(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2018년11월27일
(11) 등록번호 10-1922681
(24) 등록일자 2018년11월21일

(51) 국제특허분류(Int. Cl.)
G06F 9/50 (2018.01)
(21) 출원번호 10-2014-7017104
(22) 출원일자(국제) 2012년12월14일
심사청구일자 2017년12월13일
(85) 번역문제출일자 2014년06월20일
(65) 공개번호 10-2014-0101384
(43) 공개일자 2014년08월19일
(86) 국제출원번호 PCT/US2012/069836
(87) 국제공개번호 WO 2013/090773
국제공개일자 2013년06월20일
(30) 우선권주장
13/325,286 2011년12월14일 미국(US)
(56) 선행기술조사문헌
US20110115802 A1*
(뒷면에 계속)
전체 청구항 수 : 총 12 항

(73) 특허권자
어드밴스드 마이크로 디바이시즈, 인코포레이티드
미국 캘리포니아 95054 산타 클라라 어거스틴 드라이브 2485
(72) 발명자
하토그 로버트 스콧
미국 플로리다 34786 윈더미어 저스트 어 미어 코트 3326
레더 마크
미국 캘리포니아 95032 로스 가토스 몽클레어 로드 265
(뒷면에 계속)
(74) 대리인
박장원

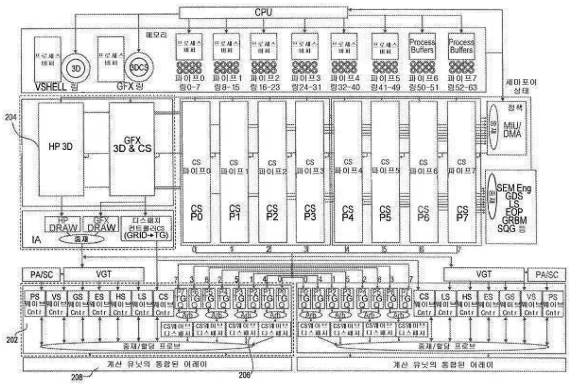
심사관 : 유진태

(54) 발명의 명칭 **셰이더 코어에서 셰이더 자원 할당을 위한 정책**

(57) 요 약

가속화된 프로세싱 디바이스 내 우선순위를 결정하는 방법이 제공된다. 가속화된 프로세싱 디바이스는 소정 기준에 따라 프로세싱되는 계산 파이프라인 큐를 포함한다. 큐는 우선순위 특성에 기반하여 선택되고 선택된 큐는 시간 퀀텀이 경과할 때까지 또는 더 높은 우선순위를 갖는 큐가 프로세싱에 이용가능하게 될 때까지 프로세싱된다.

대 표 도 - 도2



(72) 발명자

맨토 마이클

미국 플로리다 32825 올란도 피나 드라이브 1620

매커리 렉스

미국 플로리다 32765 오베이도 서머 오크스 코트 758

누스바움 세바스티앙

미국 메사추세츠 02420 렉싱톤 파이퍼 레인 99

로저스 필립 제이.

미국 메사추세츠 01463 페퍼렐 힐드 스트리트 158

데일러 랄프 클레이

미국 플로리다 32724 디랜드 록웰 하이츠 드라이브 1548

워렐 토마스

미국 텍사스 78745 오스틴 브로큰 바우 패스 4800

(56) 선행기술조사문헌

US20110154346 A1*

US20060085600 A1

US6781956 B1

WO2011053891 A1

*는 심사관에 의하여 인용된 문헌

명 세 서

청구범위

청구항 1

계산 파이프라인들을 포함하는 가속화된 프로세싱 디바이스(APD) 내 웨이브프론트들을 실행하는 방법으로서,

제2 계산 파이프라인과 디스패치 컨트롤러를 공유하는 제1 계산 파이프라인에 할당된 제1 파이프라인 큐에 의해 특정되는 웨이브프론트들의 제1 세트를, 셰이더 코어 상에서 실행하는 단계와;

정지 조건이 발생했는지를 결정하는 단계와;

상기 정지 조건에 응답하여, 상기 셰이더 코어 상에서 실행되는 웨이브프론트들을 스케줄링하기 위해 상기 제2 계산 파이프라인과 연관된 큐들 중에서 제2 파이프라인 큐를 선택하는 단계와;

새로운 대기중인 파이프라인 큐를 선정하기 위해, 선택된 새로운 파이프라인 큐와, 상기 제2 계산 파이프라인에 할당된 제3 파이프라인 큐 사이에서 중재(arbitration)를 실행하는 단계와; 그리고

상기 셰이더 코어 상에서 실행하기 위해 상기 새로운 대기중인 파이프라인 큐에서 웨이브프론트를 스케줄링하는 단계를 포함하는 것을 특징으로 하는

웨이브프론트들을 실행하는 방법.

청구항 2

제1항에 있어서,

상기 새로운 대기중인 파이프라인 큐에서 웨이브프론트를 스케줄링하는 단계는 상기 제1 파이프라인 큐에서 선취(preempt)하는 단계를 포함하는 것을 특징으로 하는

웨이브프론트들을 실행하는 방법.

청구항 3

제2항에 있어서,

상기 제1 파이프라인 큐에서 선취하는 단계는 상기 제1 파이프라인 큐에서 컨텍스트 스위칭 동작을 수행하는 단계를 포함하는 것을 특징으로 하는

웨이브프론트들을 실행하는 방법.

청구항 4

제1항에 있어서,

상기 선택된 새로운 파이프라인 큐와, 상기 제2 계산 파이프라인에 할당된 제3 파이프라인 큐 사이에서 중재를 실행하는 단계는, 상기 제1 계산 파이프라인 및 상기 제2 계산 파이프라인의 상대적 우선순위를 결정하는 단계를 더 포함하는 것을 특징으로 하는

웨이브프론트들을 실행하는 방법.

청구항 5

제4항에 있어서,

상기 제1 계산 파이프라인과 상기 제2 계산 파이프라인 사이의 동일한 우선순위는 가장 먼저 발행된 회로를 사용하여 해결되는 것을 특징으로 하는

웨이브프론트들을 실행하는 방법.

청구항 6

제5항에 있어서,

상기 가장 먼저 발행된 회로는 토템 폴 회로인 것을 특징으로 하는

웨이브프론트들을 실행하는 방법.

청구항 7

제5항에 있어서,

상기 정지 조건은,

상기 제1 파이프라인 큐보다 높은 우선순위를 갖는 상기 제2 파이프라인 큐가 대기 상태가 되는 것,

상기 제1 파이프라인 큐의 제1 퀀텀 듀레이션(quantum duration)이 경과하고, 상기 제1 파이프라인 큐와 동일한 우선순위를 갖는 상기 제2 파이프라인 큐가 대기 상태가 되는 것, 그리고

상기 제1 파이프라인 큐의 제1 퀀텀이 디스에이블되고, 상기 제1 파이프라인 큐의 웨이브프론트가 다른 큐 우선순위 레지스터에 기입되고, 다른 파이프라인 큐가 상기 제1 파이프라인 큐와 동일한 우선순위를 갖는 것

중 하나를 포함하는 것을 특징으로 하는

웨이브프론트들을 실행하는 방법.

청구항 8

가속화된 프로세싱 디바이스(APD)로서,

복수의 계산 파이프라인들과;

셰이더 코어와; 그리고

아비터(arbitar)를 포함하고,

상기 셰이더 코어는, 상기 복수의 계산 파이프라인들 중 제2 계산 파이프라인과 디스패치 컨트롤러를 공유하는 상기 복수의 계산 파이프라인들 중 제1 계산 파이프라인에 할당된 제1 파이프라인 큐에 의해 특정되는 웨이브프론트들의 제1 세트를 실행하고,

상기 아비터는,

정지 조건이 발생했는지를 결정하고,

상기 정지 조건에 응답하여, 상기 셰이더 코어 상에서 실행되는 웨이브프론트들을 스케줄링하기 위해 상기 제2 계산 파이프라인과 연관된 큐들 중에서 제2 파이프라인 큐를 선택하고,

새로운 대기중인 파이프라인 큐를 선정하기 위해, 선택된 새로운 파이프라인 큐와, 상기 제2 계산 파이프라인에 할당된 제3 파이프라인 큐 사이에서 중재를 실행하고, 그리고

상기 셰이더 코어 상에서 실행하기 위해 상기 새로운 대기중인 파이프라인 큐에서 웨이브프론트를 스케줄링하는 것을 특징으로 하는

가속화된 프로세싱 디바이스.

청구항 9

제8항에 있어서,

상기 아비터는, 또한, 상기 제1 파이프라인 큐를 선취함으로써 상기 새로운 대기중인 파이프라인 큐에서 웨이브프론트를 스케줄링하는 것을 특징으로 하는

가속화된 프로세싱 디바이스.

청구항 10

제9항에 있어서,

상기 아비터가 상기 계산 파이프라인들 각각의 상대적 우선순위를 결정하도록 또한 구성될 때, 상기 제1 계산 파이프라인과 상기 제2 계산 파이프라인 사이의 동일한 우선순위를 해결하도록 구성된 회로를 더 포함하는 것을 특징으로 하는

가속화된 프로세싱 디바이스.

청구항 11

제10항에 있어서,

상기 회로는 토템 폴 회로인 것을 특징으로 하는

가속화된 프로세싱 디바이스.

청구항 12

제10항에 있어서,

상기 정지 조건은,

상기 제1 파이프라인 큐보다 높은 우선순위를 갖는 상기 제2 파이프라인 큐가 대기 상태가 되는 것,

상기 제1 파이프라인 큐의 제1 퀀텀 듀레이션이 경과하고, 상기 제1 파이프라인 큐와 동일한 우선순위를 갖는 상기 제2 파이프라인 큐가 대기 상태가 되는 것, 그리고

상기 제1 파이프라인 큐의 제1 퀀텀이 디스에이블되고, 상기 제1 파이프라인 큐의 웨이브프론트가 다른 큐 우선순위 레지스터에 기입되고, 다른 파이프라인 큐가 상기 제1 파이프라인 큐와 동일한 우선순위를 갖는 것

중 하나를 포함하는 것을 특징으로 하는

가속화된 프로세싱 디바이스.

청구항 13

삭제

청구항 14

삭제

청구항 15

삭제

청구항 16

삭제

청구항 17

삭제

청구항 18

삭제

청구항 19

삭제

청구항 20

삭제

청구항 21

삭제

청구항 22

삭제

발명의 설명

기 술 분 야

[0001] 본 발명은 일반적으로는 컴퓨팅 시스템에 관한 것이다. 더 구체적으로는, 본 발명은 다중 파이프라인 입력 간 그래픽 프로세싱 유닛 자원을 할당하기 위한 중재 정책에 관한 것이다.

배 경 기 술

[0002] 단위 출력 및/또는 비용당 GPU(그래픽 프로세싱 유닛)의 전형적 성능에 기인하여 일반 계산에 GPU를 사용하려는 소망이 최근 더더욱 현저하게 되었다. 일반적으로 GPU에 대한 계산 능력은 대응하는 중앙 프로세싱 유닛(CPU) 플랫폼의 능력을 초과하는 레이트로 성장하였다. 이러한 성장은, 모바일 컴퓨팅 시장 및 그 필요한 지원 서버/기업 시스템의 폭발적 증가와 결합하여, 특정 품질의 소망 사용자 경험을 제공하도록 사용되어 왔다. 결과적으로, 데이터 병렬 컨텐트를 갖는 작업부하를 실행하기 위해 CPU 및 GPU의 조합된 사용은 볼륨 기술이 되고 있다.

[0003] 그렇지만, 전통적으로 GPU는 그래픽의 가속에만 이용가능한 제약된 프로그래밍 환경에서 동작하여 왔다. 이들 제약은 GPU가 CPU만큼 풍부한 프로그래밍 에코시스템을 갖지 않았다는 사실로부터 유발되었다. 그래서, 그 사용은, 그래픽 및 비디오 애플리케이션 프로그래밍 인터페이스(API)를 다루는데 이미 익숙한, 2차원(2D) 및 3차원(3D) 그래픽 및 소수의 최첨단 멀티미디어 애플리케이션으로 대부분 한정되어 왔다.

