멀티미디어 퓨전(CPU-GPU) 프로세서

5대 분야 Interactive VR/AR • **Function** Processing • **기술분야명** 프로세서

Processing

담당 센터 SoC플랫폼 • **연구자** 김동순

개 념

고화질 대용량의 실감형 2D/3D 멀티미디어 데이터를 처리하기 위한 고성능 시스템 반도체 기술

개발 내용

기술내용

고성능 멀티코어 CPU와 멀티 Shader* 기반 GPU를 통합한 퓨전 프로세서 핵심 원천기술

* 그래픽 처리 장치(GPU)의 렌더링 파이프라인을 프로그래밍하는 데 쓰이는 명령어 집합

이기종 프로세서(CPU-GPU) 간 효율적인 협업을 위한 독자적인 구조의 퓨전 아키텍쳐 개발

저전력 멀티 Shader Core IP, CPU-GPU 퓨전 아키텍처 코어 SoC 및 플랫폼

차별성

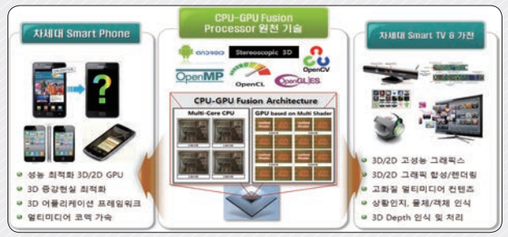
CPU가 GPU의 동작을 관리하는 이기종 컴퓨팅 기술의 관리적인 부분을 최적화하여 CPU와 메모리간 데이터 이동을 최소화

기존 HSA(Heterogeneous System Architecture) 아키텍쳐 보다 진보한 퓨전 메모리 공유(fusion memory shared) 기술 및 GPU 코어 할당 기술

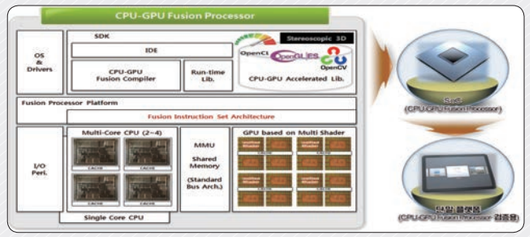
해외주요기관

AMD(Vega GPU/GPU), NVIDIA(Titan Volta/GPU)

개발 내용

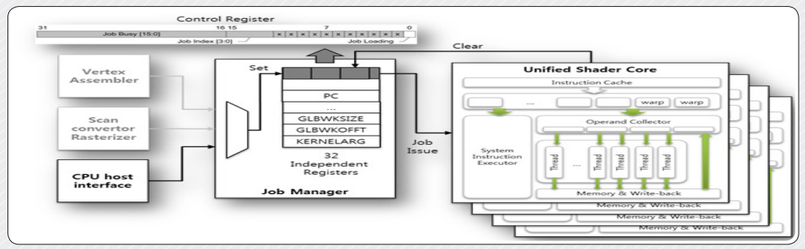


CPU-GPU 퓨전 프로세서 개념도



CPU-GPU 퓨전 프로세서 내부 구조(개념도)

연구원 보유(개발) 핵심기술



CPU-GPU간 핵심 제어 기술

연구원 보유(개발) 핵심기술

KETI 핵심기술

멀티미디어 가전 및 OTT, 차세대 스마트폰에 적용하기 위한 CPU-GPU기반의 Cortex A53쿼드기반 18.6GOPS급 GPU 처리 기술

18.6GFLOPS 급 처리가 가능한 GPU 연산속도 및 MHz당 0.489mW의 저전력 성능 확보

OTT, 셋톱박스, 차량용 블랙박스 제품에 탑재 가능한 기술 완성도 확보

차별성

ARM Cortex-A53의 옥타코어 CPU와 Nexell의 GPU 코어를 통합한 모바일용 SoC 프로세서*

* 삼성, 28nm High-K Metal Gate 공정 적용

관련기술 보유 IP

CPU와 GPU간의 캐쉬 공유 기반의 효율적 협업 구조(국내/등록/2014)

그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 방법(국내/등록/2016)

Task allocation system and method for operation of graphic processing unit(국외/출원/2014)

기술사업화 성과

사 업 화 : "상용 RAM Processor 적용" (2014)

"Set Top Box" (2015)

Business Model

● 모바일 기반 지능형 처리 서비스용 핵심 코어 시장

- 모바일, 영상 및 가전 디바이스 산업에서 CPU-GPU 원천기술이 필요한 기업들에게 IP 단위로 기술 공급
- VR용 가상용 서비스(스마트패드)(GPU기반가속기술)
- OTT 셋톱박스 및 블랙박스-차선 및 차량인지 HW처리분야(AP)

