 디큐잉(dequeuing)하는 것에 의해 하나의 유형의 프로세서의 메모리 저장매체에 액세스하고, 상이한 유형의 프로세서에 의해 처리될 수 있는 메모리 내 복수의 작업으로부터 하나의 작업을 식별하고, 메모리 저장매체에 액세스할 수 있는 복수의 디큐잉 개체를 동기화하고, 메모리 저장매체로부터 작업을 디큐잉하는 것을 포함한다.

[0012] 본 발명의 추가적인 특징과 이점 및 본 발명의 여러 실시예의 구조와 동작이 첨부 도면을 참조하여 상세히 후술한다. 본 발명은 본 명세서에 설명된 특정 실시예로 제한되는 것은 아니라는 것이 주목된다. 이 실시예는 단지 예시를 위해서만 본 명세서에 제공된 것이다. 추가적인 실시예는 본 명세서에 포함된 개시 내용에 기초하여 관련 기술 분야(들)에서 통상의 지식을 가진 자에게는 명백할 것이다.

도면의 간단한 설명

[0013] 본 명세서의 일부를 형성하고 본 명세서에 포함된 첨부 도면은 본 발명을 예시하고, 본 상세한 설명과 함께 본 발명의 원리를 설명하고 관련 기술 분야에 통상의 지식을 가진 자라면 본 발명을 제조하고 사용할 수 있게 하는 역할을 한다. 본 발명의 여러 실시예는 동일한 참조 부호가 도면 전체에 걸쳐 동일한 요소를 나타내는데 사용된

도면을 참조하여 후술한다.

도 1a는 본 발명의 실시예에 따른 처리 시스템의 예시적인 블록도;

도 1b는 도 1a에 도시된 APD의 예시적인 블록도;

도 2는 CPU와 APD가 동일한 실리콘 부분 위에 융합(fused)된 큐잉 시스템의 예시적인 블록도;

도 3은 이산 시스템 환경에서 큐잉 시스템의 예시적인 블록도;

도 4는 다수의 CPU와 APD를 위한 작업을 균형잡는 다수의 큐의 예시적인 블록도;

도 5는 융합 환경에서 CPU에서 처리하기 위해 작업을 저장하는 큐로부터 APD 디큐잉 작업의 예시적인 흐름도;

도 6은 APD에서 처리하기 위해 작업을 저장하는 큐로부터 CPU 디큐잉 작업의 예시적인 흐름도;

도 7은 이산 환경에서 CPU에서 처리하는 작업을 저장하는 큐로부터 CPU 디큐잉 작업의 예시적인 흐름도.

본 발명은 첨부 도면을 참조하여 설명된다. 일반적으로, 요소가 제일 먼저 나오는 도면은 일반적으로 대응하는 참조 부호에서 가장 좌측의 숫자(들)로 지시된다.

발명을 실시하기 위한 구체적인 내용

[0014] 이하 상세한 설명에서 "하나의 실시예", "일 실시예", "예시적인 실시예" 등으로 언급하는 것은 설명된 실시예가 특정 특징, 구조 또는 특성을 포함할 수 있으나 모든 실시예가 이 특정 특징, 구조 또는 특성을 반드시 포함하는 것은 아니라는 것을 나타낸다. 나아가, 이 어구는 반드시 동일한 실시예를 언급하는 것이 아니다. 나아가, 특정 특징, 구조 또는 특성이 일 실시예와 관련하여 설명될 때 이 특징, 구조, 또는 특성이 명시적으로 설명되었건 아니건 간에 다른 실시예에도 영향을 미친다는 것은 이 기술 분야에 통상의 지식을 가진 자의 지식 범위 내인 것으로 제시된다.

[0015] "본 발명의 실시예"라는 용어는 본 발명의 모든 실시예가 설명된 특징, 이점 또는 동작 모드를 포함하는 것을 요구하는 것이 아니다. 대안적인 실시예가 본 발명의 범위를 벗어남이 없이 고안될 수 있고, 본 발명의 잘 알려진 요소들은 본 발명의 관련 상세를 흐리게 하지 않기 위하여 상세히 설명되지 않거나 생략될 수 있다. 나아가, 본 명세서에 사용된 용어는 특정 실시예를 단지 설명하기 위한 것일 뿐 본 발명을 제한하려고 의도된 것이 전혀 아니다. 예를 들어, 본 명세서에 사용된 바와 같이, 단수 형태 "하나", "일" 및 "상기"는 문맥이 달리 명확히 지시하지 않는 한, 복수의 형태를 또한 포함하는 것을 의미한다. 또한 "포함한다", "포함하는", "구비한다" 및/또는 "구비하는"이라는 용어가 본 명세서에 사용될 때 이 용어는 언급된 특징, 완전체, 단계, 동작, 요소 및/또는 성분의 존재를 특정하는 것이나, 하나 이상의 다른 특징, 완전체, 단계, 동작, 요소, 성분, 및/또는 이들의 그룹의 존재나 추가를 배제하는 것은 아니다.

[0016] 도 1a는 2개의 프로세서, 즉 CPU(102)와 APD(104)를 구비하는 단일화된 컴퓨팅 시스템(100)의 예시적인 도면이다. CPU(102)는 하나 이상의 단일 또는 다수의 코어(CPU)를 포함할 수 있다. 본 발명의 일 실시예에서, 시스템(100)은 단일 실리콘 다이 또는 패키지 상에 형성되되 CPU(102)와 APD(104)를 결합하여 단일화된 프로그래밍 및 실행 환경을 제공한다. 이 환경은 APD(104)가 일부 프로그래밍 작업에 CPU(102)만큼 유동적으로 사용될 수 있게 한다. 그러나, CPU(102)와 APD(104)는 단일 실리콘 다이 상에 형성되는 것이 본 발명의 절대적 요건은 아니다. 일부 실시예에서 이들은 동일한 기판 상에 또는 상이한 기판 상에 별개로 형성되고 장착되는 것이 가능하다.

[0017] 일례에서, 시스템(100)은 메모리(106), 운영 시스템(108), 및 통신 인프라(109)를 또한 포함한다. 운영 시스템(108)과 통신 인프라(109)는 아래에서 보다 상세히 설명된다.

[0018] 시스템(100)은 또한 커널 모드 드라이버(KMD: kernel mode driver)(110), 소프트웨어 스케줄러(SWS: software scheduler)(112), 및 메모리 관리 유닛(memory management unit)(116), 예를 들어, 입력/출력 메모리 관리 유닛(IOMMU: input/output memory management unit)을 포함한다. 시스템(100)의 성분은 하드웨어, 펌웨어, 소프트웨어, 또는 이들의 임의의 조합으로 구현될 수 있다. 이 기술 분야에 통상의 지식을 가진 자라면 시스템(100)이 도 1a에 도시된 실시예에 도시된 것에 더하여 또는 이와 다르게 하나 이상의 소프트웨어, 하드웨어 및 펌웨어를 포함할 수 있다는 것을 인식할 수 있을 것이다.

[0019] 일례에서, KMD(110)와 같은 드라이버는 일반적으로 하드웨어와 연결된 컴퓨터 버스 또는 통신 서브시스템을 통해 디바이스와 통신한다. 호출 프로그램(calling program)이 드라이버에서 루틴을 호출할 때, 드라이버는 명령을 이 디바이스에 발송한다. 디바이스가 드라이버에 다시 데이터를 송신하면, 드라이버는 원래의 호출 프로그램

에서 루틴을 호출할 수 있다. 일례에서, 드라이버는 하드웨어에 종속하고 연산 시스템에 특정된다. 이들 드라이버는 통상 임의의 필요한 비동기 시간 종속 하드웨어 인터페이스에 필요한 인터럽트 핸들링(handling)을 제공한다.

[0020] 특히 현대 마이크로소프트 윈도우(Microsoft Windows)(등록상표) 플랫폼에 있는 디바이스 드라이버는 커널 모드(kernel-mode)(링 0)이나 유저 모드(링 3)에서 실행할 수 있다. 유저 모드에서 드라이버를 실행하는 주요 이점은 불량하게 기록된 유저 모드 디바이스 드라이버가 커널 메모리를 덮어쓰기하는(overwrite) 것에 의해 시스템과 충돌할 수 없으므로 안정성이 개선된다는 것이다. 한편, 유저/커널 모드 전이(transition)는 통상적으로 상당한 성능 오버헤드를 부과하여 이에 의해 낮은 지체(latency)와 높은 처리량 요구조건에 유저 모드 드라이버를 금지한다. 커널 공간은 시스템 호출의 사용을 통해서만 유저 모듈에 의해 액세스될 수 있다. UNIX 쉘(shell) 또는 다른 GUI 기반 애플리케이션과 같은 최종 유저 프로그램은 유저 공간의 일부이다. 이들 애플리케이션은 커널 지원 기능을 통해 하드웨어와 상호작용한다.

[0021] CPU(102)는 제어 프로세서(control processor), 전계 프로그래밍가능한 게이트 어레이(FPGA: field programmable gate array), 애플리케이션 특정 집적 회로(ASIC: application specific integrated circuit), 또는 디지털 신호 프로세서(DSP: digital signal processor) 중 하나 이상(미도시)을 포함할 수 있다. CPU(102)는 예를 들어, 컴퓨팅 시스템(100)의 동작을 제어하는 운영 시스템(108), KMD(110), SWS(112) 및 애플리케이션(111)을 포함하는 제어 로직(control logic)을 실행한다. 이 예시적인 실시예에서, CPU(102)는 일 실시예에 따라, 예를 들어 CPU(102)에 걸쳐 이 애플리케이션과 연관된 처리와 APD(104)와 같은 다른 처리 자원을 분배하는 것에 의해 애플리케이션(111)의 실행을 개시하고 제어한다.

[0022] 특히 APD(104)는 그래픽 동작, 및 예를 들어 특히 병렬 처리에 적합할 수 있는 다른 동작과 같은 선택된 기능을 위한 명령 및 프로그램을 실행한다. 일반적으로, APD(104)는 픽셀 동작, 기하학적 연산과 같은 그래픽 파이프라인 동작을 실행하고 이미지를 디스플레이로 렌더링하는데 종종 사용될 수 있다. 본 발명의 여러 실시예에서, APD(104)는 CPU(102)로부터 수신된 명령(command) 또는 인스트럭션(instruction)에 기초하여 연산(compute) 처리 동작(예를 들어, 비디오 동작, 물리적 시뮬레이션, 연산 유동 역학 등과 같은 예를 들어 그래픽과 관계없는 동작)을 더 실행할 수 있다.