발명의 내용

해결하려는 과제

[0004] 멀티-벤더 지원형 OpenCL® 및 DirectCompute®, 표준 API 및 지원 툴의 출현으로, 전통적 애플리케이션에서 GPU의 제한은 전통적 그래픽 너머로 미치게 되었다. OpenCL 및 DirectCompute가 유망한 시작이기는 하지만, CPU 및 GPU의 조합이 대부분의 프로그래밍 태스크에 CPU만큼 유동적으로 사용되도록 하는 환경 및 에코시스템을 생성하는데 남아있는 난관이 많이 있다.

[0005] 기존 컴퓨팅 시스템은 흔히 다중 프로세싱 디바이스를 포함한다. 예를 들어, 일부 컴퓨팅 시스템은 단일 칩 패키지에 또는 별개의 칩들 상에 CPU 및 GPU 둘 다를 포함할 수 있다(예를 들어, CPU는 마더보드에 위치할 수 있고 GPU는 그래픽 카드에 위치할 수 있다). 그렇지만, 이들 배열 둘 다는 여전히 - 모두 전력 소모를 최소화하면서 (i) 별개의 메모리 시스템, (ii) 효율적 스케줄링, (iii) 프로세스 간 서비스 품질(QoS) 보증 제공, (iv) 프로그래밍 모델, 및 (v) 다중 표적 명령어 세트 아키텍처(ISA)로의 컴파일링과 연관된 중요한 도전과제를 포함하고 있다.

[0006] 예를 들어, 개별 칩 배열은 시스템 및 소프트웨어 아키텍처가 각각의 프로세서에 대해 칩 대 칩 인터페이스를 이용하여 메모리에 액세스하도록 강제한다. 이들 외부 인터페이스(예를 들어, 칩 대 칩)는 이종 프로세서를 협력시키는데 메모리 레이턴시 및 전력 소모에 악영향을 미치는 한편, 별개 메모리 시스템(즉, 별개 어드레스 공간) 및 드라이버 관리형 공유 메모리는 미세 분담에 허용가능하지 않게 되는 오버헤드를 생성한다.

[0007] 개별 및 단일 칩 배열 둘 다 실행을 위해 GPU에 보내질 수 있는 커맨드의 유형을 제한할 수 있다. 예로써, 계산 커맨드(예를 들어, 물리 또는 인공 지능 커맨드)는 흔히 실행을 위해 GPU에 보내질 수 없다. 이러한 제한은 CPU가 이들 계산 커맨드에 의해 수행된 연산의 결과를 비교적 빨리 요구할 수 있기 때문에 존재한다. 그렇지만, 이들 커맨드가 먼저 실행될 다른 이전에-발행된 커맨드를 위해 줄서서 기다려야 할 수 있다는 사실 및 현재 시스템에서 GPU에 작업을 디스패치하는 높은 오버헤드 때문에, GPU에 계산 커맨드를 보냄으로써 유발된 레이턴시는 흔히 받아들여질 수 없다.

[0008] 전통적 GPU가 일부 계산 커맨드를 효율적으로 실행할 수 없으면, 그때 커맨드는 CPU 내에서 실행되어야 한다. 커맨드를 CPU 상에서 실행해야 하는 것은 CPU 상에 프로세싱 부담을 증가시키고 전반적 시스템 성능을 저해할 수 있다.

[0009] GPU가 계산 분담을 위해 뛰어난 기회를 제공한다 하더라도, 전통적 GPU는 일부 다중-프로세서 환경에서의 효율적 동작을 위해 소망되는 시스템-소프트웨어-구동 프로세스 관리에 적합하지 않을 수 있다. 이들 제한은 수개의 문제를 만들어낼 수 있다.

[0010] 예를 들어, 프로세스가 효율적으로 식별 및/또는 선취될 수 없으므로, 로그 프로세스는 제멋대로의 시간량 동안 GPU 하드웨어를 점유할 수 있다. 다른 경우에 있어서, 하드웨어를 컨텍스트 스위치 오프할 수 있는 능력이 심하게 제약을 받는다 - 매우 성기게 그리고 프로그램의 실행에 있어서 매우 제한된 포인트 세트에서만 일어난다. 이러한 제약은 프로세스를 복원 및 재개하는데 필요한 아키텍처 및 마이크로아키텍처 상태를 절약하는 것이 지원되지 않기 때문에 존재한다. 정밀 예외에 대한 지원 결여는 폴팅된 작업이 컨텍스트 스위칭 아웃되어 추후 포인트에서 복원되는 것을 방지하여, 폴팅된 스레드가 하드웨어 자원을 점유하고 폴트 취급 동안 유휴로 있으므로 더 낮은 하드웨어 사용의 결과를 초래한다.

[0011] 중재는 컴퓨터 시스템 내에서 2개의 다른 레벨로 일어난다. 하나의 레벨은 GPU 계산 파이프라인의 프론트 엔드에서 무슨 작업이 구동되고 있는지에 관한 것이다. 다른 하나의 레벨은 공유된 자원의 이용에 관한 것이다. 동시에 실행되고 있는 태스크가 다수 있기 때문에, 이들 태스크는 우선순위결정이 되어야 한다. 그래서, 공유된 자원이 어떻게 이용될지 결정하기 위한 결정이 요구된다. 예를 들어, 태스크가 디스패치 파이프라인의 시작에 도착하여 셰이더 코어로 이동할 때 그것들이 어떻게 우선순위결정될 것인가.

과제의 해결 수단

[0012] 그래서 필요로 되는 것은 시스템이 다중 계산 파이프라인을 갖는 경우 중재 정책을 해결하는 개선된 중재 방법 및 시스템이다.

[0013] GPU, 가속화된 프로세싱 유닛(APU), 및 그래픽 프로세싱 유닛의 범용 사용(GPGPU)은 이 분야에서 일반적으로 사용되는 용어이지만, "가속화된 프로세싱 디바이스(APD)"라는 표현은 더 넓은 표현이라고 생각된다. 예를 들어, APD는 종래 CPU, 종래 GPU, 및/또는 그 조합과 같은 자원에 관하여 가속화된 방식으로 그래픽 프로세싱 태스크, 데이터 병렬 태스크, 또는 네스팅된 데이터 병렬 태스크를 가속화하는 것과 연관된 그들 기능 및 계산을 수행하는 소프트웨어 및/또는 하드웨어의 어느 협력 모음을 지칭한다.

[0014] 본 발명의 일 실시예는 계산 파이프라인을 포함하는 APD 내 우선순위를 결정하는 방법을 제공한다. 그 방법은 소정 기준에 따라 계산 파이프라인의 각각 내 계산 파이프라인 프로세싱 큐로부터 제1 큐 및 제2 큐를 선택하는 단계 및 우선순위 기준에 따라 프로세싱하기 위해 제1 및 제2 큐 중 하나를 선택하는 단계를 포함한다. 선택된 큐는 시간 퀀텀의 경과가 일어날 때까지 또는 더 높은 우선순위를 갖는 큐가 이용가능하게 될 때까지 프로세싱된다.

[0015] 본 발명의 추가적 특징 및 이점과 더불어, 본 발명의 다양한 실시예의 구조 및 동작은 수반 도면을 참조하여 아래에 더 상세하게 설명된다. 본 발명은 여기에서 설명되는 특정 실시예로 한정되는 것은 아니다. 그러한 실시예는 여기에서 예시의 목적으로만 제시된다. 부가적 실시예는 여기에 포함된 가르침에 기반하여 관련 업계(들)의 당업자에게 명백할 것이다.

도면의 간단한 설명

[0016] 여기에 편입되어 명세서의 일부를 형성하는 수반 도면은 본 발명을 예시하며, 그 설명과 함께, 더욱 본 발명의 원리를 설명하고 관련 업계의 당업자가 본 발명을 하고 사용 가능하게 하는 역할을 한다. 본 발명의 다양한 실시예는 도면을 참조하여 아래에 설명되며, 유사한 참조 숫자는 곳곳에서 유사한 구성요소를 지칭하는데 사용된다.

도 1a는 본 발명의 실시예에 따른 프로세싱 시스템의 예시적 블록 선도;

도 1b는 도 1a에 도시된 APD의 예시적 블록 선도;

도 2는 도 1b의 APD의 더 상세한 예시적 블록 선도;

도 3은 도 2에 예시된 계산 파이프라인의 더 상세한 예시적 블록 선도;

도 4는 본 발명의 실시예에 따른 하드웨어 기술자 큐의 예시도;

도 5는 본 발명의 실시예를 실시하는 전형적 방법의 흐름도;

도 6은 본 발명의 실시예에 따른 전형적 방법의 예시도; 및

도 7은 본 발명의 실시예에 따른 전형적 방법의 다른 태양의 예시도.

발명을 실시하기 위한 구체적인 내용

[0017] 이하의 상세한 설명에 있어서, "하나의 실시예", "일 실시예", "예시적 실시예" 등의 언급은 설명되는 실시예가 특정 특징, 구조 또는 특성을 포함하지만, 모든 실시예가 그 특정 특징, 구조 또는 특성을 반드시 포함하지는 않을 수 있음을 나타낸다. 더욱, 그러한 문구는 반드시 동일 실시예를 지칭하는 것은 아니다. 더욱, 일 실시예와 연관하여 특정 특징, 구조 또는 특성이 설명될 때, 명시적으로 설명되든 아니든 다른 실시예와 연관하여 그러한 특징, 구조 또는 특성에 영향을 미치는 것이 당업자의 지식 내에 있는 것으로 된다.

[0018] "본 발명의 실시예"라는 용어는 본 발명의 모든 실시예가 논의되는 특징, 이점 또는 동작 모드를 포함하는 것을 요구하지는 않는다. 대체 실시예가 본 발명의 범위로부터 벗어남이 없이 고안될 수 있고, 본 발명의 주지의 엘리먼트는 상세히 설명되지 않을 수 있거나 본 발명의 관련 상세를 모호하게 하지 않도록 생략될 수 있다. 부가적으로, 여기서 사용되는 용어는 특정 실시예를 설명하는 목적을 위한 것일 뿐이고 본 발명을 한정하려는 의도는 아니다. 예를 들어, 여기서 사용되는 바와 같이, 단수 형태는, 맥락이 명확하게 다르게 나타내지 않는 한, 복수 형태 역시 포함하려는 의도이다. 용어 "구성된다", "구성되는", "포함한다" 및/또는 "포함하는"은, 여기서 사용될 때, 서술된 특징, 정수, 단계, 동작, 엘리먼트 및/또는 컴포넌트의 존재를 특정하지만, 하나 이상의 다른 특징, 정수, 단계, 동작, 엘리먼트, 컴포넌트 및/또는 그 그룹의 존재 또는 부가를 못하게 하지는 않음을 더욱 이해할 것이다.

[0019] 도 1a는 CPU(102) 및 APD(104)를 포함하는 통합된 컴퓨팅 시스템(100)의 전형적 예시이다. CPU(102)는 하나 이상의 단일 또는 멀티 코어 CPU를 포함할 수 있다. 본 발명의 일 실시예에 있어서, 시스템(100)은, 통합된 프로그래밍 및 실행 환경을 제공하도록 CPU(102) 및 APD(104)를 조합하여, 단일 실리콘 다이 또는 패키지 상에 형성된다. 이러한 환경은 APD(104)가 일부 프로그래밍 태스크에 대해 CPU(102)만큼 유동적으로 사용될 수 있게 한다. 그렇지만, CPU(102) 및 APD(104)가 단일 실리콘 다이 상에 형성되는 것이 본 발명의 절대적 요건은 아니다. 일부 실시예에서는, 그것들이 별개로 형성되거나 동일 또는 다른 기판 상에 장착되는 것이 가능하다.

[0020] 하나의 예에 있어서, 시스템(100)은 또한 메모리(106), OS(OS)(108), 및 통신 기반구조(109)를 포함한다. OS(108) 및 통신 기반구조(109)는 아래에서 더 상세하게 논의된다.

[0021] 시스템(100)은 또한 커널 모드 드라이버(KMD)(110), 소프트웨어 스케줄러(SWS)(112), 및 입/출력 메모리 관리 유닛(IOMMU)과 같은 메모리 관리 유닛(116)을 포함한다. 시스템(100)의 컴포넌트는 하드웨어, 펌웨어, 소프트웨어 또는 그 어느 조합으로 구현될 수 있다. 당업자는 시스템(100)이 도 1a에 도시된 실시예에 도시된 것에 부가하여 또는 그와는 다른 하나 이상의 소프트웨어, 하드웨어 및 펌웨어 컴포넌트를 포함할 수 있음을 인식할 것이다.