● 수요 예상 기업

- 반도체설계기업, 가전제조사, OTT 서비스 기업 등

* 협력 연구기관 D社는 저전력 소형화를 확보한 CPU-GPU 퓨전프로세서 기반 기술을 적용한 OTT 셋톱박스와 스마트 리모콘 제품을 사업화 완료

관련 연구 분야

Core Technology	Related Technology		
멀티미디어 퓨전(CPU-GPU) 프로세서 융합시스템/SoC플랫폼 김동순	**가속엔진 SoC** 지능형 멀티모달 신호처리 SoC 융합시스템/SoC플랫폼 김병수	**데이터 전송 SoC** 4K급 모바일 기기용 AV 전송 SoC 정보통신미디어/모빌리티플랫폼 박규호	**영상처리 시스템** UHD/S3D급 비디오 서버 정보통신미디어/지능형영상처리 김제우
	플랫폼 엔진 이트레이닝시스템용 공통플랫폼 코어 엔진 정보통신미디어/VRAR 정광모		

(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2014년09월19일
(11) 등록번호 10-1442643
(24) 등록일자 2014년09월15일

(51) 국제특허분류(Int. Cl.)
G06F 13/14 (2006.01) *G06F 12/02* (2006.01)
(21) 출원번호 10-2013-0048061
(22) 출원일자 2013년04월30일
심사청구일자 2013년04월30일
(56) 선행기술조사문헌
JP2011175624 A*
KR1020040106472 A*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
전자부품연구원
경기도 성남시 분당구 새나리로 25 (야탑동)
(72) 발명자
황태호
서울 송파구 문정로 83, 128동 402호 (문정동, 문정래미안아파트)
김동순
경기 성남시 분당구 정자일로 248, 606동 3203호 (정자동, 파크뷰)
(74) 대리인
특허법인지명

전체 청구항 수 : 총 13 항 심사관 : 김세영

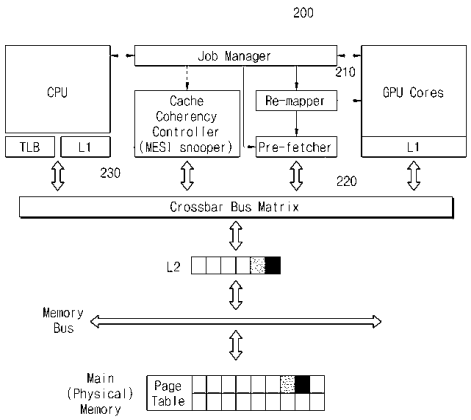
(54) 발명의 명칭 **CPU와 GPU 간의 협업 시스템 및 그 방법**

(57) 요 약

본 발명은 CPU와 GPU 간의 효율적인 협업 구조에 관한 것으로서, GPU를 제어하는 별도의 유닛을 통해 CPU의 로드를 경감시키고 GPU에 작업을 할당함에 있어서 직접적인 데이터의 복사가 없이 작업에 사용할 데이터의 주소 영역에 대한 정보만 제공되도록 함으로써, CPU와 GPU 간의 협업의 효율성을 높인 CPU와 GPU 간의 협업 시스템 및 그 방법을 제공한다.

또한 CPU와 GPU 간의 캐시일관성 유지를 위해 종래의 멀티 CPU 간의 캐시일관성 유지에 사용되는 프로토콜을 확장한 프로토콜을 제공하여 CPU와 GPU 간의 캐시 불일치 해소에 적합한 캐시일관성 유지 방법을 제공한다.

대 표 도 - 도2



이 발명을 지원한 국가연구개발사업
과제고유번호 10041664
부처명 지식경제부
연구관리전문기관 한국산업기술평가원
연구사업명 산업융합원천기술개발사업
연구과제명 멀티 Shader GPU 통합형 멀티 코어 퓨전 프로세서 원천 기술 개발
기 여 율 1/1
주관기관 전자부품연구원
연구기간 2012.06.01 ~ 2013.05.31

특허청구의 범위

청구항 1

CPU와 GPU 간의 협업 시스템에 있어서,

상기 CPU가 요청하는 작업을 전달받아 상기 GPU에 요청하며, 상기 GPU가 처리한 작업 결과를 상기 CPU로 전달하는 작업관리부; 및

상기 GPU의 주소 공간과 메인 메모리의 주소 공간의 매핑을 보조하는 주소매핑부를 포함하되,

상기 작업관리부는

상기 CPU가 요청하는 작업에 해당하는 코드 정보 및 상기 작업을 수행하기 위하여 필요한 데이터의 주소 정보를 상기 CPU로부터 전달받는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 2

삭제

청구항 3

제1항에 있어서, 상기 작업관리부는

상기 GPU의 주소 공간과 상기 작업에 필요한 데이터의 주소 정보를 매핑한 테이블을 상기 주소매핑부에 로드하는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 4

제1항에 있어서, 상기 작업관리부는

코프로세서 인터페이스와 동일한 인터페이스로 상기 CPU와 연결된 것

인 CPU와 GPU 간의 협업 시스템.