[0023] 예를 들어, 명령(command)은 일반적으로 인스트럭션 세트 아키텍처(ISA)에서 한정되지 않은 특정 인스트럭션(instruction)으로 고려될 수 있다. 명령은 디스패치 프로세서, 명령 프로세서, 또는 네트워크 제어기와 같은 특별 프로세서에 의해 실행될 수 있다. 한편, 인스트럭션은 예를 들어 컴퓨터 아키텍처 내 프로세서의 단일 동작으로 고려될 수 있다. 일례에서, ISA의 2개의 세트를 사용할 때, 일부 인스트럭션은 x86 프로그램을 실행하는데 사용되고 일부 인스트럭션은 APD 유닛에서 커널을 실행하는데 사용된다.

[0024] 예시적인 실시예에서, CPU(102)는 APD(104)에 선택된 명령을 전송한다. 이들 선택된 명령은 그래픽 명령과, 병렬 실행을 따르는 다른 명령을 포함할 수 있다. 연산 처리 명령을 더 포함할 수 있는 이 선택된 명령은 CPU(102)와는 실질적으로 독립적으로 실행될 수 있다.

[0025] APD(104)는 하나 이상의 SIMD 처리 코어를 포함하나 이로 제한되지 않는 자기 자신의 연산 유닛(미도시)을 포함할 수 있다. 본 명세서에 언급된 바와 같이, SIMD는 파이프라인이거나 프로그래밍 모델이고, 여기서 커널은 자기 자신의 데이터와 공유 프로그램 카운터를 각각 구비하는 다수의 처리 요소에서 동시에 실행된다. 모든 처리 요소는 동일한 인스트럭션 세트를 실행한다. 예측을 사용하면 작업 항목이 각 발송된 명령에 관여하거나 관여하지 않게 된다.

[0026] 일례에서, 각 APD(104) 연산 유닛은 하나 이상의 스칼라 및/또는 벡터 부동 소수점 유닛(floating-point unit) 및/또는 산술 및 로직 유닛(ALU: arithmetic and logic unit)을 포함할 수 있다. APD 연산 유닛은 또한 역 RMS 유닛(inverse-square root unit) 및 사인/코사인 유닛(sine/cosine unit)과 같은 특수 목적 처리 유닛(미도시)을 더 포함할 수 있다. 일례에서, APD 연산 유닛은 본 명세서에서 집합적으로 셰이더 코어(shader core)(122)라고 지칭된다.

[0027] 하나 이상의 SIMD를 구비하면 일반적으로 그래픽 처리에 공통인 것과 같은 데이터-병렬 작업을 실행하는데 APD(104)가 이상적으로 적합하게 된다.

[0028] 픽셀 처리와 같은 일부 그래픽 파이프라인 동작, 및 다른 병렬 연산 동작은 동일한 명령 스트림이나 연산 커널이 입력 데이터 요소의 스트림이나 집합에 수행되는 것을 요구할 수 있다. 동일한 연산 커널의 각 인스턴스화(instantiation)는 이 데이터 요소를 병렬 처리하기 위하여 셰이더 코어(122)에 있는 다수의 연산 유닛에 동시

에 실행될 수 있다. 본 명세서에 언급된 바와 같이, 예를 들어, 연산 커널은 프로그램에 선언되고 APD에서 실행되는 인스트럭션을 포함하는 함수(function)이다. 이 함수는 또한 커널, 셰이더, 셰이더 프로그램 또는 프로그램이라고도 지칭된다.

[0029] 하나의 예시적인 실시예에서, 각 연산 유닛(예를 들어, SIMD 처리 코어)은 입력 데이터를 처리하도록 특정 작업 항목의 각 인스턴스화를 실행할 수 있다. 작업 항목은 명령에 의해 디바이스에서 호출되는 커널의 병렬 실행의 집합 중 하나이다. 작업 항목은 APD 연산 유닛에서 실행되는 작업 그룹의 일부로서 하나 이상의 처리 요소에 의해 실행될 수 있다.

[0030] 작업 항목이 전체 ID와 국부 ID에 의해 집합 내에서 다른 실행과 구별된다. 일례에서, SIMD에서 동시에 실행되는 작업 그룹에 있는 작업 항목의 서브세트는 웨이브프론트(wavefront)(136)라고 지칭될 수 있다. 웨이브프론트의 폭은 연산 유닛(예를 들어, SIMD 처리 코어)의 하드웨어의 특성이다. 본 명세서에 언급된 바와 같이, 작업 그룹은 단일 연산 유닛에서 실행되는 관련된 작업 항목의 집합이다. 이 그룹에 있는 작업 항목은 동일한 커널을 실행하고 국부 메모리와 작업 그룹 배리어(barrier)를 공유한다.

[0031] 예시적인 실시예에서, 작업 그룹으로부터 모든 웨이브프론트는 동일한 SIMD 처리 코어에서 실행된다. 웨이브프론트에 걸친 인스트럭션은 한번에 하나씩 발송되고, 모든 작업 항목이 동일한 제어 흐름을 따를 때, 각 작업 항목은 동일한 프로그램을 실행한다. 웨이브프론트는 또한 워프(warp), 벡터 또는 쓰레드(thread)라고도 지칭될 수 있다.

[0032] 실행 마스크 및 작업 항목 예측은 웨이브프론트 내 제어 흐름을 발산하는데 사용되는데, 여기서 각 개별 작업 항목은 커널을 통해 사실상 유니크한 코드 경로를 취할 수 있다. 부분적으로 식재된 웨이브프론트는 작업 항목의 전체 세트가 웨이브프론트 시작 시간에 이용가능하지 않을 때 처리될 수 있다. 예를 들어, 셰이더 코어(122)는 미리 결정된 개수의 웨이브프론트(136)를 동시에 실행할 수 있는데, 여기서 각 웨이브프론트(136)는 다수의 작업 항목을 포함한다.

[0033] 시스템(100)에서 APD(104)는 그래픽 메모리(130)와 같은 자기 자신의 메모리를 포함한다(메모리(130)는 그래픽 전용 사용으로 제한되지 않는다). 그래픽 메모리(130)는 APD(104)에서 연산 동안 사용하기 위해 국부 메모리를 제공한다. 셰이더 코어(122) 내에서 개별 연산 유닛(미도시)은 자기 자신의 국부 데이터 저장소(미도시)를 구비할 수 있다. 일 실시예에서, APD(104)는 메모리(106)에의 액세스뿐만 아니라 국부 그래픽 메모리(130)에의 액세스를 포함한다. 다른 실시예에서, APD(104)는 APD(104)에 직접 부착되고 메모리(106)와는 별도로 부착된 동적 랜덤 액세스 메모리(DRAM: dynamic random access memory) 또는 다른 그러한 메모리(미도시)에의 액세스를 포함할 수 있다.

[0034] 도시된 예에서, APD(104)는 하나 또는 "n"개의 명령 프로세서(CP: command processor)(124)를 더 포함한다. CP(124)는 APD(104)에서 처리를 제어한다. CP(124)는 메모리(106)에서 명령 버퍼(125)로부터 실행될 명령을 검색하며 APD(104)에서 이 명령의 실행을 조정한다.

[0035] 일례에서, CPU(102)는 애플리케이션(111)에 기반한 명령을 적절한 명령 버퍼(125)에 입력한다. 본 명세서에 언급된 바와 같이, 애플리케이션은 CPU와 APU 내 연산 유닛에서 실행되는 프로그램 부분의 조합이다.

[0036] 복수의 명령 버퍼(125)는 각각의 처리가 APD(104)에서 실행하도록 스케줄링되게 유지될 수 있다.

[0037] CP(124)는 하드웨어, 펌웨어, 또는 소프트웨어, 또는 이들의 조합으로 구현될 수 있다. 일 실시예에서, CP(124)는 스케줄링 로직을 포함하는 로직을 구현하는 마이크로코드를 가지는 감소된 인스트럭션 세트 컴퓨터(RISC: reduced instruction set computer) 엔진으로 구현된다.

[0038] APD(104)는 하나 또는 "n"개의 디스패치 제어기(DC: dispatch controller)(126)를 더 포함한다. 본 출원에서, 디스패치 라는 용어는 연산 유닛의 세트에서 작업그룹 세트의 커널의 실행 시작을 개시하는 문맥 상태(context state)를 사용하는 디스패치 제어기에 의해 실행되는 명령을 말한다. DC(126)는 셰이더 코어(122)에서 작업 그룹을 개시하는 로직을 포함한다. 일부 실시예에서, DC(126)는 CP(124)의 일부로서 구현될 수 있다.

[0039] 시스템(100)은 APD(104)에서 실행하기 위한 실행 리스트(150)로부터 처리를 선택하는 하드웨어 스케줄러(HWS: hardware scheduler)(128)를 더 포함한다. HWS(128)는 라운드 로빈 방법, 우선순위 레벨을 사용하거나 또는 다른 스케줄링 정책에 기초하여 실행 리스트(150)로부터 처리를 선택할 수 있다. 예를 들어, 우선순위 레벨은 동적으로 결정될 수 있다. HWS(128)는 예를 들어 새로운 처리를 추가하고 실행 리스트(150)로부터 현존하는 처리를 삭제하는 것에 의해 실행 리스트(150)를 관리하는 기능(functionality)을 더 포함할 수 있다. HWS(128)의 실

행 리스트 관리 로직은 실행 리스트 제어기(RLC: run list controller)라고 종종 지칭된다.

[0040] 본 발명의 여러 실시예에서, HWS(128)가 RLC(150)로부터 처리의 실행을 개시할 때, CP(124)는 대응하는 명령 버퍼(125)로부터 명령을 검색하고 실행하기 시작한다. 일부 경우에, CP(124)는 CPU(102)로부터 수신된 명령에 대응하는 APD(104)에서 실행될 하나 이상의 명령을 생성할 수 있다. 일 실시예에서, CP(124)는 다른 성분과 함께 APD(104) 및/또는 시스템(100)의 자원의 이용을 개선하거나 최대화하는 방식으로 APD(104)에서 명령의 우선순위 및 스케줄링을 구현한다.