[0022] 하나의 예에 있어서, KMD(110)와 같은 드라이버는 전형적으로 하드웨어가 접속하는 컴퓨터 버스 또는 통신 서브시스템을 통해 디바이스와 통신한다. 호출 프로그램이 드라이버 내 루틴을 인보크할 때, 드라이버는 디바이스에 커맨드를 발행한다. 디바이스가 데이터를 다시 드라이버에 보내고 나면, 드라이버는 원래 호출 프로그램 내 루틴을 인보크할 수 있다. 하나의 예에 있어서, 드라이버는 하드웨어-종속적이고 운영-체제-특정적이다. 그것들은 통상 어느 필요한 비동기식 시간-종속적 하드웨어 인터페이스에 요구되는 인터럽트 취급을 제공한다. 디바이스 드라이버는, 특히 현대 윈도우즈 플랫폼 상에서, 커널-모드(링 0)로 또는 사용자-모드(링 3)로 실행될 수 있다.

[0023] 사용자 모드로 드라이버를 실행하는 이점은 개선된 안정성인데, 좋지 않게 쓰인 사용자 모드 디바이스 드라이버는 커널 메모리를 덮어씀으로써 시스템과 충돌할 수 없기 때문이다. 다른 한편으로, 사용자/커널-모드 트랜지션은 통상 상당한 성능 오버헤드를 부과하고, 그로써 낮은 레이턴시 및 높은 스루풋 요건에 대하여 사용자 모드-드라이버를 금지한다. 커널 공간은 시스템 호출의 사용을 통해서만 사용자 모듈에 의해 액세스될 수 있다. UNIX 쉘 또는 다른 그래픽 사용자 인터페이스(GUI) 기반 애플리케이션과 같은 최종 사용자 프로그램은 사용자 공간의 일부분이다. 이들 애플리케이션은 커널 지원된 기능을 통해 하드웨어와 대화한다.

[0024] CPU(102)는 제어 프로세서, 필드 프로그램가능한 게이트 어레이(FPGA), 주문형 반도체(ASIC), 또는 디지털 신호 프로세서(DSP) 중 하나 이상(도시하지 않음)을 포함할 수 있다. CPU(102)는, 예를 들어, 컴퓨팅 시스템(100)의 동작을 제어하는, OS(108), KMD(110), SWS(112) 및 애플리케이션(111)을 포함하는 제어 로직을 실행한다. 이러한 예시적 실시예에 있어서, CPU(102)는, 하나의 실시예에 의하면, 예를 들어, CPU(102) 및 APD(104)와 같은 다

른 프로세싱 자원을 가로질러 그 애플리케이션과 연관된 프로세싱을 분산시킴으로써 애플리케이션(111)의 실행을 개시 및 제어한다.

[0025] APD(104)는, 다른 것들 중에서도, 예를 들어 특히 병렬 프로세싱에 적합할 수 있는 그래픽 동작 및 다른 동작과 같이 선택된 기능에 대해 커맨드 및 프로그램을 실행한다. 일반적으로, APD(104)는 픽셀 동작, 기하구조 계산, 및 디스플레이할 이미지 렌더링과 같이 그래픽 파이프라인 동작을 실행하는데 빈번하게 사용될 수 있다. 본 발명의 다양한 실시예에 있어서, APD(104)는 CPU(102)로부터 수신된 커맨드 또는 명령어에 기반하여 계산 프로세싱 동작을 또한 실행할 수 있다.

[0026] 예를 들어, 커맨드는 ISA에 정의되지 않고 통상 하드웨어의 고유 피스 또는 주어진 ISA로부터의 명령어 세트에 의해 성취되는 특수 명령어로서 생각될 수 있다. 커맨드는 디스패치 프로세서, CP 또는 네트워크 컨트롤러와 같은 특수 프로세서에 의해 실행될 수 있다. 다른 한편으로, 명령어는 예를 들어 컴퓨터 아키텍처 내 프로세서의 단일 동작으로 생각될 수 있다. 하나의 예에 있어서, 2개의 ISA 세트를 사용할 때, 일부 명령어는 x86 프로그램을 실행하도록 사용되고 일부 명령어는 APU/APD 계산 유닛 상에서 커널을 실행하도록 사용된다.

[0027] 예시적 실시예에 있어서, CPU(102)는 선택된 커맨드를 APD(104)에 송신한다. 이들 선택된 커맨드는 병렬 실행을 처리할 수 있는 그래픽 커맨드 및 다른 커맨드를 포함할 수 있다. 또한 계산 프로세싱 커맨드를 포함할 수 있는 이들 선택된 커맨드는 CPU(102)와 실질적으로 독립적으로 실행될 수 있다.

[0028] APD(104)는, 국한되는 것은 아니지만, 하나 이상의 단일 명령 다중 데이터(SIMD) 프로세싱 코어와 같은 그 자신의 계산 유닛(도시하지 않음)을 포함할 수 있다. 여기서 지칭되는 바와 같이, SIMD는 그 자신의 데이터 및 공유 프로그램 카운터를 각각 갖는 다중 프로세싱 엘리먼트 상에서 동시에 커널이 실행되는 수학 파이프라인 또는 프로그래밍 모델이다. 모든 프로세싱 엘리먼트는 정확히 똑같은 명령어 세트를 실행한다. 예측의 사용은 작업-항목이 각각의 발행된 커맨드에 대해 참가하게 또는 하지 않게 할 수 있다.

[0029] 하나의 예에 있어서, 각각의 APD(104)는 계산 유닛은 하나 이상의 스칼라 및/또는 벡터 부동-소수점 유닛 및/또는 산술 및 로직 유닛(ALU)을 포함할 수 있다. APD 계산 유닛은 또한 역-제곱근 유닛 및 사인/코사인 유닛과 같이 특수 목적 프로세싱 유닛(도시하지 않음)을 포함할 수 있다. 하나의 예에 있어서, APD 계산 유닛은 여기서는 일괄하여 셰이더 코어(122)라고 지칭된다.

[0030] 하나 이상의 SIMD를 갖는 것은, 일반적으로, APD(104)를 그래픽 프로세싱에서 공통적인 것과 같은 데이터-병렬 태스크의 실행에 이상적으로 적합하게 한다.

[0031] 픽셀 프로세싱과 같은 일부 그래픽 파이프라인 동작, 및 다른 병렬 계산 동작은 동일한 커맨드 스트림 또는 계산 커널이 입력 데이터 엘리먼트의 스트림 또는 모음에 수행될 것을 요구할 수 있다. 동일 계산 커널의 각자의 인스턴스 생성은 그러한 데이터 엘리먼트를 병렬로 프로세싱하기 위해 셰이더 코어(122) 내 다중 계산 유닛에서 동시에 실행될 수 있다. 여기서 지칭되는 바와 같이, 예를 들어, 계산 커널은 프로그램 내 선언되고 APU/APD 계산 유닛 상에서 실행되는 명령어를 포함하는 함수이다. 이러한 함수는 또한 커널, 셰이더, 셰이더 프로그램, 또는 프로그램이라고 지칭된다.

[0032] 하나의 예시적 실시예에 있어서, 각각의 계산 유닛(예를 들어, SIMD 프로세싱 코어)은 입중계 데이터를 프로세싱하도록 특정 작업-항목의 각자의 인스턴스 생성을 실행할 수 있다. 작업-항목은 커맨드에 의해 디바이스 상에 인보크된 커널의 병렬 실행의 모음 중 하나이다. 작업-항목은 계산 유닛 상에서 실행하는 작업-그룹의 일부분으로서 하나 이사의 프로세싱 엘리먼트에 의해 실행될 수 있다. 작업-항목은 또한 스레드, 레인 또는 인스턴스라고 지칭될 수 있다.

[0033] 작업-항목은 그 글로벌 ID 및 로컬 ID에 의해 모음 내 다른 실행과 구별된다. 하나의 예에 있어서, 단일 SIMD 엔진 상에서 함께 동시에 실행하는 작업그룹 내 작업-항목의 서브세트는 웨이브프론트(136)라고 지칭될 수 있다. 웨이브프론트의 폭은 하드웨어 SIMD 엔진의 특성이다. 여기서 지칭되는 바와 같이, 작업그룹은 단일 계산 유닛 상에서 실행하는 관련 작업-항목의 모음이다. 그룹 내 작업-항목은 ED일 커널을 실행하고 로컬 메모리 및 작업-그룹 배리어를 공유한다. 작업 그룹은 또한 스레드 그룹 또는 스레드 블록이라고 지칭될 수 있다.

[0034] 작업그룹으로부터의 모든 웨이브프론트는 동일 SIMD 엔진 상에서 프로세싱된다. 웨이브프론트를 가로지르는 명령어는 한번에 하나씩 발행되고, 모든 작업-항목이 동일 제어 흐름을 따를 때, 각각의 작업-항목은 동일 프로그램을 실행한다. 실행 마스크 및 작업-항목 예측은 웨이브프론트 내 발산 제어 흐름을 가능하게 하도록 사용되고, 여기서 각각의 개개의 작업-항목은 실제로는 커널을 통해 고유 코드 경로를 취할 수 있다. 부분적으로 파퓰레이팅된 웨이브프론트는 작업-항목의 완전 세트가 웨이브프론트 시작 시간에 이용가능하지 않을 때 프

로세싱될 수 있다. 또한 웨이브프론트는 워프, 벡터 또는 스레드라고 지칭될 수 있다.

[0035] 커맨드는 웨이브프론트에 대하여 한번에 하나씩 발행될 수 있다. 모든 작업-항목이 동일 제어 흐름을 따를 때, 각각의 작업-항목은 동일 프로그램을 실행할 수 있다. 하나의 예에 있어서, 실행 마스크 및 작업-항목 예측은 발산 제어 흐름을 가능하게 하도록 사용되고, 여기서 각각의 개개의 작업-항목은 실제로는 커널 드라이버를 통해 고유 코드 경로를 취할 수 있다. 부분적 웨이브프론트는 작업-항목의 완전 세트가 시작 시간에 이용가능하지 않을 때 프로세싱될 수 있다. 예를 들어, 셰이더 코어(122)는 미리 결정된 수의 웨이브프론트(136)를 동시에 실행할 수 있고, 각각의 웨이브프론트(136)는 미리 결정된 수의 작업-항목을 포함한다.

[0036] 시스템(100) 내에서, APD(104)는 그래픽 메모리(130)와 같이 그 자신의 메모리를 포함한다. 그래픽 메모리(130)는 APD(104)에서의 계산 동안의 사용을 위한 로컬 메모리를 제공한다. 셰이더 코어(122) 내 개개의 계산 유닛(도시하지 않음)은 그들 자신의 로컬 데이터 스토어(도시하지 않음)를 가질 수 있다. 일 실시예에 있어서, APD(104)는 로컬 그래픽 메모리(130)로의 액세스와 더불어, 메모리(106)로의 액세스도 포함한다. 다른 실시예에 있어서, APD(104)는 메모리(106)와는 별개로 그리고 APD(104)에 직접 부착된 동적 램(DRAM) 또는 다른 그러한 메모리(도시하지 않음)로의 액세스를 포함할 수 있다.

[0037] 도시된 예에 있어서, APD(104)는 또한 하나 또는 "n"개의 커맨드 프로세서(CP)(124)를 포함한다. CP(124)는 APD(104) 내 프로세싱을 제어한다. CP(124)는 또한 메모리(106) 내 커맨드 버퍼(125)로부터 실행될 커맨드를 검색하고 APD(104) 상의 그들 커맨드의 실행을 조정한다.

[0038] 하나의 예에 있어서, CPU(102)는 애플리케이션(111)에 기반한 커맨드를 적합한 커맨드 버퍼(125)에 입력한다. 여기서 지칭되는 바와 같이, 애플리케이션은 CPU 및 APD 내 계산 유닛 상에서 실행할 프로그램 부분들의 조합이다.

[0039] 복수의 커맨드 버퍼(125)는 APD(104) 상에서의 실행을 위해 스케줄링되는 각각의 프로세스로 유지될 수 있다.