청구항 5

제1항에 있어서, 상기 작업관리부는

상기 CPU가 요청한 작업을 상기 GPU의 각 코어에 분배하여 요청하고, 상기 GPU의 각 코어의 작업 상태를 모니터링하는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 6

제1항에 있어서,

상기 GPU가 처리 중인 데이터 다음에 처리되어야할 데이터를 상기 메인 메모리로부터 캐시 메모리로 가져오는 프리페처

를 더 포함하는 CPU와 GPU 간의 협업 시스템.

청구항 7

제6항에 있어서, 상기 프리페처는

상기 작업관리부로부터 작동신호를 입력받으면, 상기 GPU에 필요한 데이터를 상기 메인 메모리에서 상기 캐시 메모리로 가져오고 처리완료된 데이터를 상기 캐시 메모리에서 제거하는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 8

제1항에 있어서,

상기 CPU의 캐시 메모리에 저장된 데이터와 상기 GPU의 캐시 메모리에 저장된 데이터를 일치시키는 캐시일관성제어부

를 더 포함하는 CPU와 GPU 간의 협업 시스템.

청구항 9

제8항에 있어서, 상기 작업관리부는

상기 CPU의 캐시 메모리에 저장된 데이터와 상기 GPU의 캐시 메모리에 저장된 데이터를 일치시킬 필요가 있는지 여부를 확인하고, 데이터 일치가 필요하면 상기 캐시일관성제어부를 작동시키는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 10

CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계;

상기 GPU의 주소 공간을 메인 메모리의 주소 공간과 매핑하는 단계; 및

상기 GPU가 처리한 작업 결과를 상기 CPU로 전달하는 단계를 포함하되,

상기 CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계는

상기 CPU로부터 작업에 해당하는 코드 정보 및 작업에 필요한 데이터의 주소 정보를 전달받는 단계를 포함하는 것

인 CPU와 GPU 간의 협업 방법.

청구항 11

삭제

청구항 12

제10항에 있어서, 상기 CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계는

상기 전달받은 작업을 분배하여 상기 GPU의 각 코어에 요청하고, 상기 GPU의 각 코어의 작업 상태를 모니터링하는 단계를 포함하는 것

인 CPU와 GPU 간의 협업 방법.

청구항 13

제10항에 있어서, 상기 GPU의 주소 공간을 메인 메모리의 주소 공간과 매핑하는 단계는

상기 GPU의 주소 공간과 상기 작업에 필요한 데이터의 주소 정보를 매핑한 테이블을 생성하는 단계; 및

상기 테이블을 참조하여 상기 GPU가 주소를 변환하는 단계를 포함하는 것

인 CPU와 GPU 간의 협업 방법.

청구항 14

제10항에 있어서,

상기 GPU가 처리 중인 데이터 다음에 처리되어야할 데이터를 확인하는 단계; 및

상기 확인된 데이터를 상기 메인 메모리로부터 캐시 메모리로 가져오는 단계

를 더 포함하는 CPU와 GPU 간의 협업 방법.

청구항 15

제10항에 있어서,

상기 CPU의 데이터와 상기 GPU의 데이터를 일치시킬 필요가 있는 경우에 양 데이터를 일치시키기 위해 캐시일관성 제어 모듈을 작동시키는 단계

를 더 포함하는 CPU와 GPU 간의 협업 방법.

명 세 서

기 술 분 야

[0001] 본 발명은 CPU와 그래픽 프로세서(GPU) 간의 협업 시스템 및 그 방법에 관한 것으로서, 구체적으로는, CPU와 GPU 간의 효율적인 협업을 위한 메모리 구조와 관리 방법에 관한 것이다.

배 경 기 술

[0002] 최근 Samsung(社) Exynos, nVidia(社) Tegra, Texas Instrument(社) OMAP 등의 AP(Application Processor)에서 ARM cortex 계열의 멀티 CPU 및 nVidia 혹은 Imagination(社)의 SGX 계열의 멀티 GPU를 채택하여 one chip에 집적되는 추세이다.

[0003] 전통적으로 멀티 CPU의 경우 시스템 성능 향상을 위해 1차 혹은 2차 캐시를 공유하는 형태가 주류를 이루고 있다. 또한 각 CPU에 속한 캐시 간의 coherency를 위해 MESI(Modified, Exclusive, Shared, Invaild)와 같은 프로토콜이 채택되고 이를 위해 Snoop Control Unit(SCU)이 탑재되어 있다. 그리고 외부 메모리의 접근을 최소화하기 위해 write-back, write-once, write allocate 방식이 적용되었다.