[0041] APD(104)는 인터럽트 생성기(146)에 액세스하거나 이를 포함할 수 있다. 인터럽트 생성기(146)는 페이지 폴트(page fault)와 같은 인터럽트 이벤트가 APD(104)에 의해 나타날 때 운영 시스템(108)을 인터럽트하도록 APD(104)에 의해 구성될 수 있다. 예를 들어, APD(104)는 IOMMU(116) 내 인터럽트 생성 로직에 의존하여 전술한 페이지 폴트 인터럽트를 생성할 수 있다.

[0042] APD(104)는 셰이더 코어(122) 내에서 동시에 실행되는 처리를 선취하는 선취 및 문맥 스위치 로직(120)을 더 포함할 수 있다. 문맥 스위치 로직(120)은 예를 들어 처리를 중지하고 그 현재 상태(예를 들어, 셰이더 코어(122) 상태 및 CP(124) 상태)를 저장하는 기능을 포함한다.

[0043] 본 명세서에 언급된 바와 같이, 상태 라는 용어는 초기 상태, 중간 상태 및/또는 최종 상태를 포함할 수 있다. 초기 상태는 기계가 프로그래밍 순서에 따라 입력 데이터 세트를 처리하여 출력 데이터 세트를 생성하는 시작점이다. 예를 들어 처리가 순방향 진행을 하게 하는 여러 지점에서 저장될 필요가 있는 중간 상태가 있을 수 있다. 이 중간 상태는 일부 다른 처리에 의해 인터럽트될 때 차후에 계속 실행을 허용하기 위해 종종 저장된다. 출력 데이터 세트의 일부로 기록될 수 있는 최종 상태가 또한 있다.

[0044] 선취 및 문맥 스위치 로직(120)은 다른 처리를 APD(104)로 문맥 스위칭하는 로직을 더 포함할 수 있다. 다른 처리를 APD(104)에서 실행되는 것으로 문맥 스위칭하는 기능은 예를 들어 APD(104)에서 실행되는 CP(124)와 DC(126)를 통해 처리를 인스턴스화하고 이 처리에 대해 이전에 저장된 상태를 복원하며 그 실행을 시작하는 것을 포함할 수 있다.

[0045] 메모리(106)는 DRAM(미도시)과 같은 비 영구적인 메모리를 포함할 수 있다. 메모리(106)는 예를 들어, 애플리케이션이나 다른 처리 로직의 부분의 실행 동안 처리 로직 인스트럭션, 상수값, 및 변수값을 저장할 수 있다. 예를 들어, 일 실시예에서, CPU(102)에서 하나 이상의 동작을 수행하는 제어 로직의 부분들은 CPU(102)에 의한 동작의 각 부분의 실행 동안 메모리(106) 내에 상주할 수 있다.

[0046] 실행 동안, 각 애플리케이션, 운영 시스템 함수, 처리 로직 명령, 및 시스템 소프트웨어는 메모리(106)에 상주할 수 있다. 운영 시스템(108)에 기본적인 제어 로직 명령은 일반적으로 실행 동안 메모리(106)에 상주한다. 예를 들어, KMD(110)와 소프트웨어 스케줄러(112)를 포함하는 다른 소프트웨어 명령이 또한 시스템(100)의 실행 동안 메모리(106)에 상주할 수 있다.

[0047] 이 예에서, 메모리(106)는 APD(104)에 명령을 송신하도록 CPU(102)에 의해 사용되는 명령 버퍼(125)를 포함한다. 메모리(106)는 처리 리스트와 처리 정보(예를 들어, 활성 리스트(152)와 처리 제어 블록(154))를 더 포함한다. 이들 리스트 및 정보는 CPU(102)에서 실행되는 스케줄링 소프트웨어에 의해 사용되어 스케줄링 정보를 APD(104) 및/또는 관련된 스케줄링 하드웨어에 전달한다. 메모리(106)에 액세스는 메모리(106)에 연결된 메모리 제어기(140)에 의해 관리될 수 있다. 예를 들어, CPU(102)로부터 또는 다른 디바이스로부터 메모리(106)를 판독하거나 이 메모리에 기록하는 요청은 메모리 제어기(140)에 의해 관리된다.

[0048] 시스템(100)의 다른 측면을 더 참조하면, IOMMU(116)는 다수 문맥의 메모리 관리 유닛이다.

[0049] 본 명세서에 사용된 바와 같이 문맥은 커널이 실행되는 환경과, 동기화와 메모리 관리가 한정되는 범위로 고려될 수 있다. 문맥은 디바이스 세트, 이들 디바이스에 액세스가능한 메모리, 대응하는 메모리 특성, 및 메모리 객체에 대한 동작이나 커널(들)의 실행을 스케줄링하는데 사용되는 하나 이상의 명령 큐(command-queue)를 포함한다.

[0050] 도 1a에 도시된 예를 더 참조하면, IOMMU(116)는 APD(104)를 포함하는 디바이스에 대한 메모리 페이지 액세스를 위한 가상 어드레스-물리적 어드레스의 변환(virtual to physical address translation)을 수행하는 로직을 포함한다. IOMMU(116)는 예를 들어 APD(104)와 같은 디바이스에 의해 페이지 액세스가 페이지 폴트를 초래할 때 인터럽트를 생성하는 로직을 더 포함할 수 있다. IOMMU(116)는 변환 룩어사이드 버퍼(TLB: translation lookaside buffer)(118)를 더 포함하거나 이에 대한 액세스를 구비할 수 있다. TLB(118)는 일례로서 메모리

(106)에 있는 데이터에 대해 APD(104)에 의해 이루어진 요청에 대해 논리적(즉, 가상) 메모리 어드레스를 물리적 메모리 어드레스로 변환을 가속시키기 위해 콘텐츠 어드레스 가능한 메모리(CAM: content addressable memory)에 구현될 수 있다.

[0051] 도시된 예에서, 통신 인프라(109)는 필요에 따라 시스템(100)의 성분을 상호연결한다. 통신 인프라(109)는 주변 성분 상호연결(PCI: peripheral component interconnect) 버스, 확장된 PCI(extended PCI)(PCI-E) 버스, 개선된 마이크로제어기 버스 아키텍처(advanced microcontroller bus architecture)(AMBA) 버스, 가속 그래픽 포트(accelerated graphics port)(AGP), 또는 다른 이러한 통신 인프라 중 하나 이상(미도시)을 포함할 수 있다. 통신 인프라(109)는 이더넷, 또는 유사한 네트워크, 또는 애플리케이션의 데이터 전달율 요구조건(data transfer rate requirement)을 충족하는 임의의 적절한 물리적 통신 인프라를 더 포함할 수 있다. 통신 인프라(109)는 컴퓨팅 시스템(100)의 성분을 포함하는 성분을 상호연결하는 기능을 포함한다.

[0052] 이 예에서, 운영 시스템(108)은 시스템(100)의 하드웨어 성분을 관리하고 공통 서비스를 제공하는 기능을 포함한다. 여러 실시예에서, 운영 시스템(108)은 CPU(102)에서 실행되어 공통 서비스를 제공할 수 있다. 이 공통 서비스는 예를 들어, CPU(102)에서 실행하기 위한 애플리케이션의 스케줄링, 폴트 관리, 인터럽트 서비스, 및 다른 애플리케이션의 입력과 출력의 처리를 포함할 수 있다.

[0053] 일부 실시예에서, 인터럽트 제어기(148)와 같은 인터럽트 제어기에 의해 생성된 인터럽트에 기초하여 운영 시스템(108)은 적절한 인터럽트 핸들링 루틴을 호출한다. 예를 들어, 페이지 폴트 인터럽트를 검출할 때 운영 시스템(108)은 인터럽트 핸들러를 호출하여 메모리(106)에 관련 페이지의 로딩을 개시하고 대응하는 페이지 테이블을 업데이트할 수 있다.

[0054] 운영 시스템(108)은 운영 시스템으로 관리되는 커널 기능을 통해 하드웨어 성분에의 액세스가 중재되는 것을 보장하는 것에 의해 시스템(100)을 보호하는 기능을 더 포함할 수 있다. 사실상, 운영 시스템(108)은 애플리케이션(111)과 같은 애플리케이션이 유저 공간에서 CPU(102)에서 실행되는 것을 보장한다. 운영 시스템(108)은 애플리케이션(111)이 하드웨어 및/또는 입력/출력 기능에 액세스하기 위해 운영 시스템에 의해 제공되는 커널 기능을 호출하는 것을 더 보장한다.

[0055] 예를 들어, 애플리케이션(111)은 CPU(102)에서 또한 실행되는 유저 연산을 수행하는 여러 프로그램이나 명령을 포함한다. CPU(102)는 APD(104)에서 처리하기 위한 선택된 명령을 끊김없이 송신할 수 있다.

[0056] 일례에서, KMD(110)는 CPU(102), 또는 CPU(102) 또는 다른 로직에서 실행되는 애플리케이션이 APD(104) 기능을 호출할 수 있게 하는 애플리케이션 프로그램 인터페이스(API: application program interface)를 구현한다. 예를 들어, KMD(110)는 CPU(102)로부터 명령 버퍼(125)로 명령을 인큐잉시키고(enqueue) 이 명령 버퍼로부터 APD(104)는 이 명령을 후속적으로 검색할 수 있다. 추가적으로, KMD(110)는 SWS(112)와 함께 APD(104)에서 실행되는 처리의 스케줄링을 수행할 수 있다. SWS(112)는 예를 들어 APD에서 실행되는 처리의 우선순위 리스트를 유지하는 로직을 포함할 수 있다.

[0057] 본 발명의 다른 실시예에서, CPU(102)에서 실행되는 애플리케이션은 명령을 인큐잉시킬 때 KMD(110)를 완전히 바이패스(bypass)할 수 있다.