[0040] CP(124)는 하드웨어, 펌웨어 또는 소프트웨어, 또는 그 조합으로 구현될 수 있다. 일 실시예에 있어서, CP(124)는 스케줄링 로직을 포함하는 로직을 구현하기 위해 마이크로코드를 갖는 축소된 명령어 세트 컴퓨터(RISC) 엔진으로서 구현된다.

[0041] APD(104)는 또한 하나 또는 (n)개의 디스패치 컨트롤러(DC)(126)를 포함한다. 본 출원에 있어서, 디스패치라는 용어는 계산 유닛 세트 상의 작업 그룹 세트에 대해 커널의 실행의 시작을 개시하도록 컨텍스트 상태를 사용하는 DC에 의해 실행되는 커맨드를 지칭한다. DC(126)는 셰이더 코어(122)에서 작업그룹을 개시하는 로직을 포함한다. 일부 실시예에 있어서, DC(126)는 CP(124)의 일부분으로서 구현될 수 있다.

[0042] 또한 시스템(100)은 APD(104) 상에서의 실행을 위해 실행 리스트(150)로부터 프로세스를 선택하도록 하드웨어 스케줄러(HWS)(128)를 포함한다. HWS(128)는 라운드 로빈 방법론을 사용하여, 우선순위 레벨을 사용하여, 또는 다른 스케줄링 정책에 기반하여 실행 리스트(150)로부터 프로세스를 선택할 수 있다. 예를 들어, 우선순위 레벨을 동적으로 결정될 수 있다. HWS(128)는 또한, 예를 들어, 새로운 프로세스를 부가함으로써 그리고 실행-리스트(150)로부터 현존 프로세스를 삭제함으로써, 실행 리스트(150)를 관리하는 기능성을 포함할 수 있다. HWS(128)의 실행 리스트 관리 로직은 때로는 실행 리스트 컨트롤러(RLC)라고 지칭된다.

[0043] 본 발명의 다양한 실시예에 있어서, HWS(128)가 RLC(150)로부터 프로세스의 실행을 개시할 때, CP(124)는 대응하는 커맨드 버퍼(125)로부터 커맨드를 검색 및 실행하기를 시작한다. 일부 경우에 있어서, CP(124)는, CPU(102)로부터 수신된 커맨드와 대응하는, APD(104) 내 실행될 하나 이상의 커맨드를 발생시킬 수 있다. 일 실시예에 있어서, CP(124)는, 다른 컴포넌트와 함께, APD(104) 및/또는 시스템(100)의 자원의 이용을 개선 또는 최대화하는 방식으로 APD(104) 상의 커맨드의 우선순위결정 및 스케줄링을 구현한다.

[0044] APD(104)는 인터럽트 발생기(146)를 포함할 수 있거나 그로의 액세스를 가질 수 있다. 인터럽트 발생기(146)는 페이지 폴트와 같은 인터럽트 이벤트가 APD(104)에 의해 마주쳐질 때 OS(108)를 인터럽트하도록 APD(104)에 의해 구성될 수 있다. 예를 들어, APD(104)는 위에서 언급된 페이지 폴트 인터럽트를 생성하도록 IOMMU(116) 내 인터럽트 발생 로직에 의존할 수 있다.

[0045] APD(104)는 또한 셰이더 코어(122) 내 현재 실행 중인 프로세스를 선취하기 위한 선취 및 컨텍스트 스위치 로직(120)을 포함할 수 있다. 컨텍스트 스위치 로직(120)은, 예를 들어, 프로세스를 정지시키고 그 현재 상태(예를 들어, 셰이더 코어(122) 상태 및 CP(124) 상태)를 저장하는 기능성을 포함한다.

[0046] 여기서 지칭되는 바와 같이, 상태라는 용어는 초기 상태, 중간 상태, 및/또는 최종 상태를 포함할 수 있다. 초

기 상태는 머신이 출력 데이터 세트를 생성하기 위해 프로그램에 따라 입력 데이터 세트를 프로세싱하는 시작 포인트이다. 예를 들어, 프로세싱이 포워드 진행을 가능하게 하도록 수개 포인트에 저장될 필요가 있는 중간 상태가 있다. 이러한 중간 상태는 때로는 어떤 다른 프로세스에 의해 인터럽트될 때 나중 시간에 실행의 계속을 허용하도록 저장된다. 또한 출력 데이터 세트의 일부분으로서 기록될 수 있는 최종 상태가 있다.

[0047] 선취 및 컨텍스트 스위치 로직(120)은 또한 또 다른 프로세스를 APD(104) 내로 컨텍스트 스위칭하는 로직을 포함할 수 있다. 또 다른 프로세스를 APD(104) 상에서 실행 중으로 컨텍스트 스위칭하는 기능성은, 예를 들어, APD(104) 상에서 실행하도록 CP(124) 및 DC(126)를 통해 프로세스의 인스턴스를 생성하고, 그 프로세스에 대해 어느 이전에 저장된 상태를 복원하고, 그 실행을 시작하는 것을 포함할 수 있다.

[0048] 메모리(106)는 DRAM(도시하지 않음)과 같은 비-영속적 메모리를 포함할 수 있다. 메모리(106)는, 예를 들어, 애플리케이션 또는 다른 프로세싱 로직의 부분들의 실행 동안 프로세싱 로직 명령어, 상수 값, 변수 값을 저장할 수 있다. 예를 들어, 일 실시예에 있어서, CPU(102) 상에서 하나 이상의 동작을 수행하는 제어 로직의 부분은 CPU(102)에 의한 각자의 동작 부분의 실행 동안 메모리(106) 내 거주할 수 있다. 여기서 사용되는 바와 같이, 용어 "프로세싱 로직" 또는 "로직"은 제어 흐름 커맨드, 계산을 수행하기 위한 커맨드, 및 자원으로의 연관된 액세스를 위한 커맨드를 지칭한다.

[0049] 실행 동안, 각자의 애플리케이션, OS 함수, 프로세싱 로직 커맨드 및 시스템 소프트웨어는 메모리(106)에 거주할 수 있다. OS(108)에 핵심적인 제어 로직 커맨드는 일반적으로 실행 동안 메모리(106)에 거주할 수 있다. 예를 들어 커널 모드 드라이버(110) 및 소프트웨어 스케줄러(112)를 포함하는 다른 소프트웨어 커맨드는 또한 시스템(100)의 실행 동안 메모리(106)에 거주할 수 있다.

[0050] 이러한 예에 있어서, 메모리(106)는 APD(104)에 커맨드를 보내도록 CPU(102)에 의해 사용되는 커맨드 버퍼(125)를 포함한다. 또한 메모리(106)는 프로세스 리스트 및 프로세스 정보(예를 들어, 액티브 리스트(152) 및 프로세스 제어 블록(154))를 포함하고 있다. 이들 리스트와 더불어 그 정보는 APD(104) 및/또는 관련 스케줄링 하드웨어에 스케줄링 정보를 통신하도록 CPU(102) 상에서 실행하는 스케줄링 소프트웨어에 의해 사용된다. 메모리(106)로의 액세스는 메모리(106)에 연결되어 있는 메모리 컨트롤러(140)에 의해 관리될 수 있다. 예를 들어, 메모리(106)에 쓰기 위해 또는 그로부터 읽기 위해, CPU(102)로부터 또는 다른 디바이스로부터의 요청은 메모리 컨트롤러(140)에 의해 관리된다.

[0051] 시스템(100)의 다른 태양을 다시 참조하면, IOMMU(116)는 멀티-컨텍스트 메모리 관리 유닛이다.

[0052] 여기서 사용되는 바와 같이, 컨텍스트(때때로 프로세스라고 지칭)는 커널이 실행되는 환경 및 동기화 및 메모리 관리가 정의되는 도메인으로 생각될 수 있다. 컨텍스트는 디바이스 세트, 그들 디바이스에 액세스가능한 메모리, 대응하는 메모리 속성, 및 메모리 오브젝트 상에서 커널(들) 또는 동작의 실행을 스케줄링하도록 사용되는 하나 이상의 커맨드-큐를 포함한다. 다른 한편으로, 프로세스는 애플리케이션을 위한 프로그램의 실행이 컴퓨터 상에서 실행되는 프로세스를 생성할 것으로 생각될 수 있다. OS는 실행할 프로그램을 위해 가상 메모리 어드레스 공간 및 데이터 레코드를 생성할 수 있다. 프로그램의 실행의 현재 상태 및 메모리는 프로세스라 불릴 수 있다. OS는 초기 상태부터 최종 상태까지 메모리 상에서 동작할 프로세스를 위해 태스크를 스케줄링할 것이다.

[0053] 도 1a에 도시된 예를 다시 참조하면, IOMMU(116)는 APD(104)를 포함하는 디바이스에 대해 메모리 페이지 액세스를 위해 가상 대 물리적 어드레스 변환을 수행하는 로직을 포함한다. IOMMU(116)는 또한, 예를 들어, APD(104)와 같은 디바이스에 의한 페이지 액세스가 페이지 폴트의 결과를 초래할 때, 인터럽트를 발생시키는 로직을 포함할 수 있다. IOMMU(116)는 또한 변환 색인 버퍼(TLB)(118)를 포함하거나 그로의 액세스를 가질 수 있다. TLB(118)는, 예로서, 메모리(106) 내 데이터에 대하여 APD(104)에 의해 이뤄진 요청에 대해 물리적 메모리 어드레스로 논리적(즉, 가상) 메모리 어드레스의 변환을 가속화하도록 컨텐트 어드레싱가능한 메모리(CAM)에 구현될 수 있다.

[0054] 도시된 예에 있어서, 통신 기반구조(109)는 필요에 따라 시스템(100)의 컴포넌트를 상호접속한다. 통신 기반구조(109)는 주변장치 컴포넌트 상호접속(PCI) 버스, 확장된 PCI(PCI-E) 버스, 어드밴스트 마이크로컨트롤러 버스 아키텍처(AMBA) 버스, 가속화된 그래픽 포트(AGP), 또는 그러한 통신 기반구조 중 하나 이상(도시하지 않음)을 포함할 수 있다. 통신 기반구조(109)는 또한 이더넷, 또는 유사한 네트워크, 또는 애플리케이션의 데이터 전송 레이트 요건을 만족하는 어느 적합한 물리적 통신 기반구조를 포함할 수 있다. 통신 기반구조(109)는 컴퓨팅 시스템(100)의 컴포넌트를 포함하는 컴포넌트를 상호접속하는 기능성을 포함한다.

[0055] 이 예에 있어서, OS(108)는 시스템(100)의 하드웨어 컴포넌트를 관리하고 공통 서비스를 제공하는 기능성을 포함한다. 다양한 실시예에 있어서, OS(108)는 CPU(102) 상에서 실행되고 공통 서비스를 제공할 수 있다. 이들 공통 서비스는, 예를 들어, CPU(102) 내 실행을 위한 애플리케이션 스케줄링, 폴트 관리, 인터럽트 서비스와 더불어, 다른 애플리케이션의 입력 및 출력 프로세싱을 포함할 수 있다.

[0056] 일부 실시예에 있어서는, 인터럽트 컨트롤러(148)와 같은 인터럽트 컨트롤러에 의해 발생된 인터럽트에 기반하여, OS(108)가 적합한 인터럽트 취급 루틴을 인보크한다. 예를 들어, 페이지 폴트 인터럽트 검출시, OS(108)는 메모리(106) 내로 관련 페이지의 로딩을 개시하도록 그리고 대응하는 페이지 테이블을 업데이트하도록 인터럽트 핸들러를 인보크할 수 있다.

[0057] OS(108)는 또한 하드웨어 컴포넌트로의 액세스가 OS 관리된 커널 기능성을 통해 중재됨을 보장함으로써 시스템(100)을 보호하는 기능성을 포함할 수 있다. 실제로, OS(108)는 애플리케이션(111)과 같은 애플리케이션이 사용자 공간에서 CPU(102) 상에서 실행됨을 보장한다. OS(108)는 또한 애플리케이션(111)이 하드웨어 및/또는 입/출력 기능성에 액세스하도록 OS에 의해 제공된 커널 기능성을 인보크함을 보장한다.