[0004] Intel(社), AMD(社)가 주가 되어 PC 쪽에서 먼저 시작된 GP-GPU는 위에서 언급한 것과 같이 AP로 확장되어 one chip에 집적되고 있다. 공통적으로 하위 레벨의 캐시를 공유한다. 하지만 모바일 AP와 PC에서는 memory management의 방식에 큰 차이점이 존재한다.

[0005] 예컨대 AMD Fusion APU의 경우, CPU와 GPU 각각이 다른 page table을 가지고 동작을 한다. 이에 반해 ARM Mali

T604의 경우, Cortext A15과 같은 page table을 가지고 memory를 관리한다. 둘 중 어느 것이 더 좋은 것인지는 아직 검증이 되지 않고 있다.

[0006] 현재 CPU/GPU 집적 시스템에서 CPU는 Bridge(PC) 혹은 Bus(AP)를 통해 GPU를 제어하고 있다. 일반적으로 GPU는 주로 CPU를 통해 처리하여야할 작업들의 코드와 데이터를 메모리 인터페이스를 통해 위임받고, 이를 GPU 로컬 메모리에 복사한 후, GPU는 이를 처리하여 CPU의 메인 메모리에 결과를 다시 복사하는 구조이다. 이를 위해 현재 CPU-GPU 집적 시스템에서 운영체제의 소프트웨어 드라이버가 CPU에서 브릿지 혹은 버스 인터페이스를 통해 GPU를 제어하는 구조이며, 메모리 공유 및 캐시 컨트롤러는 이 제어 구조와는 별개로 작동한다.

[0007] 하지만 이로 인해 시스템 성능이 저하되기 때문에 CPU와 GPU 간의 직접적인 inter-processor communication이 필요하고 이를 위한 control unit이 별도로 추가되어야 한다. 또한 캐시 공유에 있어 CPU와 GPU가 별도의 page table을 가지는 것과 공통의 page table을 가지는 것 사이의 검증이 필요하다.

발명의 내용

해결하려는 과제

[0008] 본 발명은 전술한 문제점을 해결하기 위하여, 별도의 제어 모듈을 통해 GPU를 제어하여 CPU의 로드를 감소시킬 수 있는 CPU와 GPU 간의 협업 시스템 및 방법을 제공하는 것을 목적으로 한다.

[0009] 또한 멀티 프로세서 간의 캐시일관성 문제를 해소하는 종래의 프로토콜을 확장하여, CPU와 GPU 간의 캐시일관성 유지에 효율적인 캐시일관성 제어 모듈을 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0010] 본 발명은 CPU와 GPU 간의 협업 시스템에 있어서, 상기 CPU가 요청하는 작업을 전달받아 상기 GPU에 요청하며, 상기 GPU가 처리한 작업 결과를 상기 CPU로 전달하는 작업관리부; 상기 GPU의 주소 공간과 메인 메모리의 주소 공간의 매핑을 보조하는 주소매핑부; 상기 GPU가 처리 중인 데이터 다음에 처리되어야할 데이터를 상기 메인 메모리로부터 캐시 메모리로 가져오는 프리페처; 및 상기 CPU의 캐시 메모리에 저장된 데이터와 상기 GPU의 캐시 메모리에 저장된 데이터를 일치시키는 캐시일관성제어부를 포함하는 CPU와 GPU 간의 협업 시스템을 제공한다.

[0011] 본 발명의 일면에 따르면, 상기 작업관리부는 상기 CPU가 요청하는 작업에 해당하는 코드 정보 및 상기 작업을 수행하기 위하여 필요한 데이터의 주소 정보를 상기 CPU로부터 전달받는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0012] 본 발명의 다른 일면에 따르면, 상기 작업관리부는 상기 GPU의 주소 공간과 상기 작업에 필요한 데이터의 주소 정보를 매핑한 테이블을 상기 주소매핑부에 로드하는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0013] 본 발명의 다른 일면에 따르면, 상기 작업관리부는 상기 CPU가 요청한 작업을 상기 GPU의 각 코어에 분배하여 요청하고 상기 GPU의 각 코어의 작업 상태를 모니터링하는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0014] 본 발명의 또 다른 일면에 따르면, 상기 프리페처는 상기 작업관리부로부터 작동신호를 입력받으면, 상기 GPU에 필요한 데이터를 상기 메인 메모리에서 상기 캐시 메모리로 가져오고 처리완료된 데이터를 상기 캐시 메모리에서 제거하는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0015] 본 발명의 또 다른 일면에 따르면, 상기 작업관리부는 상기 CPU의 캐시 메모리에 저장된 데이터와 상기 GPU의 캐시 메모리에 저장된 데이터를 일치시킬 필요가 있는지 여부를 확인하고, 데이터 일치가 필요하면 상기 캐시일관성제어부를 작동시키는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0016] 본 발명은 CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계; 상기 GPU의 주소 공간을 메인 메모리의 주소 공간과 매핑하는 단계; 상기 GPU가 처리한 작업 결과를 상기 CPU로 전달하는 단계; 상기 GPU가 처리 중인 데이터 다음에 처리되어야할 데이터를 확인하는 단계; 상기 확인된 데이터를 상기 메인 메모리로부터 캐시 메모리로 가져오는 단계; 및 상기 CPU의 데이터와 상기 GPU의 데이터를 일치시킬 필요가 있는 경우에 양 데이터를 일치시키기 위해 캐시일관성 제어 모듈을 작동시키는 단계를 포함하는 CPU와 GPU 간의 협업 방법으로 이용될 수 있다.