[0058] 일부 실시예에서, SWS(112)는 APD(104)에서 실행되는 처리의 메모리(106)에 활성 리스트(152)를 유지한다. SWS(112)는 하드웨어에서 HWS(128)에 의해 관리되는 활성 리스트(152)에서 처리의 서브세트를 더 선택한다. APD(104)에서 각 처리를 실행하는데 관련된 정보는 CPU(102)로부터 처리 제어 블록(PCB: process control block)(154)을 통해 APD(104)로 전달된다.

[0059] 애플리케이션, 운영 시스템, 및 시스템 소프트웨어를 위한 처리 로직은 궁극적으로 본 명세서에 설명된 본 발명의 측면을 구현하는 하드웨어 디바이스를 생성하도록 마스크작업/포토마스크의 생성을 통해 제조 공정을 구성할 수 있도록 C와 같은 프로그래밍 언어로 및/또는 베릴로그(Verilog), RTL 또는 네트리스트와 같은 하드웨어 설명 언어(hardware description language)로 지정된 명령을 포함할 수 있다.

[0060] 이 기술 분야에 통상의 지식을 가진 자라면 본 설명을 판독하는 것에 의해 연산 시스템(100)이 도 1a에 도시된 것보다 더 많거나 더 적은 수의 성분을 포함할 수 있다는 것을 이해할 수 있을 것이다. 예를 들어, 연산 시스템(100)은 하나 이상의 입력 인터페이스, 비휘발성 저장매체, 하나 이상의 출력 인터페이스, 네트워크 인터페이스, 및 하나 이상의 디스플레이 또는 디스플레이 인터페이스를 포함할 수 있다.

[0061] 도 1b는 도 1a에 도시된 APD(104)의 보다 상세한 설명을 보여주는 일 실시예이다. 도 1b에서, CP(124)는 CP 파

이프라인(124a, 124b, 124c)을 포함할 수 있다. CP(124)는 도 1a에 도시된 명령 버퍼(125)로부터 입력으로 제공된 명령 리스트를 처리하도록 구성될 수 있다. 도 1b의 예시적인 동작에서, CP 입력 0(124a)은 그래픽 파이프라인(graphics pipeline)(162)으로 명령을 구동하는 일을 담당한다. CP 입력 1 및 2(124b, 124c)는 연산 파이프라인(160)에 명령을 전달한다. 또한, HWS(128)의 동작을 제어하는 제어기 메커니즘(166)이 제공된다.

[0062] 도 1b에서, 그래픽 파이프라인(162)은 본 명세서에서 정렬된 파이프라인(164)라고 지칭된 블록 세트를 포함할 수 있다. 일례로서, 정렬된 파이프라인(164)은 정점 그룹 변환기(VGT: vertex group translator)(164a), 프리미티브 어셈블러(PA: primitive assembler)(164b), 스캔 변환기(SC: scan converter)(164c), 및 셰이더-엑스포트(shader-export), 렌더-백 유닛(SX/RB: render-back unit)(176)을 포함한다. 정렬된 파이프라인(164) 내 각 블록은 그래픽 파이프라인(162)에서 상이한 그래픽 처리 단계를 나타낼 수 있다. 정렬된 파이프라인(164)은 고정된 함수의 하드웨어 파이프라인일 수 있다.

[0063] 또한 본 발명의 사상과 범위 내에 있을 수 있는 다른 구현들이 사용될 수 있다. 소량의 데이터만이 그래픽 파이프라인(162)에 입력으로 제공될 수 있지만 이 데이터는 그래픽 파이프라인(162)으로부터 출력으로 제공되는 시간만큼 증폭된다. 그래픽 파이프라인(162)은 CP 파이프라인(124a)으로부터 수신된 작업 항목 그룹 내 범위를 통해 카운트하는 DC(166)를 더 포함한다. DC(166)를 통해 제출된 연산 작업은 그래픽 파이프라인(162)과 반동기적이다.

[0064] 연산 파이프라인(160)은 셰이더 DC(168, 170)를 포함한다. DC(168, 170) 각각은 CP 파이프라인(124b, 124c)으로부터 수신된 작업 그룹 내 연산 범위를 통해 카운트하도록 구성된다.

[0065] 도 1b에 도시된 DC(166, 168, 170)는 입력 범위를 수신하고 이 범위를 작업그룹으로 분할하고 이후 작업그룹을 셰이더 코어(122)로 전달한다.

[0066] 그래픽 파이프라인(162)은 일반적으로 고정된 함수의 파이프라인이므로, 그 상태를 저장하고 복원하는 것은 어렵고, 그 결과 그래픽 파이프라인(162)은 문맥 스위칭하는 것이 어렵다. 그리하여 대부분의 경우에 본 명세서에 설명된 바와 같이 문맥 스위칭은 그래픽 처리 중에서 문맥 스위칭에 관한 것이 아니다. 예외는 문맥 스위칭될 수 있는 셰이더 코어(122)에서 그래픽 작업에 대한 것이다.

[0067] 그래픽 파이프라인(162)에서 작업의 처리가 완료된 후에 완료된 작업은 렌더 백 유닛(176)을 통해 처리되는데, 이 렌더백 유닛은 깊이와 컬러 계산을 한 후에 최종 결과를 메모리(130)에 기록한다.

[0068] 셰이더 코어(122)는 그래픽 파이프라인(162)과 연산 파이프라인(160)에 의해 공유될 수 있다. 셰이더 코어(122)는 웨이브프론트를 실행하도록 구성된 일반 프로세서일 수 있다.

[0069] 일례에서, 연산 파이프라인(160) 내 모든 작업은 셰이더 코어(122) 내에서 처리된다. 셰이더 코어(122)는 프로그래밍가능한 소프트웨어 코어를 실행하고 상태 데이터와 같은 여러 형태의 데이터를 포함한다.

[0070] 도 2는 작업 부하를 CPU와 APD 처리 디바이스에서 처리하기 위해 균형잡고 재분배하는 큐잉 시스템(200)의 예시적인 블록도이다. 큐잉 시스템(200)은 큐(202), 작업(204), 세마포어 블록(semaphore block)(206), CP(124)(본 명세서에 설명된), 하나 이상의 SIMD 스케줄러(208), 셰이더 코어(122), CPU 동기화 모듈(210), CPU 디큐잉 모듈(212), 및 CPU 코어(214)를 포함한다.

[0071] CPU(102)는 본 명세서에 설명된 바와 같은 하나 이상의 CPU 코어(214)를 포함한다. 각 CPU 코어(214)는 CPU(102)에서 컴퓨터 인스트럭션과 데이터를 처리한다.

[0072] 큐(202)는 시스템 메모리(106)로부터 할당된 메모리 세그먼트이다. 큐는 선입선출("FIFO": first-in, first-out) 원리에 따라 동작한다. 즉, 큐에 제일 먼저 인큐잉된 작업 부하가 큐에서 제일 먼저 디큐잉된 작업 부하가 된다. 추가적으로, 이 기술 분야에 통상의 지식을 가진 자라면 특정 큐 데이터 구조에 대한 설명은 예를 들어 주어진 것이고 발명을 제한하는 것이 아니며 데이터 구조를 저장하는 다른 메모리가 사용될 수 있다는 것을 이해할 수 있을 것이다.

[0073] 큐(202)는 공중 큐(public queue)이다. 공중 큐는 CPU(102)와 APD(104)와 같은 처리 디바이스에 액세스가능하다. 큐(202)는 FIFO 원리에 따라 큐(202)에 인큐잉되고 디큐잉되는 다수의 작업(204)을 저장한다. 작업(204)은 APD(104) 또는 CPU(102)에서 처리하기 위해 스케줄링된 운영 시스템 인스트럭션, 애플리케이션 인스트럭션, 이미지 및 데이터를 포함하는 독립적인 작업이다. 작업은 "입도(grain)"에 따라 작업(204)으로 분리되며, 여기서 입도는 작업(204)의 사이즈를 나타낸다. 입도의 사이즈는 APD(104)와 CPU(102) 프로세서에 대해 스케줄링된 작업(204)에 따라 변한다. 예를 들어, CPU(102)에서 처리되는 작업(204)에 대한 입도 사이즈는 APD(104)에서 처리

되는 작업(204)에 대한 입도 사이즈보다 일반적으로 더 작다.

[0074] 작업(204)은 처리를 요구하는 데이터에 대한 포인터 및/또는 정보 인스트럭션을 보유하는 데이터 구조를 포함한다. 예를 들어, 작업(204)을 위한 정보를 보유하는 데이터 구조는 MyTask 구조(MyTask structure)로 한정될 수 있다. 비제한적인 예로서 MyTask 구조는 이하 파라미터를 포함할 수 있다:

```
struct MyTask {
        MyPtr myCodePtr
                myCPUCodePtr : pointer to code (x86 binary format)
                myAPDCodePtr :
                        //Pointer to code (shader binary format)
        MyPtr myDataPtr :
                myExecRange:
                        //Global grid dimensions
                        //Local grid dimensions
                myArgSize
                myArgs {(variable size)}
        MyNotification
                 //Pointer to notification mechanism
        MyAffinity
                //processing preference
}
```

[0075]

[0076] MyTask 구조는 시스템 메모리(106)에 또는 다른 메모리 디바이스에 저장된 컴파일된 CPU 코드와 APD 마이크로코드에의 포인터를 포함한다. 상기 예에서, MyPtr myCodePtr는 myAPDCodePtr로 CP(124)에서 실행되는 마이크로코드에 및 myCPUCodePtr로 CPU(102)에서 실행되는 컴파일된 소스 코드에 대한 포인터를 한정한다. myAPDCodePtr은 셰이더 코어(122)가 작업(204)에서 데이터를 실행하는데 사용하는 함수를 포함하는 마이크로코드를 나타낸다. 예를 들어, 작업(204)이 APD(104)에서 실행되면, APD(104)는 myAPDCodePtr에서 저장된 어드레스를 가지는 함수에 액세스한다. 작업(204)이 CPU(102)에서 실행되면, CPU(102)는 myCPUCodePtr에서 저장된 어드레스를 가지는 함수에 액세스한다. 일 실시예에서, myCodePtr는 미리 결정된 이벤트의 발생 후에 실행가능하게 되는 종속 정보를 포함하는 중간 언어 표현을 또한 나타낼 수도 있다.