[0058] 예로써, 애플리케이션(111)은 CPU(102) 상에 또한 실행되는 사용자 계산을 수행하도록 다양한 프로그램 또는 커맨드를 포함한다. 통합화 개념은 CPU(102)가 APD(104) 상의 프로세싱을 위해 선택된 커맨드를 무결절성으로 보낼 수 있게 한다. 이러한 통합된 APD/CPU 프레임워크 하에, 애플리케이션(111)으로부터의 입/출력 요청은 대응하는 OS 기능성을 통해 프로세싱될 것이다.

[0059] 하나의 예에 있어서, KMD(110)는 CPU(102) 또는 CPU(102) 상에서 실행되는 애플리케이션 또는 다른 로직이 통해 APD(104) 기능성을 인보크할 수 있는 API를 구현한다. 예를 들어, KMD(110)는 APD(104)가 후속하여 커맨드를 검색해낼 커맨드 버퍼(125)에 CPU(102)로부터의 커맨드를 인큐잉할 수 있다. 부가적으로, KMD(110)는, SWS(112)와 함께, APD(104) 상에서 실행될 프로세스의 스케줄링을 수행할 수 있다. SWS(112)는, 예를 들어, APD 상에서 실행될 프로세스의 우선순위결정된 리스트를 유지하는 로직을 포함할 수 있다.

[0060] 본 발명의 다른 실시예에 있어서, CPU(102) 상에서 실행되는 애플리케이션은 커맨드를 인큐잉할 때 KMD(110)를 전적으로 우회할 수 있다.

[0061] 일부 실시예에 있어서, SWS(112)는 APD(104) 상에서 실행될 프로세스의 메모리(106)에 액티브 리스트(152)를 유지한다. SWS(112)는 또한 하드웨어에서 HWS(128)에 의해 관리될 액티브 리스트(152) 내 프로세스의 서브세트를 선택한다. APD(104) 상에서 각각의 프로세스를 실행하는데 관련된 정보는 CPU(102)로부터 APD(104)로 프로세스 제어 블록(PCB)(154)을 통해 통신된다.

[0062] 애플리케이션용 프로세싱 로직, OS, 및 시스템 소프트웨어는, 궁극적으로는 마스크워크/포토마스크의 발생을 통해 제조 프로세스를 구성 가능하게 하여 여기서 설명되는 발명의 태양을 구체화하는 하드웨어 디바이스를 발생시키도록, C와 같은 프로그래밍 언어로 그리고/또는 베릴로그, RTL 또는 넷리스트와 같은 하드웨어 기술 언어로 특정된 커맨드를 포함할 수 있다.

[0063] 당업자는, 이 설명을 읽을 때, 컴퓨팅 시스템(100)이 도 1a에 도시된 것보다 더 많거나 더 적은 컴포넌트를 포함할 수 있음을 이해할 것이다. 예를 들어, 컴퓨팅 시스템(100)은 하나 이상의 입력 인터페이스, 비-휘발성 저장소, 하나 이상의 출력 인터페이스, 네트워크 인터페이스, 및 하나 이상의 디스플레이 또는 디스플레이 인터페이스를 포함할 수 있다.

[0064] 도 1b는 도 1a에 도시된 APD(104)의 더 상세한 예시를 나타내는 일 실시예이다. 도 1b에 있어서, CP(124)는 CP 파이프라인(124a, 124b, 124c)을 포함할 수 있다. CP(124)는, 도 1a에 도시된, 커맨드 버퍼(125)로부터 입력으로서 제공되는 커맨드 리스트를 프로세싱하도록 구성될 수 있다. 도 1b의 전형적 동작에 있어서, CP 입력 0(124a)은 그래픽 파이프라인(162)으로의 커맨드를 구동하는데 책임이 있다. CP 입력 1 및 2(124b, 124c)는 계산 파이프라인(160)에 커맨드를 포워딩한다. 또한 HWS(128)의 동작을 제어하기 위한 컨트롤러 메커니즘(166)이 제공된다.

[0065] 도 1b에 있어서, 그래픽 파이프라인(162)은 여기서 순차 파이프라인(164)이라고 지칭되는 블록 세트를 포함할 수 있다. 예로서, 순차 파이프라인(164)은 정점 그룹 변환기(VGT)(164a), 원시 어셈블러(PA)(164b), 스캔 컨버터(SC)(164c), 및 셰이더-엑스포트, 렌더-백 유닛(SX/RB)(176)을 포함한다. 순차 파이프라인(164) 내 각각의 블록은 그래픽 파이프라인(162) 내 그래픽 프로세싱의 다른 스테이지를 표현할 수 있다. 순차 파이프라인(164)은 고정 함수 하드웨어 파이프라인일 수 있다. 그렇지만, 본 발명의 취지 및 범위 내에 또한 있을 다른 구현이 사

용될 수 있다.

[0066] 소량의 데이터만이 그래픽 파이프라인(162)에 입력으로서 제공될 수 있지만, 이러한 데이터는 그것이 그래픽 파이프라인(162)으로부터 출력으로서 제공될 때까지 증폭될 것이다. 그래픽 파이프라인(162)은 또한 CP 파이프라인(124a)으로부터 수신된 작업-항목 그룹 내 레인지에 걸쳐 카운트하기 위한 DC(166)를 포함한다. DC(166)를 통해 의뢰된 계산 작업은 그래픽 파이프라인(162)과 반-동기식이다.

[0067] 계산 파이프라인(160)은 셰이더 DC(168, 170)를 포함한다. DC의 각각은 CP 파이프라인(124b, 124c)으로부터 수신된 작업 그룹 내 계산 레인지에 걸쳐 카운트하도록 구성된다.

[0068] 도 1b에 예시된 DC(166, 168, 170)는 입력 레인지를 수신하고, 레인지를 작업그룹으로 세분하고, 그 후 작업그룹을 셰이더 코어(122)에 포워딩한다.

[0069] 그래픽 파이프라인(162)은 일반적으로 고정 함수 파이프라인이므로, 그 상태를 저장 및 복원하는 것이 어렵고, 결과로서, 그래픽 파이프라인(162)은 컨텍스트 스위칭하기가 어렵다. 그래서, 대부분 경우에 있어서, 여기서 논의되는 바와 같은 컨텍스트 스위칭은 그래픽 프로세스 간 컨텍스트 스위칭과 관련 있지 않다. 컨텍스트 스위칭될 수 있는 예외는 셰이더 코어(122)에서의 그래픽 작업에 대해서이다.

[0070] 셰이더 코어(122)는 그래픽 파이프라인(162) 및 계산 파이프라인(160)에 의해 공유될 수 있다. 셰이더 코어(122)는 웨이브프론트를 실행하도록 구성된 일반 프로세서일 수 있다.

[0071] 하나의 예에 있어서, 계산 파이프라인(160) 내 모든 작업은 셰이더 코어(122) 내에서 프로세싱된다. 셰이더 코어(122)는 프로그램가능한 소프트웨어 코드를 실행하고 상태 데이터와 같이 다양한 형태의 데이터를 포함한다. 그렇지만, 계산 파이프라인(160)은 프로세싱을 위해 그래픽 파이프라인(162)에 작업을 보내지 않는다. 그래픽 파이프라인(162) 내 작업의 프로세싱이 완료된 후에, 완료된 작업은 렌더 백 유닛(176)을 통해 프로세싱되는데, 깊이 및 컬러 계산을 행하고 그 후 그 최종 결과를 그래픽 메모리(130)에 쓴다.

[0072] 도 2는 도 1b에 도시된 전형적 APD(104)의 더 상세한 예시적 블록 선도이다. 도 1b에 도시된 바와 같이, APD(104)는 셰이더 코어(122)에 입력 1 및 입력 2를 제공하는 계산 파이프라인(160)을 포함한다. 도 2에 예시된 전형적 APD는 8개의 계산 파이프라인 CS 파이프 0 - CS 파이프 7(CS P0 - CS P7)을 포함한다. 이러한 구성은 다중 계산 파이프라인을 통해 다중 계산 태스크를 프로세싱하도록 구성된다. APD(200) 내 다중 계산 파이프라인은 계산 작업부하 간 유연한 자원 할당을 용이하게 한다. 전형적 APD(200)가 8개의 계산 파이프라인을 예시하고 있지만, 당업자는 다른 수의 계산 및 그래픽 입력이 사용될 수 있음을 인식할 것이다.

[0073] 다중 계산 입력으로부터의 데이터를 효율적으로 프로세싱하기 위해, 도 3에 더 상세히 예시된 바와 같이, 계산 파이프라인 CS P0 - CS P7 내 파이프라인 큐 간 중재가 일어난다. 더 구체적으로, 본 발명의 실시예에 따른 중재 정책은 다중 파이프라인 입력 간 APD 자원을 할당한다. 셰이더 입력 블록(SPI)(202)은 계산 파이프라인 CS P0 - CS P7과 그래픽 파이프라인(204) 간 웨이브프론트를 의뢰하기 위한 중재 기법을 제공한다. 웨이브 디스패처(206)는 웨이브프론트를 셰이더 코어(208)에 포워딩하기 위해 교번 2개의 계산 파이프라인으로부터 접속된다. 셰이더 코어(208)는 웨이브프론트를 실행한다.

[0074] 도 3은 도 2에 도시된 계산 파이프라인 CS P0 - CS P7의 더 상세한 예시적 블록 선도이다. 이들 8개의 계산 파이프라인은 공유된 셰이더 코어(208)로의 액세스를 위해 중재에 참가한다. 각각의 계산 파이프라인 CS P0 - CS P7은, 예를 들어, 하드웨어 큐 기술자(HQD)를 포함한다. 계산 파이프라인 CS P0은 HQD0과 연관되고, 계산 파이프라인 CS P1은 HQD1과 연관되고 그렇게 계속 CS P7과 HQD7까지이다. 각각의 하드웨어 큐 기술자(HQD)는 8개의 메모리 큐의 연관된 큐를 포함한다. 도 4에 도시된 바와 같이, 예를 들어, CS P0은 큐 Q0 - Q7과 연관된다. 유사하게, CS P1은 큐 Q8 - Q15과 연관되고, 그렇게 계속 CS 파이프 7 큐 Q56 - Q63이다. CP 멀티스레딩된 마이크로프로세서 엔진 ME(301) 및 그리드 DC Cntr 0 - Cntr 3이 스레드 그룹을 프로세싱하도록 제공된다.

[0075] 위에서 논의된 바와 같이, 하드웨어 스케줄러(HWS)(128)는 APD 상에서의 실행을 위해 스케줄링된 프로세스를 RLC(150)로부터 선택하도록 구성된다. 예를 들어, HWS(128)는, 우선순위 레벨에 기반하여 또는 다른 중재 스케줄링 기준에 기반하여, RLC(150)에 적용된 스케줄링 기술을 지원한다. 부가적으로, KMD(110)는, SWS(112)와 함께, APD 상에서 실행될 프로세스의 스케줄링을 수행할 수 있다. OS SWS(112)는, 예를 들어, 중재의 결과로서 APD(200) 상에서 실행될 프로세스의 우선순위결정된 리스트를 유지하도록 로직을 포함할 수 있다.

[0076] 또 다른 예시적 실시예에 있어서, 각각의 파이프라인의 계산 파이프라인 CS P0 - CS P7 하드웨어 큐 기술자 - HQD0 - HQD7 간 중재는 멀티레벨 스케줄링 프로세스를 사용하여 해결된다. 다중 계산 입력으로의 구현에

있어서, 멀티레벨 스케줄링은 각각의 계산 입력이 유사한 우선순위의 작업과 연관되는 멀티-레벨 우선순위 큐 간 자원 할당을 제어하도록 사용될 수 있다.

[0077] OS는 하드웨어 큐 기술자를 프로그래밍함으로써 계산 파이프라인 하드웨어 큐 기술자 HQD0 - HQD7에 의한 프로세싱을 위해 큐 Q0 - Q7를 스케줄링할 수 있다. 8개의 하드웨어 큐 기술자(HQD) 중 어느 것이라도 액티브 큐를 포함하고 있을 수 있다. 하나의 계산 파이프라인과 연관된 큐는 독립적 프로세스일 수 있거나 프로세스의 서브세트의 구현을 표현할 수 있다. 예를 들어, 하나의 파이프라인에서 확립된 어느 시스템이라도, 도 1a 및 도 1b에 예시된 L2 R/W 캐시(174), 그래픽 또는 메모리(130)와 같이, 공유된 메모리 중 하나 이상에서 확립된 동기화를 통해 다른 계산 파이프라인으로부터의 큐 또는 큐 세트와 상호작용할 수 있다.