[0017] 본 발명의 일면에 따르면, 상기 CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계는, 상기 CPU로부터 작업

에 해당하는 코드 정보 및 작업에 필요한 데이터의 주소 정보를 전달받는 단계; 및 상기 전달받은 작업을 분배하여 상기 GPU의 각 코어에 요청하고, 상기 GPU의 각 코어의 작업 상태를 모니터링하는 단계를 포함하는 것인 CPU와 GPU 간의 협업 방법을 제공한다.

[0018] 본 발명의 다른 일면에 따르면, 상기 GPU의 주소 공간을 메인 메모리의 주소 공간과 매핑하는 단계는, 상기 GPU의 주소 공간과 상기 작업에 필요한 데이터의 주소 정보를 매핑한 테이블을 생성하는 단계; 및 상기 테이블을 참조하여 상기 GPU가 주소를 변환하는 단계를 포함하는 것인 CPU와 GPU 간의 협업 방법을 제공한다.

발명의 효과

[0019] 본 발명은 GPU의 작업을 관리하는 제어 모듈과 동기화되어 CPU가 GPU에 위임할 데이터 영역만을 공유하는 CPU와 GPU 간의 협업 시스템을 제공한다. 따라서 메모리 간의 복사없이 CPU가 사용하는 가상 주소 공간을 캐시에서 바로 접근할 수 있어 성능이 크게 향상된다.

[0020] 또한 캐시 레벨에서의 공유 구조에서 작업관리 모듈의 동작과 동기화되어 메인 메모리에서 캐시로의 프리페치를 효율적으로 제어할 수 있어, GPU의 직접적인 메인 메모리 접근을 최소화할 수 있는 장점을 가진다.

[0021] 그리고 CPU와 GPU의 캐시의 Coherency를 위한 제어는 작업에 따라 CPU가 작업관리 모듈을 통해 Enable/Disable 할 수 있는 구조이기 때문에 스누핑에 의한 성능 저하의 문제를 최적화할 수 있는 구조를 제공한다.

도면의 간단한 설명

[0022] 도 1은 종래의 CPU와 GPU 간의 협업 시스템의 구조를 나타낸 도면.

도 2는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템의 구조를 나타낸 도면.

도 3은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 작업관리부(Job Manager)의 구조를 나타낸 도면.

도 4는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 주소매핑부(Re-mapper)의 구조를 나타낸 도면.

도 5는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 프리페처(Pre-fetcher)의 구조를 나타낸 도면.

도 6 내지 도 10은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 캐시일관성제어부(Cache Coherency Controller)의 구조와 이를 설명하기 위한 도면.

도 11은 본 발명의 일실시예에 따른 CPU와 GPU 간의 확장된 협업 시스템의 구조를 나타낸 도면.

발명을 실시하기 위한 구체적인 내용

[0023] 본 발명의 이점 및 특징, 그리고 그것들을 달성하는 방법은 첨부되는 도면과 함께 상세하게 후술 되어 있는 실시예들을 참조하면 명확해질 것이다. 그러나 본 발명은 이하에서 개시되는 실시예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 것이며, 단지 본 실시예들은 본 발명의 개시가 완전하도록 하며, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 발명의 범주를 완전하게 알려주기 위해 제공되는 것이며, 본 발명은 청구항의 범주에 의해 정의된다.

[0024] 한편, 본 명세서에서 사용된 용어는 실시예들을 설명하기 위한 것이며 본 발명을 제한하고자 하는 것은 아니다. 본 명세서에서, 단수형은 문구에서 특별히 언급하지 않는 한 복수형도 포함한다. 명세서에서 사용되는 "포함한다(comprises)" 및/또는 "포함하는(comprising)"은 언급된 구성소자, 단계, 동작 및/또는 소자는 하나 이상의 다른 구성소자, 단계, 동작 및/또는 소자의 존재 또는 추가를 배제하지 않는다. 이하, 첨부된 도면을 참조하여 본 발명의 실시예를 상세히 설명하기로 한다.