[0077] 상기 예에서, MyTask 구조는 MyPtr myDataPtr을 포함할 수 있다. myDataPtr은 작업(204)이 처리를 요구하는 시스템 메모리(106) 내 데이터의 위치에 대한 포인터이다. 또한, myDataPtr는 작업(204)에서 데이터와 연관된 정보를 포함하는 파라미터를 구비한다. 예를 들어, 파라미터 myArgs는 변수(argument) 리스트를 포함하고, myArgSize는 변수의 수를 포함하고, myExecRange는 데이터 그리드(grid)의 차원을 포함한다.

[0078] 본 발명의 실시예에서, MyTask 구조는 MyAffinity 파라미터를 더 포함한다. MyAffinity의 값은 작업(204)을 실행하는 처리 디바이스를 결정한다. 예를 들어, MyAffinity 파라미터의 값은 CPU(102) 또는 APD(104)와 같은 처리 디바이스에 대한 선호도, 요구조건, 힌트 등을 나타낼 수 있다.

[0079] 이 기술 분야에서 통상의 지식을 가진 자라면 MyTask와 같은 데이터 구조는 다른 파라미터를 더 포함할 수 있다는 것을 이해할 수 있을 것이다.

[0080] CPU 디큐잉 모듈(212)과 CP(124)는 디큐잉 개체로 기능한다. 디큐잉 개체는 처리 디바이스에서 실행하기 위해 큐(202)로부터 작업을 디큐잉하거나 제거한다.

[0081] CPU 디큐잉 모듈(212)은 큐(202)에 액세스하고 CPU(102)에서 처리하기 위해 작업(204)을 제거하는 소프트웨어 모듈이다. 일 실시예에서, CPU 디큐잉 모듈(212)은 CPU(102)가 처리할 작업(204)을 요구할 때 APD(104)와 연관된 큐(202)로부터 작업을 제거한다. 예를 들어, CPU(102)와 연관된 큐(202)가 비어있을 때 APD(104)와 연관된 큐(202)는 처리를 요구하는 작업(204)을 저장한다.

[0082] 일반적으로, CPU 디큐잉 모듈(212)은 FIFO 원리를 사용하여 작업(204)을 검색한다. 작업(들)(204)을 제거하기 전에, CPU 디큐잉 모듈(212)은 MyAffinity 파라미터에 액세스하여 작업(204)이 CPU(102)에서 처리하는데 적합한지의 여부를 결정한다. 예를 들어, CPU 디큐잉 모듈(212)은 MyAffinity 파라미터가 요구조건으로 APD(104)에서 처리하는 것으로 설정되지 않은 작업(들)(204)을 디큐잉한다. 다른 예에서, CPU 디큐잉 모듈(212)은 MyAffinity 파라미터가 참조로 APD(104)에서 처리하는 것으로 설정되지 않은 작업(들)(204)을 디큐잉한다. 일반적으로, 병

렬 프로세서에 의해 실행될 수 있는 수리적으로 복잡한 동작을 포함하는 작업(들)(204)은 참조 또는 요구조건으로 APD(104)에서 처리하는 것으로 MyAffinity 파라미터를 설정될 수 있다.

[0083] 다수의 CPU 코어(214) 환경에서, CPU 디큐잉 모듈(212)은 특정 CPU 코어(214)에 대응한다.

[0084] CP(124)는 큐(202)에 액세스하여 APD(104)에서 처리하기 위해 작업(204)을 제거한다. CP(124)는 APD(104)에서 처리하기 위해 큐(202)로부터 작업(204)을 제거하는 하드웨어 모듈이다. CPU 디큐잉 모듈(212)과 유사하게, APD(104)와 연관된 큐(202)는 비어 있으나, CPU(102)와 연관된 큐(202)는 처리를 요구하는 작업(204)을 저장하고 있을 때 CP(124)는 CPU(102)와 연관된 큐(202)로부터 작업(204)을 제거할 수 있다.

[0085] CP(124)는 FIFO 원리에 따라 작업(204)을 검색한다. 작업(들)(204)을 제거하기 전에, CP(124)는 MyAffinity 파라미터를 사용하여 작업(204)이 APD(104)에서 처리하기에 적합한지의 여부를 결정한다. 예를 들어, CP(124)는 MyAffinity 파라미터가 요구조건으로 CPU(102)에서 처리하는 것으로 설정되지 않은 작업(들)(204)을 디큐잉한다. 다른 예에서, CP(124)는 MyAffinity 파라미터가 참조로 CPU(102)에서 처리하는 것으로 설정하지 않은 작업(들)(204)을 디큐잉한다. 일반적으로, 브랜치 같은 코드를 포함하는 작업(들)(204)은 MyAffinity 파라미터를 참조 또는 요구조건으로 CPU(102)에서 처리하는 것으로 설정될 수 있다.

[0086] CP(124)가 작업(204)을 제거한 후에, CP는 작업(204)을 하나 이상의 셰이더 파이프 보간기(SPI: shader pipe interpolator)(208)에 전달한다. SPI(208)는 셰이더 코어(122)에서 처리하기 위한 작업(204)을 준비한다. 일 실시예에서, SPI(208)는 작업(204)을 처리하는데 요구되는 셰이더 코어(122)와 작업 항목의 수를 결정한다.

[0087] CPU 디큐잉 모듈(212)과 CP(124)가 큐(202)로부터 작업(204)을 제거하기 전에, 이들은 동기화된다. 동기화는 작업(204)이 제거될 때 큐(202)에 연속적이고 배타적인 액세스를 보장한다. CPU 동기화 모듈(210)은 CPU 디큐잉 모듈(212)이 큐(202)로부터 작업(204)을 제거하기 전에 큐(202) 및 APD(104)와 CPU 디큐잉 모듈(212)을 동기화한다. CPU 동기화 모듈(210)은 CPU 디큐잉 모듈(212)이 CPU(102)에서 처리하기 위한 작업(204)을 제거하는 시도를 할 때에만 CPU 디큐잉 모듈(212)이 큐(202)에 액세스하는 것을 보장한다.

[0088] CPU 동기화 모듈(210)은 CPU 디큐잉 모듈(212)이 큐(202)에 배타적인 액세스를 구비하는 것을 보장하기 위해 원자 동작(atomic operation)을 사용한다. 이 기술 분야에 통상의 지식을 가진 자라면 메모리 위치에 액세스하는 다른 처리 또는 하드웨어 디바이스가 액세스를 완료할 때까지 처리 또는 하드웨어 디바이스가 메모리 위치로부터 판독되거나 메모리 위치에 기록되는 것을 원자 동작이 방지한다는 것을 이해할 수 있을 것이다.

[0089] APD(104)에서 처리하기 위해 작업(204)을 제거하기 전에, 세마포어 블록(206)은 큐(202) 및CPU(102)와 CP(124)를 동기화한다. 세마포어 블록(206)은 또한 CP(124)를 위한 큐(202)에 배타적인 액세스를 보장한다. 일 실시예에서, 세마포어 블록(206)은 CP(124)가 큐(202)에 배타적인 액세스를 하는 것을 보장하기 위해 원자 동작을 사용한다. 다른 실시예에서, 세마포어 블록(206)은 큐(202)에 배타적인 액세스를 보장하기 위해 이벤트 통지 메커니즘(event notification mechanism)을 사용한다. 이 기술 분야에 통상의 지식을 가진 자라면 이벤트 통지 메커니즘은 다른 처리 또는 하드웨어 디바이스에 의해 특정 메모리 위치가 액세스되고 있다는 것을 처리 또는 하드웨어 디바이스에 통지한다는 것을 이해할 수 있을 것이다.

[0090] APD(104)와 CPU(102)는 큐(202)로부터 상이한 개수의 작업(204)을 검색한다. 이 기술 분야에 통상의 지식을 가진 자라면 APD(104)는 더 많은 작업(204)을 병렬로 처리할 수 있으므로 APD(104)는 더 많은 작업(204)을 검색한다는 것을 이해할 수 있을 것이다. 그 결과, CP(124)와 CPU 디큐잉 모듈(212)이 큐(202)로부터 작업(204)을 검색할 때, 각 디큐잉 디바이스가 큐(202)로부터 제거하는 작업(204)의 수는 APD(104) 또는 CPU(102)가 처리를 요청하였는지에 좌우된다.

[0091] 이산 프로세서 환경에서, 세마포어 블록(216)은 큐(202)를 직접 동기화하지 못할 수 있어서 추가적인 성분을 요구한다. 도 3은 이산 처리 환경에서 작업부하를 재분배하는 큐잉 시스템의 블록도이다. 본 명세서에 설명된 성분에 더하여, 이산 시스템 환경에서, APD(104)는 APD 드라이버 모듈(302)과, 큐(202)로부터 작업(204)을 디큐잉하는 APD 디큐잉 모듈(304)을 포함한다. APD 드라이버 모듈(302)은 APD(104)에서의 전반적인 실행을 제어하는 소프트웨어 모듈이다. APD 디큐잉 모듈(302)은 큐(202)로부터 작업(204)을 검색하는 소프트웨어 기반 모듈이다.

[0092] APD(104)가 작업을 요청할 때, 세마포어 블록(206)은 APD 드라이버 모듈(302)과 통신한다. APD 드라이버 모듈(302)은 APD 디큐잉 모듈(304)과 통신한다. APD 디큐잉 모듈(304)은 큐(202)로부터 작업(204)을 제거하고 작업(204)을 CP(124)에 제공한다.

[0093] 도 4는 CPU(102)와 APD(104)와 통신하는 다수의 큐(202)를 포함하는 동작 환경(400)의 블록도이다.

[0094] 각 큐(202)는 다수의 CPU(102)와 APD(104)와 통신할 수 있으나, 큐(202)는 기본적으로 특정 CPU(102), 특정 CPU 코어(214), 또는 특정 APD(104)를 위한 작업을 저장할 수 있다.

[0095] CP(124)는 CPU(102)와 연관된 다수의 큐(202)로부터 작업(204)을 제거하고 본 명세서에 설명된 바와 같은 처리를 위해 작업(204)을 APD(104)에 전달할 수 있다. 유사하게, CPU 디큐잉 모듈(212)은 본 명세서에 설명된 바와 같이 CPU(102)에서 처리하기 위해 APD(104)와 연관된 다수의 큐(202)로부터 작업(204)을 제거할 수 있다.