[0078] 각각의 큐와 연관된 하드웨어 큐 기술자(HQD)는 OS가 어느 셰이더 자원도 아직 할당하지 않은 어느 더 많은 작업 그룹을 디스패치하는 것으로부터 액티브 프로세스를 선취할 수 있는 능력을 제공할 수 있다. 하드웨어로부터 제거되는 어느 큐라도 OS에 의해 소망되면 종료되거나 추후 시간에서의 계속을 위해 다시 스케줄링될 수 있다.

[0079] 하드웨어 큐 기술자 HQD0 - HQD7의 각각은 OS 할당된 메모리 큐 기술자(MQD)의 메모리 큐 기술자 어드레스(MQDA)를 포함할 수 있다. OS는 MQD를 사용하여 큐의 영구 상태를 저장하고 MQDA 어드레스를 HQD에 제공할 수 있고 그래서 하드웨어는 메모리 큐 기술자의 필드를 선택하도록 업데이트할 수 있다. 메모리 큐 기술자가 HQD로부터 접속해제될 때, 하드웨어는 필요한 영속 데이터를 어느 선취 동안 임시로 저장하기 위해 MQD의 일부를 사용할 것이다. 또한 공간의 서브세트는 OS와 HQD 간 동기화 조정에 사용될 수 있다.

[0080] 큐 중재

[0081] 도 5는 본 발명의 실시예를 실시하는 예시적 방법의 흐름도이다. 도 5의 단계(502)에서, 각각의 계산 파이프라인 CS P0 - CS P7에 대해 8개의 하드웨어 큐 기술자 HQD0 - HQD7 큐 중으로부터 레디 큐 및 액티브 큐가 선택된다. 예로써, 선택은 각각의 계산 파이프라인에 의해 독립적으로 그리고 병렬로 수행될 수 있다.

[0082] 본 발명의 일 실시예에 있어서, 웨이브프론트 큐 패킷당 다음의 레지스터 제어가 제공된다:

[0083] 1). 큐 액티브(1 비트),

[0084] 2). 큐 우선순위(4 비트 0-15 → L-H), 3),

[0085] 3). 퀀텀 듀레이션(5000 clks 유닛으로 5 비트),

[0086] 4). 퀀텀 이네이블(1 비트),

[0087] 5). 파이프 우선순위(2 비트), 및

[0088] 6). 레디("레디"는 큐가 액티브 AND임을 나타낸다(공백 아님, OR 디스패치 파이프 공백 아님) AND 큐 기능 정지 아님.

[0089] 단계(504)에서, 프로세싱을 위해 레디인 것으로 결정되는 가장 높은 큐 우선순위를 갖는 큐가 선택된다. 선택되고 나면, 예를 들어, 큐는 다음 조건 중 하나가 일어날 때까지 선택된 채로 있다:

[0090] 1). 더 높은 우선순위 큐가 레디로 된다,

[0091] 2). 프로세싱 듀레이션이 초과되고 동일 우선순위의 또 다른 큐가 프로세싱을 위해 레디인 것처럼, 퀀텀이 이네이블로 된다,

[0092] 3). 퀀텀이 디스에이블로 되고 현재 큐 내 웨이브프론트 패킷이 어느 다른 큐 우선순위 레지스터를 쓰고 동일 우선순위의 또 다른 큐가 레디이다,

[0093] 4). 현재 큐 웨이브프론트 패킷이 계산 파이프라인으로부터 큐를 선취하고 예를 들어 타이머 만료와 같은 특정 조건으로 스마트 대기를 스케줄링한다,

[0094] 5). 현재 큐 및 계산 파이프라인 DC(206)가 공백으로 되고 동일 계산 파이프라인에서의 어느 다른 큐가 레디이다, 그리고

[0095] 6). OS는 현재 큐가 선취하도록 요청한다.

[0096] 단계(506)에서, 계산 파이프라인의 상부에서의 큐 아비터는 더 나은 큐가 프로세싱을 위해 레디임을 아비터가 결정할 때 다음 패킷 경계 상에서 정지하도록 각자의 CP ME(301) 스레드를 시그널링한다. 더 나은 큐가 이용가

능하지 않음이 결정되면, 프로세스는 단계(508)에서 계속된다.

[0097] 단계(510)에서, CP ME(301)는 컨텍스트 스위칭 루틴을 수행하고 인출기에 큐 데이터 인출을 정지하라고 그리고 DC에 현재 큐에 대해 웨이브프론트 디스패치를 정지하라고 시그널링한다. CP ME(301)는 각자의 그리드 DC Contr 0 - Contr 3의 재시작 스레드 그룹 아이디를 저장할 수 있다.

[0098] 단계(512)에서, 스위칭 계산 파이프라인의 현재 영속 상태는 큐를 재시작하기 이전에 상태 사전-로딩에 사용될 미리-정의된 오프셋에서 각자의 메모리 큐 기술자(MQD)에 저장된다. 파이프 종료 저장 디워드 현재 최종 읽기 어드레스를 갖는 MQD를 겨냥하는 프로세스 종료(EOP) 펜스 이벤트가 삽입될 수 있다. 큐가 하드웨어로부터 제거되는 한편 작업이 셰이더 콤플렉스에서 미결인 이벤트에서, HQD 최종 읽기 어드레스는 MQD에 저장된다. 그 후 낮은 레벨 드라이버는, 파이프 상부 최종 읽기 어드레스와 파이프 종료 최종 읽기 어드레스를, 그것들이 모든 작업이 마쳐졌음을 매칭할 때, 비교함으로써, 큐에 대한 모든 미결 작업이 완료된 때를 결정할 수 있다.

[0099] 단계(514)에서, 이전 큐의 상태가 저장되도록 스케줄링되고 사전-인출된 데이터는 폐기되도록 스케줄링된다. CP ME는 프로세싱을 위한 다음 큐를 선택하도록 인출기를 릴리스할 수 있다. 다음 큐가 퍼스트 타임 상태 비트 세트를 갖고 있으면, 인출기는 MQD로부터 저장된 영속 상태의 로드를 삽입하고 큐 인출을 위해 큐 읽기/쓰기 포인터 셋업이 뒤따를 것이다. 일 실시예에 있어서, 스위치의 예상 시간은 CP ME가 다음 큐 프로세싱을 시작할 때까지 대략 500 클록일 수 있다.

[0100] 도 6에 도시된 다음의 예를 생각해본다, 여기서, T<n>=시간, 및 n=clk 수이다,

[0101] QA = 큐 액티브,

[0102] QP = 큐 우선순위,

[0103] QE = 퀀텀 이네이블, 및

[0104] QD = 5000 clks의 퀀텀 듀레이션 유닛. 시간은 단일 계산 파이프라인의 8개의 큐 Q0 - Q7에 대해 수직으로 증가한다.

[0105] 도 6에 도시된 바와 같이, 각각의 우선순위 레벨에 대해, 계산 파이프라인은 마지막 큐 실행된 스코어보드를 유지한다. 그 우선순위 레벨로의 복귀는 다음 레디 큐를 프로세싱할 것이다. 하나의 큐만이 우선순위 레벨에서 레디이면, 그것은 재개할 것이다.

[0106] 파이프는 0부터 7까지 큐를 오더링할 수 있고, 리셋시 이전 큐는 7로 설정되어, 네이티브 오더링으로서 Q0 → Q7의 결과를 초래할 것이다. Q0, Q3, Q7이 리셋 직후 이네이블로 된 퀀텀에서 큐 우선순위 7로 레디로 되면, 큐는 다음 순서 Q0, Q3, Q7, Q0 등으로 프로세싱할 것이다. Q5가 동일 큐 우선순위 레벨(7)로 나타났으면, 그것은 다음 사이클 동안 Q3 후 및 Q7 전에 실행되게 될 것이다.

[0107] 그 후 Q1 및 Q4가 Q5 퀀텀 동안 우선순위 10으로 도착하면, Q1은 Q5에 선취하고, 시스템은 큐가 공백으로 될 때까지 또는 또 다른 큐가 프로세싱을 위해 스케줄링될 때까지 Q4와 Q1 간 반복적으로 퀀텀을 스위치 온 한다. Q1 및 Q4가 공백으로 되면, 프로세스는 Q5가 우선순위 7 레벨에서 이전에 프로세싱되었으므로 Q7으로 되돌아간다.

[0108] 도 6은 본 발명의 실시예에 따른 전형적 방법의 예시이다. 도 6에 도시된 예에 있어서, 계산 파이프라인은 파이프를 다른 큐에 내어놓도록 2개의 주요 방법 중 하나를 사용한다. 제1 방법은 시간 퀀텀 만료에 응답하여서이고 제2 방법은 큐 우선순위 레지스터에 쓰는 것이다.

[0109] 위에서 논의된 바와 같이, 초과되고 이네이블로 된 시간 퀀텀을 갖는 큐는 동일 우선순위의 기존 큐 또는 동일 또는 더 높은 우선순위의 도착 큐에 기인하여 선취를 이네이블 할 것이다. 큐가 유일한 가장 높은 우선순위 큐이면, 큐는 동일 또는 더 높은 우선순위의 큐가 레디로 될 때까지 계산 파이프라인의 소유권을 보유할 것이다.

[0110] 대안의 실시예에 있어서, 중재 이벤트는 계산 파이프라인의 큐 우선순위 레지스터에 어느 쓰기에 대해 생성될 수 있다. 이러한 방법은 파이프의 다른 큐가 진행하도록 이네이블 하기 이전에 발행된 작업량을 사용자가 제어 가능하게 할 수 있다. 부가적으로 이러한 대안 실시예는 CP ME당 특권 큐를 이네이블 할 수 있다.

[0111] 계산 파이프라인을 가로지르는 중재

[0112] 가장 높은 우선순위 큐가 각각의 계산 파이프라인 하드웨어 기술자 큐 내에서 해결되고 나면, 중재의 다음 포인트는 가장 높은 파이프 우선순위를 갖는 계산 파이프라인으로부터의 어느 웨이브프론트가 프로세싱을 위해 셰이더 코어에 의뢰될지 해결해야 한다. 2개의 계산 파이프라인이 교번 방식으로 공통 DC를 공유하기 때문에, 우선

순위가 결정된 후, 공유된 회로는 어느 계산 파이프라인이 셰이더 코어에 의뢰되는지 할당한다.

[0113] 예를 들어, 참가 파이프라인은 그래픽 파이프라인 중 어느 하나, HP3D 태스크(LS, HS, ES, GS, VS, PS) 및 GFX 태스크(LS, HS, ES, GS, VS, PS, CS), 및 8개의 계산 파이프라인 중 4개일 수 있다. 계산 파이프라인은 다음 파이프 우선순위 중 하나를 가질 수 있다: CS_HIGH - HP3D 위, CS_MEDIUM - HP3D와 GFX 사이, CS_LOW - GFX 아래.

[0114] 동일 파이프 우선순위 레벨의 다중 계산 파이프라인 간 동점을 해결하기 위해, 예를 들어, 토템 폴 회로와 같이 가장 덜 최근 발행되거나 가장 덜 최근 사용된 회로가 채용될 수 있다. 파이프라인이 어느 작업을 셰이더 코어에 발행하도록 선택될 때마다, 파이프라인은 가장 덜 최근 발행된 회로의 바닥으로 이동하여 동일 우선순위의 또 다른 파이프라인이 웨이브프론트를 발행할 때까지 파이프라인 우선순위의 가장 낮은 우선순위를 그 파이프라인에 할당할 것이다. 이러한 특수 회로는 동일 우선순위의 작업 그룹을 발행함에 있어서 공정성을 조성하는 것을 도와주도록 사용될 것이다.

[0115] 리셋에서 나와, 가장 덜 최근 발행된 리스트는 P0 → P7로 되어 파이프 0이 그 주어진 파이프 우선순위에 대해 처음에 가장 혜택받게 될 것이다.