[0025] 도 2는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템의 구조를 나타낸 도면이다.

[0026] 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템은 도 1에 도시된 종래의 CPU와 GPU 간의 협업 시스템에

있어서, 작업관리부(Job Manager, 200), 주소매핑부(Re-mapper, 210), 프리페처(Pre-fetcher, 220) 및 캐시일관성제어부(Cache Coherency Controller, 230)를 포함한다.

[0027] 작업관리부(Job Manager : CPU/GPU Inter Processor Communication Controller, 200)는 CPU가 bus 혹은 bridge를 통해 GPU를 구동하지 않고 직접 구동할 수 있도록 서로 간의 인터페이스를 지정하고 통신하도록 한다.

[0028] 작업관리부(200)는 CPU의 co-processor 인터페이스로 CPU와 밀접하게 연결되어, CPU에서 발생한 요청사항을 다수개의 GPU 코어에 나누어 요청하고 처리 결과를 CPU로 알려주는 역할을 담당한다. 따라서 작업관리부(200)는 이에 필요한 정보들을 CPU로부터 주고받을 수 있는 인터페이스를 포함한다.

[0029] 주소매핑부(Re-mapper : Memory Management Unit for GPU, 210)는 GPU의 주소 공간을 CPU가 사용하는 메인 메모리의 주소 공간으로 매핑을 보조하는 기능을 한다.

[0030] 기존 GPU는 가상 주소메모리 공간을 사용하지 않고, 직접 물리 주소에 접근한다. 설사 GPU가 별도의 MMU를 통해 가상 주소를 사용하더라도 CPU에서 사용하는 주소영역과는 다르기 때문에 GPU가 바라보는 주소 공간을 CPU가 사용하는 메인 메모리의 페이지 테이블을 이용하여 주소 공간으로 매핑하는 기능이 필요한데, 이 기능을 주소매핑부(210)가 담당하는 것이다. GPU 측에서는 주소매핑부(210)를 통해 Unified Shared Memory에 접근한다.

[0031] 프리페처(Pre-fetcher, 220)는 주 메모리와 L2 캐시로부터의 데이터 블록 패턴을 발견하여 이를 참조를 위한 패턴으로 받고 이를 필요한 데이터를 pre-fetch한다.

[0032] 캐시일관성제어부(Cache Coherency Controller, 230)는 CPU와 GPU가 서로 cache를 공유할 수 있도록 제어하는 기능을 한다. 이는 CPU뿐만 아니라 GPU와의 coherency도 유지할 수 있도록 기존 SCU(Snoop Control Unit)를 확장하여 설계한다.

[0033] 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에 의한 협업 과정은 아래와 같이 진행된다.

[0034] CPU는 GPU 코어용으로 컴파일된 코드와 데이터, 그리고 GPU 코어별로 분할된 데이터의 주소 및 오프셋 정보들을 작업관리부(200)의 정해진 인터페이스에 전달한다. 작업관리부(200)는 주어진 메인 메모리의 데이터 주소 정보를 GPU 주소공간으로 remapping 하여 주소매핑부(210)에 로드한다.

[0035] 작업관리부(200)는 주어진 주소 정보를 바탕으로 프리페처(220)를 작동시켜 메인 메모리에서 L2 캐시로 데이터를 미리 가져오고, CPU에서 cache coherency의 제어가 필요할 경우 캐시일관성제어부(230)를 작동시킨다.

[0036] 작업관리부(200)는 GPU의 각 코어에 작업을 할당하며, 할당된 작업이 GPU에서 처리되는 동안 이어서 다음에 처리할 데이터를 프리페처(220)를 통해 L2로 가져오고, 이미 처리된 데이터가 있을 경우 메인 메모리에 해당 캐시 데이터를 Flush시킨다.

[0037] GPU는 위임받은 작업이 끝나면 작업관리부(200)에게 완료 신호를 보내며, 작업관리부(200)는 CPU로 작업이 완료되었음을 전달한다.

[0038] 도 3은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 작업관리부의 구조를 나타낸 도면이다.

[0039] 기존 CPU가 GPU에게 작업을 위임하는 방식은 CPU가 제어의 요청을 시스템 버스를 통해 GPU의 Host Request Queue를 직접 관리하는 방식이다. 따라서 CPU는 GPU의 디바이스 드라이버 소프트웨어가 시스템 버스의 인터럽트 인터페이스를 통해 GPU의 동작을 지속적으로 관리하여야 하는 구조이다.

[0040] 반면에 본 발명은 이를 개선하기 위해 작업관리부의 별도의 하드웨어 장치를 통해 GPU가 작동하는 작업들의 관리를 위임하는 장치이다. 작업관리부를 통해 CPU는 GPU와 관련된 관리적인 로드가 크게 경감될 수 있다.