[0096] 도 5는 큐(202)로부터 작업(204)을 제거하는 CP(124)의 예시적인 환경의 흐름도(500)이다.

[0097] 동작(502)에서, APD(104)는 처리를 요구하는 작업(204)을 요청한다.

[0098] 동작(504)에서, CP(124)는 큐(202)에 액세스한다.

[0099] 동작(506)에서, CP(124)는 처리를 요구하는 작업(들)(204)을 식별하고 APD(104)에서 처리될 수 있다. 예를 들어, CP(124)는 작업(들)(204)에서 MyAffinity 파라미터의 값을 식별한다. 일 실시예에서, CP(124)는 큐(202)로부터 디큐잉되도록 스케줄링된 작업(204) 내 MyAffinity 파라미터를 식별한다. CP(124)가 작업(들)(204)을 식별하면, 흐름도는 동작(508)으로 진행한다. 그렇지 않으면 흐름도(500)는 종료한다.

[0100] 동작(508)에서, 세마포어 블록(206)은 큐(202)와 CPU 동기화 모듈(210)을 동기화한다.

[0101] 동작(510)에서, CP(124)는 큐(202)로부터 작업(들)(204)을 디큐잉한다.

[0102] 동작(512)에서, CP(124)는 작업(들)(204)을 SPI(208)로 송신한다.

[0103] 동작(514)에서, SPI(208)는 셰이더 코어(122)에서 처리 작업(들)(204)에 요구되는 자원을 결정한다.

[0104] 동작(516)에서, 작업(204)은 셰이더 코어(122)에서 처리된다.

[0105] 도 6은 큐(202)로부터 작업(204)을 제거하는 APD 디큐잉 모듈(210)의 예시적인 실시예의 흐름도(600)이다.

[0106] 동작(602)에서, CPU(102)는 큐(202)로부터 작업(들)(204)을 요청한다.

[0107] 동작(604)에서, CPU 디큐잉 모듈(212)은 처리를 요구하여 CPU(102)에서 처리될 수 있는 작업(들)(204)을 식별한다. 예를 들어, CP(124)는 작업(들)(204)에서 MyAffinity 파라미터의 값을 식별한다. 일 실시예에서, CP(124)는 큐(202)로부터 디큐잉되도록 스케줄링된 작업(들)(204)에서 MyAffinity 파라미터를 식별한다. CPU 디큐잉 모듈(212)이 작업(들)(204)을 식별하면, 흐름도는 동작(606)으로 진행한다. 그렇지 않으면 흐름도는 종료한다.

[0108] 동작(606)에서, CPU 동기화 모듈(212)은 큐(202)와 APD(104)를 동기화하여 CPU 디큐잉 모듈(212)만이 큐(202)에 액세스하게 한다.

[0109] 동작(608)에서, CPU 동기화 모듈(212)은 본 명세서에 설명된 바와 같이 큐(202)로부터 작업(204)을 제거하고 CPU(102)에서 처리하기 위해 작업(204)을 송신한다.

[0110] 동작(610)에서 CPU(102)는 작업(204)을 처리한다.

[0111] 도 7은 이산 환경에서 APD(104)에서 처리하기 위해 큐(202)로부터 작업(204)을 제거하는 APD 디큐잉 모듈(304)의 예시적인 실시예의 흐름도(700)이다.

[0112] 동작(702)에서, APD(104)는 동작(502)에서 설명된 처리를 요구하는 작업(들)(204)을 요청한다.

[0113] 동작(706)에서, APD(104)는 작업(들)(204)을 위한 요청을 APD 드라이버 모듈(302)에 송신한다.

[0114] 동작(708)에서, APD 드라이버 모듈(302)은 요청을 APD 디큐잉 모듈(304)에 송신한다.

[0115] 동작(710)에서, APD 디큐잉 모듈(304)은 처리를 요구하여 동작(506)에서 설명된 바와 같이 APD(104)에서 처리될 수 있는 작업(들)을 식별한다.

[0116] 동작(712)에서, 세마포어 블록(206)은 동작(508)에서 설명된 바와 같이 큐(202)와 CPU 동기화 모듈(210)을 동기화한다.

[0117] 동작(714)에서, APD 디큐잉 모듈(304)은 동작(510)에서 설명된 바와 같이 APD(104)에서 처리하기 위한 작업(204)을 디큐잉하고 작업(204)을 APD 드라이버 모듈(302)에 송신한다.

[0118] 동작(716)에서, APD 드라이버 모듈(302)은 작업(204)을 APD(104)에 송신하며, 여기서 작업(204)은 동작(508 내

지 512)에서 설명된 바와 같이 처리된다.

[0119] 본 발명의 여러 측면은 소프트웨어, 펌웨어, 하드웨어, 또는 이들의 조합으로 구현될 수 있다. 예를 들어, 도 5의 흐름도(500), 도 6의 흐름도(600), 도 7의 흐름도(700)로 예시된 방법은 도 1의 단일화된 컴퓨팅 시스템(100)으로 구현될 수 있다. 본 발명의 여러 실시예는 이 예시적인 단일화된 컴퓨팅 시스템(100)에 비춰 설명된다. 이 기술 분야에 통상의 지식을 가진 자에게는 다른 컴퓨터 시스템 및/또는 컴퓨터 아키텍처를 사용하여 본 발명을 구현할 수 있다는 것은 명백할 것이다.

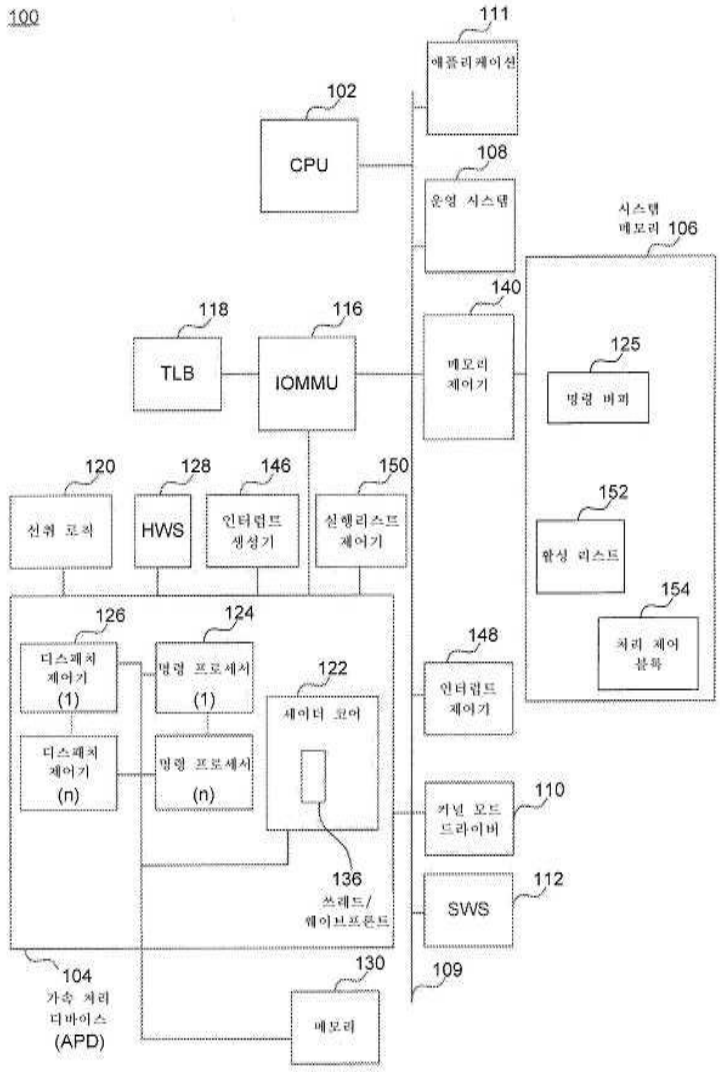
[0120] 본 문서에서, "컴퓨터 프로그램 매체" 및 "컴퓨터 사용가능한 매체"라는 용어는 일반적으로 이동식 저장 유닛 또는 하드 디스크 드라이브와 같은 매체를 말하는 데 사용된다. 컴퓨터 프로그램 매체 및 컴퓨터 사용가능한 매체는 메모리 반도체(예를 들어, DRAM 등)일 수 있을 시스템 메모리(106)와 그래픽 메모리(130)와 같은 메모리를 또한 말할 수 있다. 이들 컴퓨터 프로그램 제품은 단일화된 컴퓨팅 시스템(100)에 소프트웨어를 제공하는 수단이다.

[0121] 본 발명은 또한 임의의 컴퓨터 사용가능한 매체에 저장된 소프트웨어를 포함하는 컴퓨터 프로그램 제품에 관한 것이다. 이러한 소프트웨어는 하나 이상의 데이터 처리 디바이스에서 실행될 때 데이터 처리 디바이스(들)로 하여금 본 명세서에 설명된 바와 같이 동작하게 하거나 또는 전술한 바와 같이 본 명세서에 설명된 본 발명의 실시예를 구현하도록 컴퓨팅 디바이스(예를 들어, ASIC, 또는 프로세서)를 합성하거나 및/또는 제조할 수 있게 한다. 본 발명의 실시예는 현재 알려지거나 미래에 알려진 임의의 컴퓨터 사용하거나 판독가능한 매체를 사용한다. 컴퓨터 사용가능한 매체의 예로는 1차 저장 디바이스(예를 들어, 임의의 유형의 랜덤 액세스 메모리), 2차 저장 디바이스(예를 들어, 하드 드라이브, 플로피 디스크, CD ROMS, ZIP 디스크, 테이프, 자기 저장 디바이스, 광 저장 디바이스, MEMS, 나노기술 저장 디바이스 등)를 포함하나 이로 제한되지 않는다.