[0116] 도 7은 전형적 파이프라인 중재 정책의 예시이다. CS HIGH, HP3D, CS MEDIUM, GFX, CS_LOW의 5개의 우선순위 레벨 중에서, 가장 높은 것으로부터 가장 낮은 우선순위 레벨로 최상의 승자가 선택될 것이다.

[0117] 그 예에 있어서, "굵은" 파이프라인은 웨이브프론트 론치에 대해 생각되고 "굵고 밑줄친" 파이프라인은 웨이브프론트 론치를 위해 선택된다.

[0118] 도 7에 예시된 테이블은 좌로부터 우로 전형적 토템 폴 배열을 도시하고 있다. 그 예에 있어서, 계산 파이프라인 CS P0 - CS P7은 Pn으로 표현되고, 여기서 n은 웨이브프론트를 제공하는 계산 파이프라인이고 (-)는 없음 = 작업 없음을 의미한다. H는 - 파이프 우선순위 높음을 표현하고, M은 - 파이프 우선순위 중간을 표현하고, L은 - 파이프 우선순위 낮음을 표현한다. 각각의 시간 기간 동안, DC에 의한 계산 파이프라인 쌍 중재에서 살아남은 8개의 계산 파이프라인 중 4개는 "굵게" 도시되고 "굵은 밑줄친" 계산 파이프라인은 겨뤄지는 6개의 파이프라인 중 파이프 중재가 선택할 파이프이다.

[0119] 결어

[0120] 개요 및 요약 절은 발명자(들)에 의해 고려되는 바와 같이 본 발명의 하나 이상의 그러나 모두는 아닌 전형적 실시예를 제시하며, 그리하여, 본 발명 및 첨부 청구범위를 어떠한 식으로도 한정하려는 의도는 아니다.

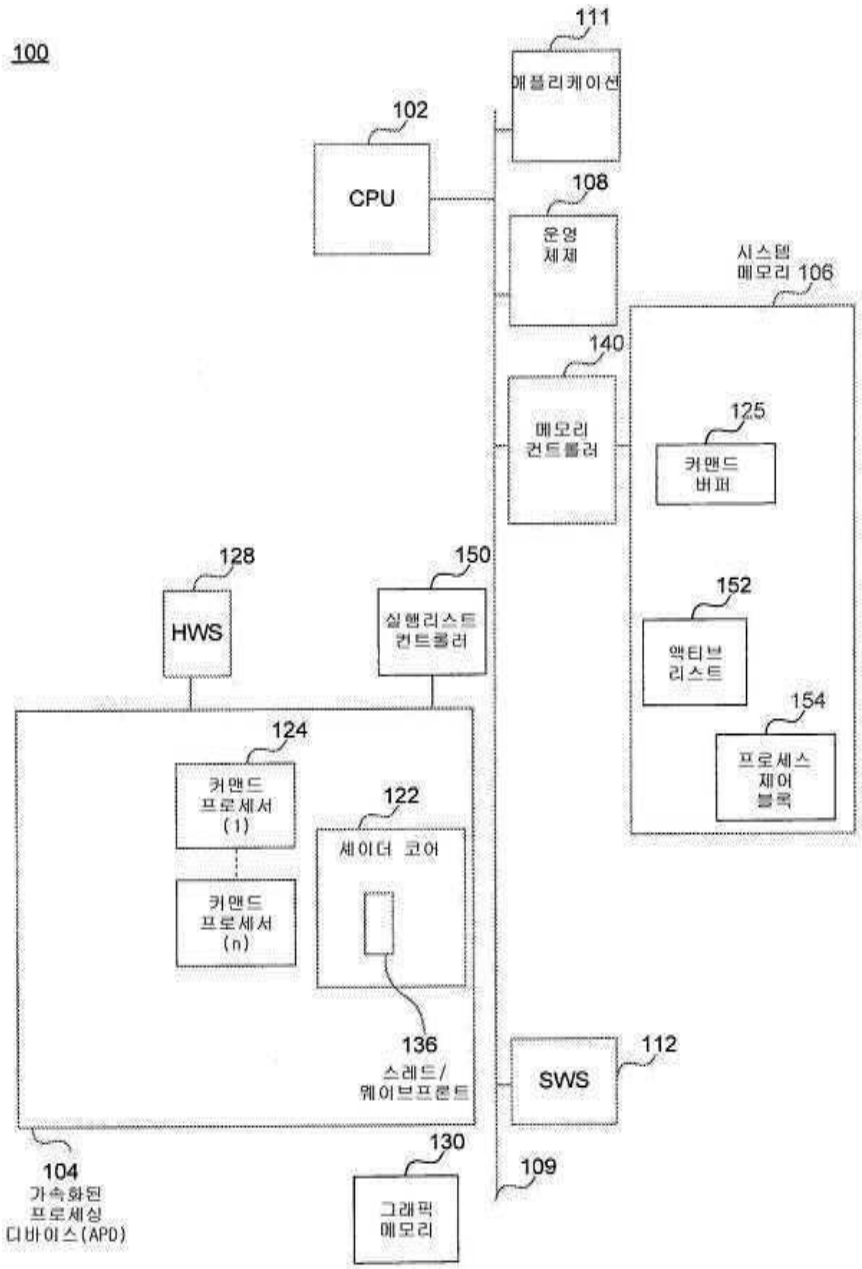
[0121] 본 발명은 특정 기능 및 그 관계의 구현을 예시하는 기능적 구조 블록의 도움으로 위에서 설명되었다. 이들 기능적 구조 블록의 경계는 설명의 편의를 위해 여기서 임의로 정의되었다. 특정 기능 및 그 관계가 적절히 수행되는 한 대체 경계가 정의될 수 있다.

[0122] 특정 실시예의 상기 설명은 타인이 본 발명의 일반적 개념으로부터 벗어남이 없이 과도한 실험 없이 당업자의 지식을 적용함으로써 그러한 특정 실시예의 다양한 응용을 위해 쉽게 수정 및/또는 적응할 수 있도록 그렇게 본 발명의 일반적 본성을 충분히 드러낼 것이다. 그래서, 그러한 적응 및 수정은, 여기서 제시된 가르침 및 가이드에 기반하여, 개시된 실시예의 균등물의 의미 및 범위 내에 있는 것으로 의도된다. 본 명세서의 어법 또는 용어는 그 가르침 및 가이드에 비추어 당업자에 의해 해석되도록 여기에서의 어법 또는 용어는 제한이 아닌 설명의 목적을 위한 것임을 이해해야 한다.

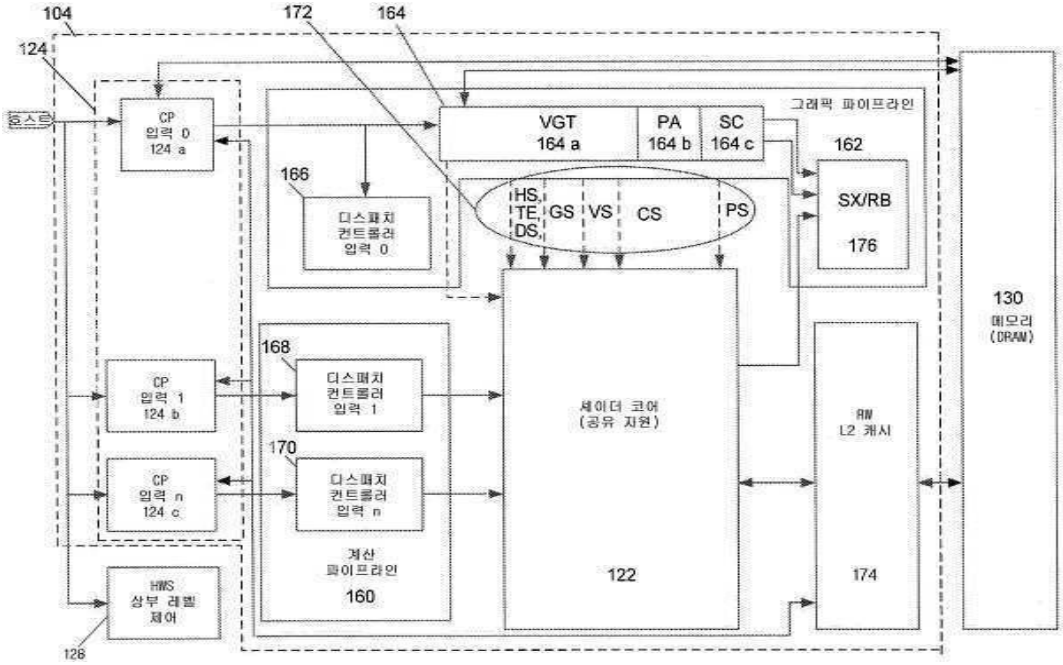
[0123] 본 발명의 폭 및 범위는 위에서 설명된 전형적 실시예 중 어느 것에 의해서도 한정되어서는 안 되며, 이하의 청구범위 및 그들 균등물에 의해서만 정의되어야 한다.

도면

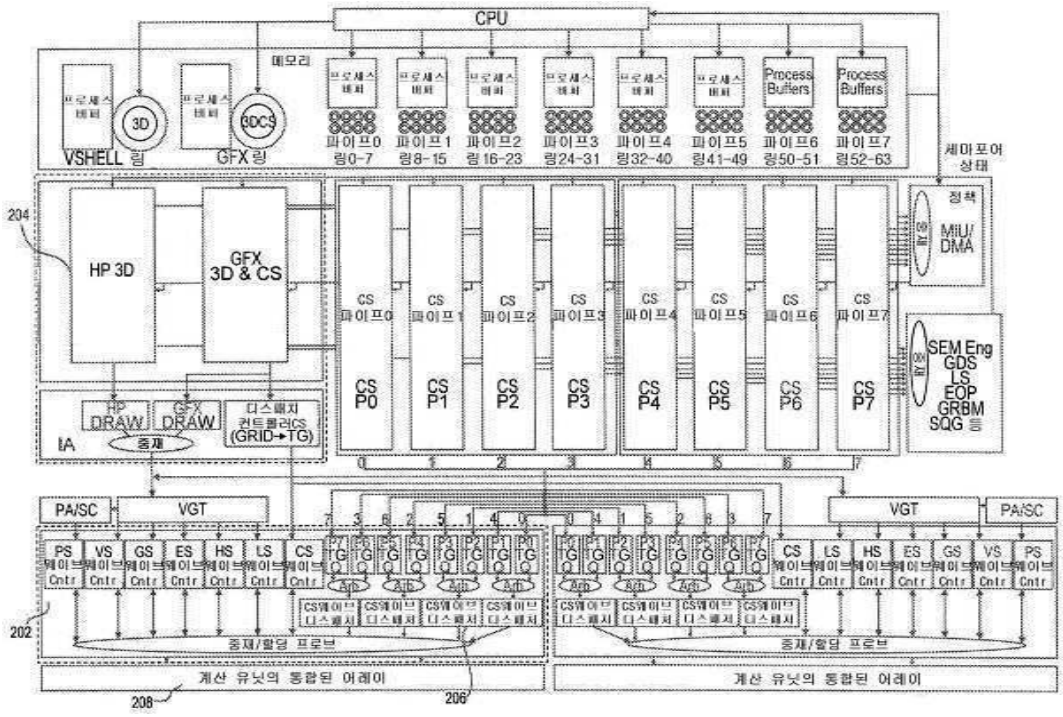
도면1a


100
111
애플리케이션
102
CPU
108
운영
체제
시스템
메모리 106
140
메모리
컨트롤러
125
커맨드
버퍼
128
HWS
150
실행리스트
컨트롤러
152
액티브
리스트
154
프로세스
제어
블록
124
커맨드
프로세서
(1)
커맨드
프로세서
(n)
122
쉐이더 코어
136
스레드/
웨이브프론트
SWS
112
104
가속화된
프로세싱
디바이스(APD)
130
그래픽
메모리
109