[0041] 작업관리부는 CPU의 co-processor 명령어와 같은 인터페이스로 연결되어 GPU가 실행해야할 작업 및 메모리 주소, 코어별 오프셋, 파라미터 등을 설정할 수 있는 레지스터들을 제공한다. 또한 GPU의 각 코어별 작업의 상태 및 동작을 모니터링할 수 있는 기능을 제공할 수 있다.

[0042] 작업관리부는 하나의 Host CPU의 인터페이스뿐만 아니라, 추가적인 인터페이스로 확장(최대 4개)이 가능하도록 설계되어 멀티 코어 프로세서와 다른 GPU 하드웨어와의 협업과 같은 이기종의 프로세서들과의 동작을 관리하는 역할을 수행할 수 있다.

[0043] 도 4는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 주소매핑부의 구조를 나타낸 도면이다.

[0044] OpenCL, OpenGL의 모델은 CPU-GPU의 시스템이 non-unified memory 구조에서 동작을 가정하고 설계되었다. 즉,

물리적으로 분리된 메모리를 가지고 있기 때문에 CPU가 사용하는 가상 메모리 주소공간과 GPU가 사용하는 메모리 주소공간은 서로 다르게 사용하도록 발전해왔다. 그러나 최근 CPU-GPU의 구조는 SoC상에서 공유 메모리 기반의 구조로 개발되면서 CPU와 GPU는 Unified Shared Memory 상에서 주소체계 및 변환에 대한 필요가 발생하였다. 이 문제를 해결하기 위한 통상적인 방법은 GPU도 CPU와 같이 각각의 TLB를 통해 메인 메모리 상의 같은 페이지 테이블을 참조하여 동일한 가상 메모리 주소공간을 사용하도록 하는 방법이다.

[0045] 일반적으로 GPU는 CPU로부터 대용량의 데이터 처리를 위임받고, 이를 순차적으로 나누어 병렬 처리하여 결과를 되돌려주는 방식이다. 이러한 점을 고려하였을 때 Unified shared memory 접근을 위해 TLB를 통해 공통의 주소 매핑 테이블을 공유하는 구조는 문제점이 있다. GPU는 큰 범위의 데이터를 전달받게 되고, GPU를 구성하는 각 코어들은 각각의 해당 공간을 TLB를 통해 변환하게 된다.

[0046] 그러나 제한적인 TLB의 크기와 GPU의 분할 및 순차적인 처리 특성상 TLB에 존재하는 변환 정보의 재사용률이 낮은 점을 고려할 때, GPU가 처리해야할 데이터가 클 경우 메인 메모리의 페이지 테이블에 접근하는 횟수가 증가할 수밖에 없다. 또한 많은 GPU 코어들이 각각의 TLB를 가지고 메모리 버스에 접근할 경우 더욱 많은 트래픽이 발생할 뿐만 아니라 구현의 복잡도 역시 높아진다.

[0047] 이러한 문제점을 개선하기 위해 본 발명은 다음의 접근 방식으로 설계된다. CPU가 GPU에게 작업을 위임하기 전에 필요한 데이터의 범위와 위치가 정해져 있기 때문에 CPU에서 OpenCL/OpenGL API를 통한 드라이버는 GPU로 넘겨질 메모리를 가능한 연속된 페이지에 allocation하고, 해당 페이지의 물리적 주소를 연속된 GPU의 가상주소로 매핑하는 테이블을 주소매핑부에 로딩한다. 이때 데이터가 연속된 페이지에 위치하지 않고 페이지 단위로 fragmentation되었으면 이 페이지 정보를 GPU를 위한 연속된 가상 주소공간으로 remapping하여 주소 매핑 테이블에 반영한다.

[0048] 주소 매핑 테이블에는 GPU에 넘겨질 모든 데이터의 페이지주소 정보들이 포함되어 GPU는 주소변환을 위한 추가적인 메모리 접근 없이 주소매핑부에 로딩된 매핑 테이블의 정보를 참조하여 주소변환을 진행한다.

[0049] 주소매핑부의 주소변환은 GPU의 각 코어의 개수만큼 구현된 translator 장치에 의해 매핑 테이블을 참조하여 수행되고, 변환된 주소 정보로 cache controller를 통해 Unified Shared Memory로 접근한다.

[0050] 도 5는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 프리페처의 구조를 나타낸 도면이다. GPU는 위임받은 작업을 분할하여 병렬적으로 그리고 순차적으로 처리하게 되고, 본 발명은 이를 보다 효율적으로 관리하기 위해 도 5와 같은 구조로 프리페처를 설계한다.