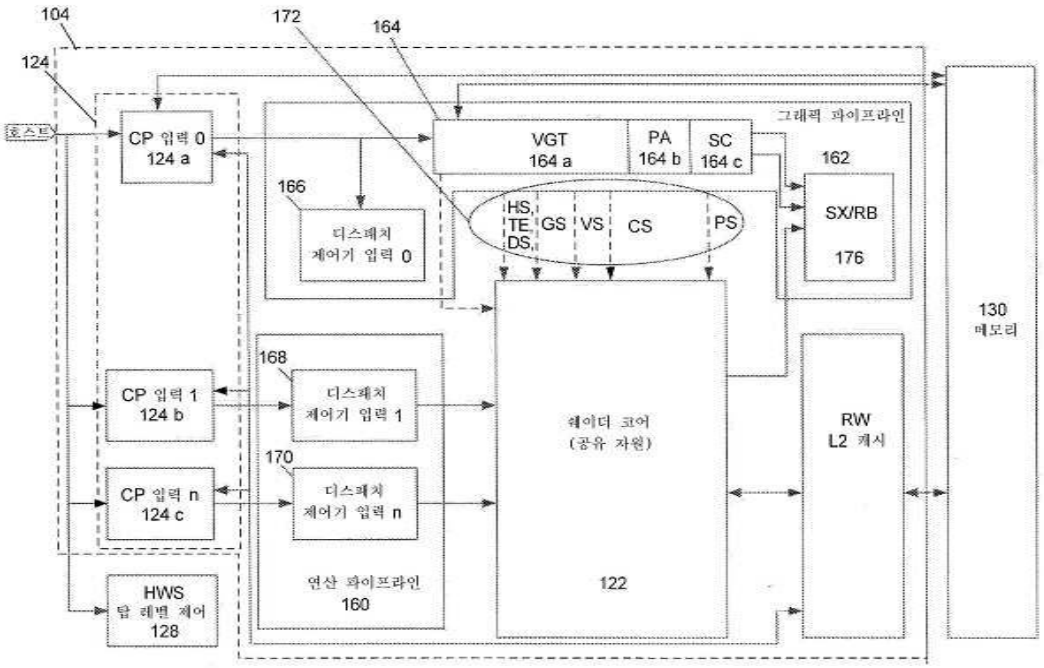
[0122] 본 발명의 여러 실시예가 전술되었으나, 본 발명의 이들 실시예는 예를 들어 제한 없이 제공된 것인 것으로 이해된다. 이 기술 분야에 통상의 지식을 가진 자라면 형태와 상세에서 여러 변경이 첨부된 청구범위에 한정된 본 발명의 사상과 범위를 벗어남이 없이 이루어질 수 있다는 것을 이해할 수 있을 것이다. 본 발명은 이들 예로만 제한되지 않는 것으로 이해된다. 본 발명은 본 명세서에 설명된 바와 같이 동작하는 임의의 요소에 적용가능하다. 따라서, 본 발명의 폭과 범위는 전술한 예시적인 실시예 중 어느 하나로 제한되어서는 안 되며 이하의 특허청구범위와 그 균등물에 따라서만 한정되어야 한다.

도면

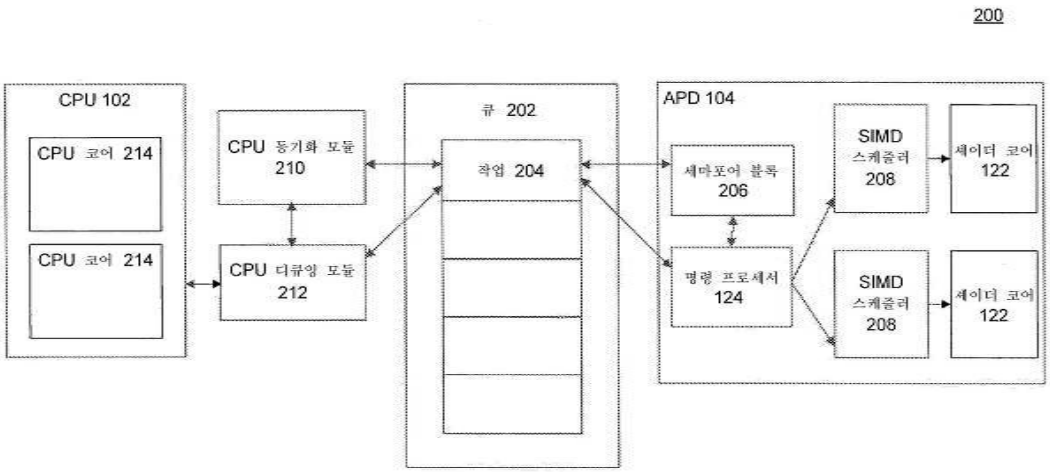
도면1a


100
111
애플리케이션
102
CPU
108
운영 시스템
시스템 메모리 106
140
메모리 제어기
118
TLB
116
IOMMU
125
명령 버퍼
120
선취 로직
128
HWS
146
인터럽트 생성기
150
실행리스트 제어기
152
활성 리스트
154
처리 제어 블록
126
디스패치 제어기 (1)
124
명령 프로세서 (1)
122
셰이더 코어
148
인터럽트 제어기
디스패치 제어기 (n)
명령 프로세서 (n)
110
커널 모드 드라이버
112
SWS
136
쓰레드/웨이브프론트
104
가속 처리 디바이스 (APD)
130
메모리
109

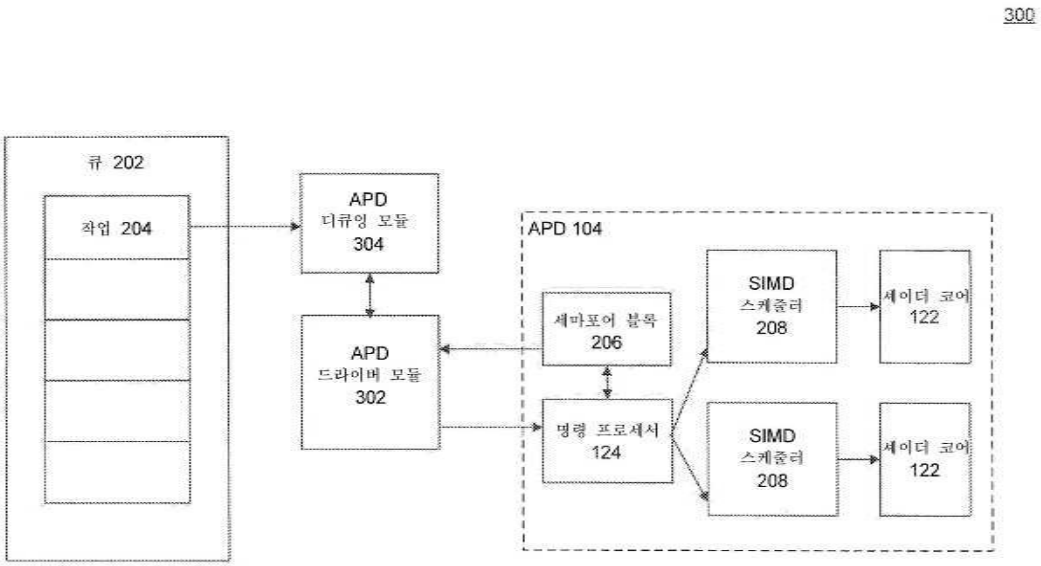
도면1b


104
172
164
124
호스트
CP 입력 0
124 a
그래픽 파이프라인
VGT
164 a
PA
164 b
SC
164 c
162
166
디스패치
제어기 입력 0
HS,
TE,
DS,
GS
VS
CS
PS
SX/RB
176
130
메모리
168
CP 입력 1
124 b
디스패치
제어기 입력 1
셰이더 코어
(공유 자원)
RW
L2 캐시
170
CP 입력 n
124 c
디스패치
제어기 입력 n
연산 파이프라인
160
122
HWS
탑 레벨 제어
128

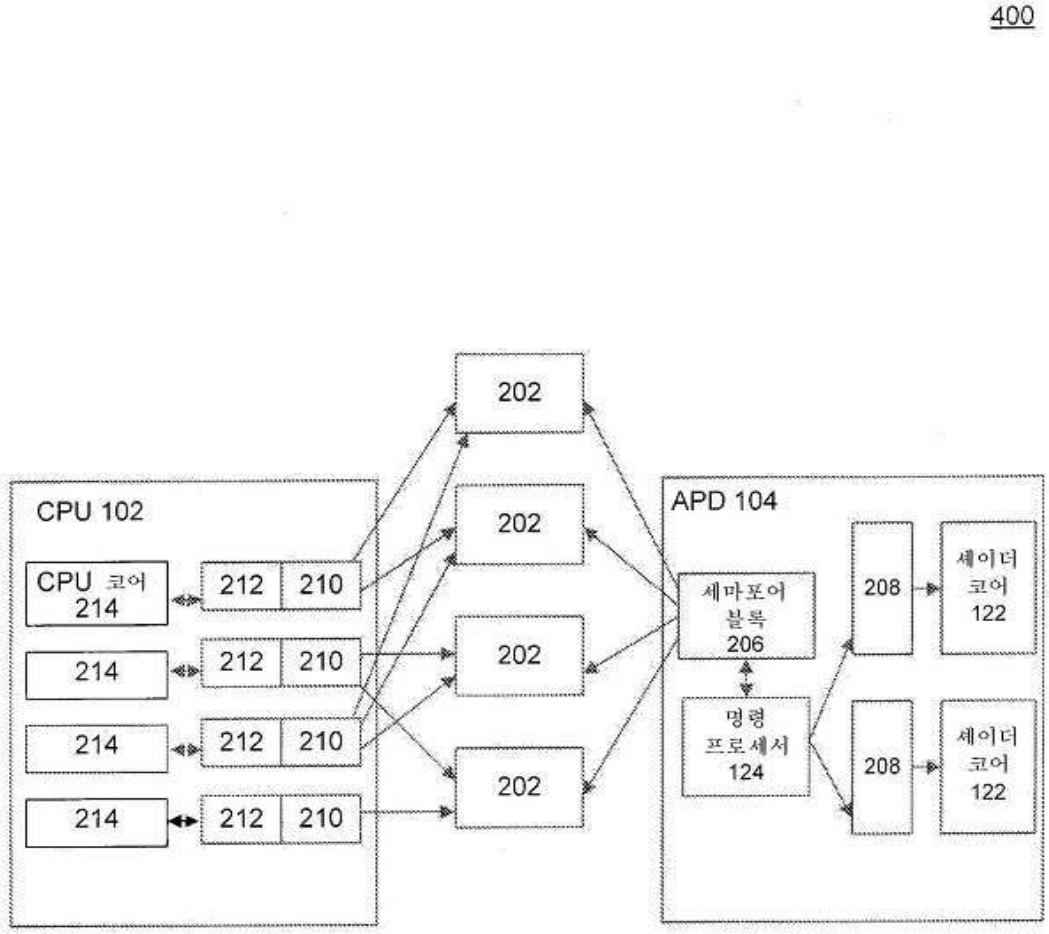
도면2


200
CPU 102
CPU 코어 214
CPU 코어 214
CPU 동기화 모듈
210
CPU 디큐잉 모듈
212
큐 202
작업 204
APD 104
세마포어 블록
206
명령 프로세서
124
SIMD
스케줄러
208
SIMD
스케줄러
208
셰이더 코어
122
셰이더 코어
122