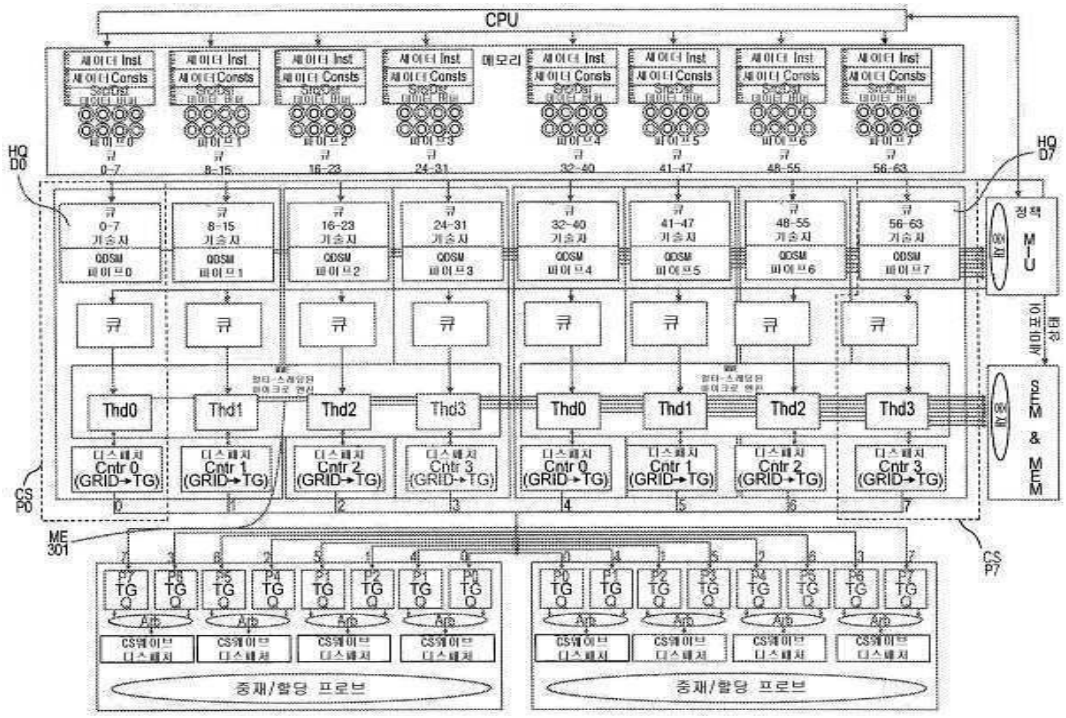
도면1b


104
172
164
124
호스트
CP
입력 0
124 a
그래픽 파이프라인
VGT
164 a
PA
164 b
SC
164 c
162
166
디스패치
컨트롤러
입력 0
HS,
TE,
DS,
GS
VS
CS
PS
SX/RB
176
130
메모리
(DRAM)
168
CP
입력 1
124 b
디스패치
컨트롤러
입력 1
쉐이더 코어
(공유 자원)
RW
L2 캐시
170
CP
입력 n
124 c
디스패치
컨트롤러
입력 n
계산
파이프라인
160
122
174
HWS
상부 레벨
제어
126

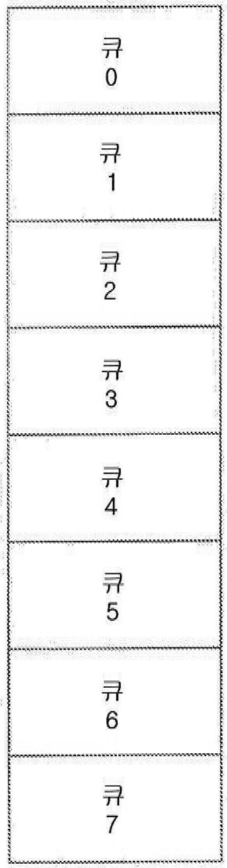
도면2


CPU
메모리
VSHELL 링
GFX 링
파이프0
링0-7
파이프1
링8-15
파이프2
링16-23
파이프3
링24-31
파이프4
링32-40
파이프5
링41-49
파이프6
링50-51
파이프7
링52-63
Process Buffers
Process Buffers
세마포어
상태
204
HP 3D
GFX
3D & CS
정책
MIU/
DMA
CS
파이프0
CS
파이프1
CS
파이프2
CS
파이프3
CS
파이프4
CS
파이프5
CS
파이프6
CS
파이프7
CS
P0
CS
P1
CS
P2
CS
P3
CS
P4
CS
P5
CS
P6
CS
P7
SEM Eng
GDS
LS
EOP
GRBM
SQG 등
HP
DRAW
GFX
DRAW
디스패치
컨트롤러CS
(GRID→TG)
IA
중재
PA/SC
VGT
VGT
PA/SC
PS
VS
GS
ES
HS
LS
CS
Cntr
Arb
중재/할당 프로브
계산 유닛의 통합된 어레이
202
208
206

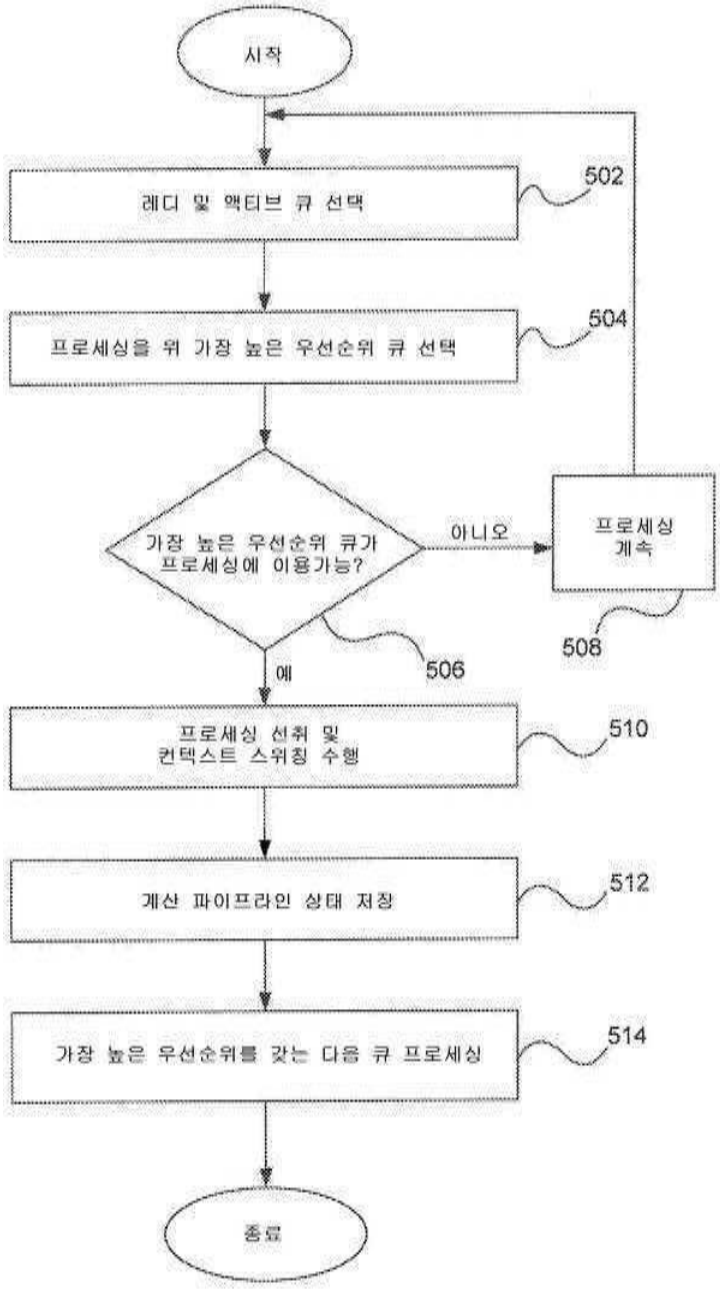
도면3


CPU
메모리
HQ D0
HQ D7
정책 MIU
SPM & MEM
Thd0
Thd1
Thd2
Thd3
중재/할당 프로브
중재/할당 프로브
CS P0
CS P7
ME 301

도면4


큐 0
큐 1
큐 2
큐 3
큐 4
큐 5
큐 6
큐 7

도면5


시작
레디 및 액티브 큐 선택
502
프로세싱을 위 가장 높은 우선순위 큐 선택
504
가장 높은 우선순위 큐가 프로세싱에 이용가능?
아니오
프로세싱 계속
508
506
예
프로세싱 선취 및 컨텍스트 스위칭 수행
510
계산 파이프라인 상태 저장
512
가장 높은 우선순위를 갖는 다음 큐 프로세싱
514
종료

도면6

시간	큐 변화/액션	큐 선택
T0	=> Q0 QA=1, QP=5, QE=1, QD=2	Q0
T 1,000	=> Q7 QA=1, QP=0, QE=1, QD=2	Q0
T10,000	=> Q0 퀀텀 만료. 동일 또는 더 높은 우선순위 큐 없음. 계속	Q0
T11,000	=> Q1 QA=1, QP=5, QE=1, QD=2 (Q0 퀀텀 만료. 스위치)	Q1
T21,000	=> Q1 퀀텀 만료. 동일 우선순위의 다음 큐로 스와핑	Q0
T24,000	=> Q2 QA=1, QP=8, QE=1, QD=2 (가장 높은 우선순위)	Q2
T28,000	=> Q2 세마포어 대기하도록 패킷을 프로세싱. 다음 이용가능한 것으로 스위칭	Q1
T38,000	=> Q1 퀀텀 만료. 우선순위 레벨의 다음 큐로 스와핑	Q0
T40,000	=> Q2 세마포어 만족 및 Q2 레디(가장 높은 우선순위, 선취)	Q2
T44,000	=> Q2 큐&파이프 공백. 우선순위 레벨5의 다음 큐	Q1
T46,000	=> Q1 큐&파이프 공백. 우선순위 레벨5의 다음 큐	Q0
T49,000	=> Q0 큐&파이프 공백. 우선순위 레벨0의 다음 큐	Q7
T51,000	=> Q2 레디로 됨(가장 높은 우선순위, 선취)	Q2
T55,000	=> Q3 QA=1, QP=10, QE=0, QD=0 (가장 높은 선취)	Q3
T56,000	=> Q4 QA=1, QP=10, QE=0, QD=0	Q3
T59,000	=> Q3 그 QP=10로 리셋하는 패킷 발행. 동일 우선순위 스위칭 이네이블	Q4
T66,000	=> Q4 Q3 QP=11로 리셋하는 패킷 발행. 더 높은 우선순위 큐 생성	Q3
T67,000	=> Q2 레디로 됨 ... 높은 우선순위 아니기 때문에 액션 없음	Q3
T70,000	=> Q3 Q3&Q4를 QP4로 리셋하는 패킷 발행하여 시스템의 우선순위 감소	Q2

도면7

P0-P7에 대해 가장덜최근 발행된 H->L 우선순위리스트

시간										
T0	P0(-)	P1(H)	P2(-)	P3(-)	P4(-)	P5(-)	P6(-)	P7(-)	HP3D(-)	GFX(-)
T1	P0(-)	P2(-)	P3(-)	P4(M)	P5(H)	P6 (-)	P7(-)	P1(H)	HP3D(-)	GFX(-)
T2	P0(M)	P2(L)	P3(L)	P4(M)	P5(H)	P6 (M)	P7(M)	P1(H)	HP3D(-)	GFX(-)
T3	P0(M)	P2(L)	P3(L)	P4(M)	P6 (M)	P7(M)	P1(H)	P5(-)	HP3D(-)	GFX(-)
T4	P0(M)	P2(L)	P3(L)	P4(M)	P6 (M)	P7(M)	P5(-)	P1(H)	HP3D(-)	GFX(-)
T5	P2(L)	P3(L)	P4(M)	P6 (M)	P7(M)	P5(-)	P1(H)	P0(M)	HP3D(-)	GFX(-)
T6	P2(L)	P3(L)	P4(M)	P6 (M)	P7(M)	P5(-)	P0(M)	P1(H)	HP3D(X)	GFX(-)
T7	P2(L)	P3(L)	P4(M)	P6 (M)	P7(M)	P5(L)	P0(M)	P1(-)	HP3D(-)	GFX(X)
T8	P2(L)	P3(L)	P4(M)	P6 (M)	P5(L)	P0(M)	P1(-)	P7(M)	HP3D(-)	GFX(X)
T9	P2(L)	P3(L)	P4(M)	P5(L)	P0(M)	P1(-)	P7(-)	P6 (M)	HP3D(-)	GFX(X)
T10	P2(L)	P3(L)	P5(L)	P0(M)	P1(L)	P7(-)	P6(-)	P4(M)	HP3D(-)	GFX(X)