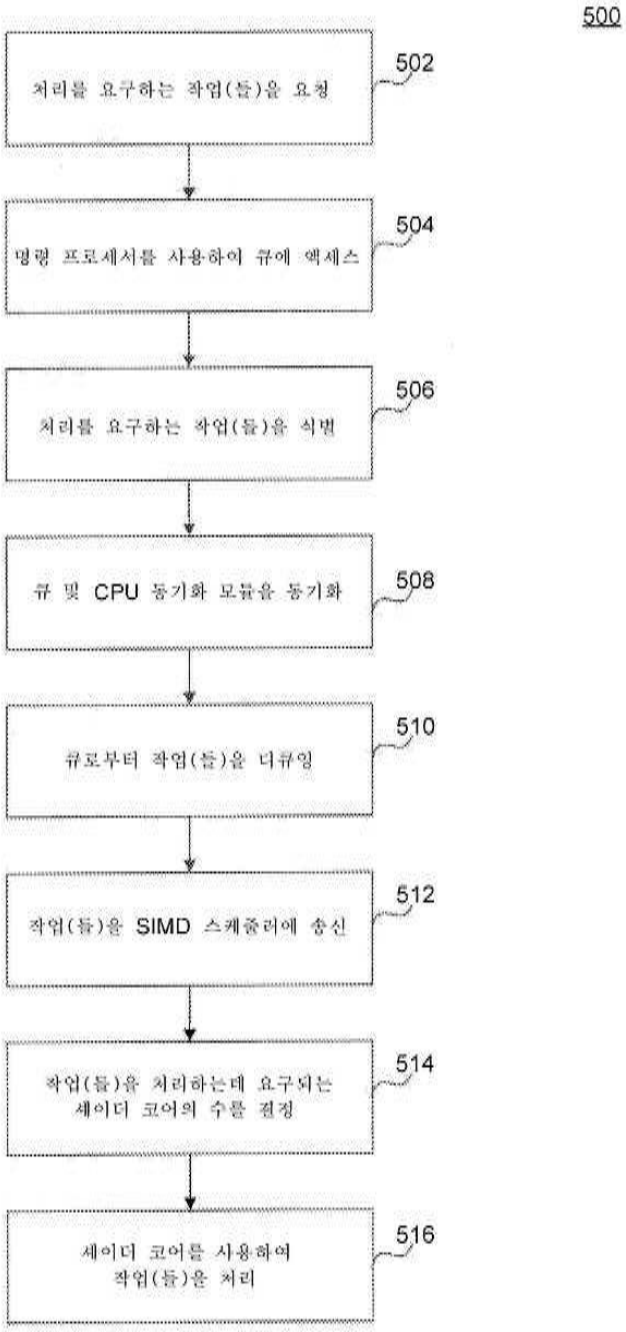
도면3


300
큐 202
작업 204
APD
디큐잉 모듈
304
APD
드라이버 모듈
302
APD 104
세마포어 블록
206
명령 프로세서
124
SIMD
스케줄러
208
SIMD
스케줄러
208
셰이더 코어
122
셰이더 코어
122

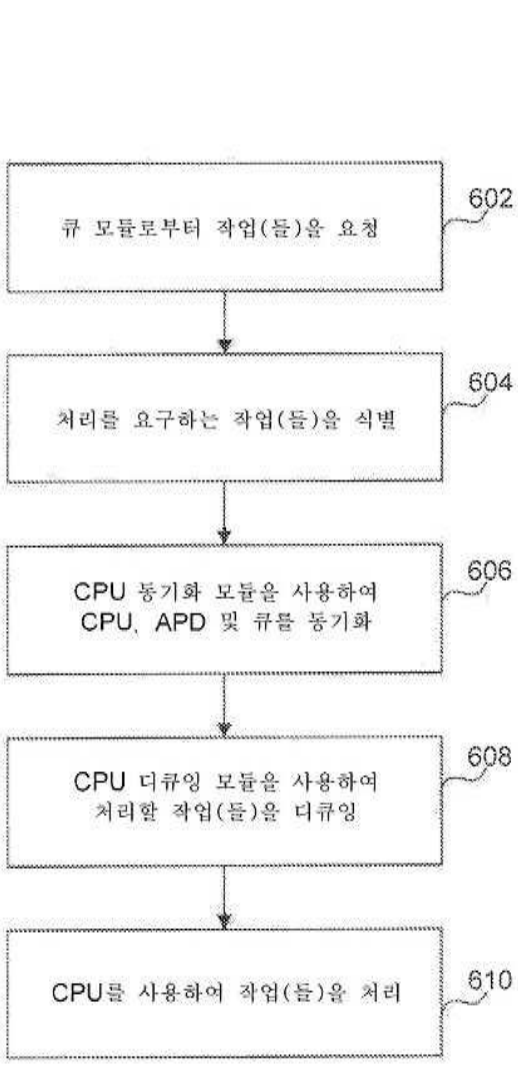
도면4


400
202
202
202
202
CPU 102
CPU 코어
214
214
214
214
212
210
212
210
212
210
212
210
APD 104
세마포어
블록
206
명령
프로세서
124
208
208
셰이더
코어
122
셰이더
코어
122

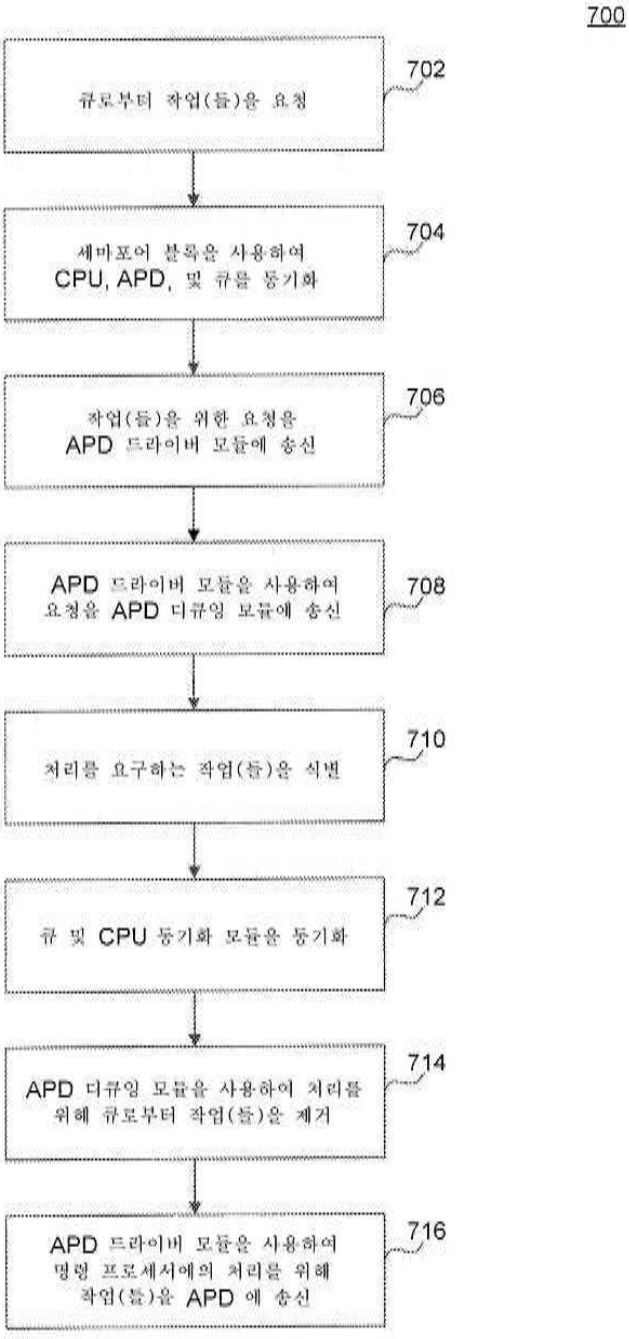
도면5


500
처리를 요구하는 작업(들)을 요청
502
명령 프로세서를 사용하여 큐에 액세스
504
처리를 요구하는 작업(들)을 식별
506
큐 및 CPU 동기화 모듈을 동기화
508
큐로부터 작업(들)을 디큐잉
510
작업(들)을 SIMD 스케줄러에 송신
512
작업(들)을 처리하는데 요구되는 쉐이더 코어의 수를 결정
514
쉐이더 코어를 사용하여 작업(들)을 처리
516

도면6


600
큐 모듈로부터 작업(들)을 요청
602
처리를 요구하는 작업(들)을 식별
604
CPU 동기화 모듈을 사용하여
CPU, APD 및 큐를 동기화
606
CPU 디큐잉 모듈을 사용하여
처리할 작업(들)을 디큐잉
608
CPU를 사용하여 작업(들)을 처리
610

도면7


700
큐로부터 작업(들)을 요청
702
세마포어 블록을 사용하여
CPU, APD, 및 큐를 동기화
704
작업(들)을 위한 요청을
APD 드라이버 모듈에 송신
706
APD 드라이버 모듈을 사용하여
요청을 APD 디큐잉 모듈에 송신
708
처리를 요구하는 작업(들)을 식별
710
큐 및 CPU 동기화 모듈을 동기화
712
APD 디큐잉 모듈을 사용하여 처리를
위해 큐로부터 작업(들)을 제거
714
APD 드라이버 모듈을 사용하여
명령 프로세서에의 처리를 위해
작업(들)을 APD 에 송신
716