[0051] 작업관리부를 통해 GPU가 동작을 시작하는 것과 함께 프리페처는 L2의 캐시 영역을 GPU의 코어가 한 번의 작업에 필요한 공간의 2배를 예약하고 이를 두 개의 windows로 구분한다. 첫 번째 윈도우에는 현재 GPU의 작업에 필요한 데이터를 로딩하고, 두 번째 windows의 영역에는 다음에 이어서 처리할 작업을 위한 데이터를 로딩하기 위해 예약한다.

[0052] 이렇게 예약된 window 영역은 L2의 캐시 컨트롤러가 기존의 eviction rule을 적용하지 않으며 두 개의 windows는 GPU의 메모리 latency hiding을 위해 전용으로 사용된다.

[0053] 도 6은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 캐시일관성제어부의 구조를 나타낸 도면이다.

[0054] 캐시일관성제어부는 멀티코어 CPU와 GPU와의 L1 cache간의 coherency를 위한 프로토콜과 더불어 프로토콜에 따른 각 코어간의 memory-to-cache, cache-to-cache의 데이터 전송, 그리고 앞서 설명한 pre-fetchering을 위한 L2 캐시의 제어를 담당한다.

[0055] 캐시일관성제어부는 Single-Core CPU를 위한 구조와 이를 확장하는 구조의 두 가지로 설계된다. 첫 번째 Single-core CPU와 GPU간의 unified memory상에서의 공유를 위한 coherency 모델은 도 7에 도시된 바와 같다.

[0056] 도 7에서 이를 위한 상태 변환의 프로토콜은 도 8과 같다. 도 8의 프로토콜의 특징은 기본적으로 L1 캐시 간의 데이터 전송 기반으로 한다. 그리고 GPU에게 작업을 위임한 CPU가 GPU의 동작처리 과정 중에 해당 데이터를 다시 접근하는 경우가 낮기 때문에, Invalidation 기반의 GPU와의 coherency를 위해 snooping을 최소화한다. 즉 데이터의 ownership뿐만 아니라, 캐시된 데이터 자체도 복사되도록 하는 방식이다. 따라서 GPU와 공유된 데이터는 하나의 copy만 L1 캐시에 존재하도록 한다.

[0057] 그러나 멀티코어 CPU와 GPU를 위한 구조는 CPU간의 coherency의 프로토콜과 함께 동작하여야 하기 때문에 보다

복잡하다. 이를 위해 MOESI기반의 Dragon 프로토콜을 확장한다.

[0058] 도 9는 확장된 프로토콜에 필요한 상태들의 정의를 보여준다. RD의 상태가 추가되고 INV_REQ의 invalidation request가 추가된다. RD의 상태는 GPU가 데이터를 자신의 cache에 로딩 후, 데이터를 쓰기를 진행할 때의 상태를 나타낸다. 그리고 CPU간의 공유와 GPU와의 공유를 구분하기 위한 condition이 추가되는데 이것은 앞서 설명한 주소매핑부를 통해 제공된다. 주소매핑부는 자신의 테이블을 참조하여 접근하는 데이터의 경우 condition r을 true로 설정한다. 도 9에서 정의된 상태를 이용하여 설계된 coherency 프로토콜은 도 10과 같다.

[0059] 도 10에서 프로토콜은 기본적으로 앞서 설명한 Single-core CPU에서와 같이 GPU와의 공유되는 데이터는 기본적으로 invalidation을 기본으로 한다. 이것은 CPU가 위임한 작업을 위한 데이터에 대하여 CPU가 공유하여 쓰고자 할 때 update를 최소화하기 위해 기본적으로 GPU는 CPU의 공유된 캐시라인들을 invalidation하도록 한다.

[0060] 이러한 프로토콜을 포함하는 캐시일관성제어부의 개략적은 구조는 도 6에 도시된 바와 같고 캐시일관성제어부는 크게 세 가지 부분으로 구성된다.

[0061] 첫 번째는 앞서 설명한 프로토콜의 상태 변화를 조정하기 위한 comparator이다. comparator는 GPU와 CPU의 L1 cache controller로부터 주소와 line의 상태를 입력받아 이들의 상태를 관리한다.

[0062] 두 번째는 cache-to-cache 데이터 전송 unit이다. 이 unit은 comparator로부터 L1 cache간의 데이터 전송이 필요할 경우 이들 간의 데이터 전송을 담당한다.

[0063] 세 번째는 L2 cache controller이다. L2 controller는 통상적인 cache eviction rule을 적용하여 L2를 관리할 뿐만 아니라, 앞서 설명한 프리페처로부터 요청이 있을 경우 L2를 필요한 크기의 영역으로 partitioning하여 GPU의 프리페칭을 위해 필요한 메모리전송을 수행한다.

[0064] 도 11은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템이 확장된 시스템을 나타낸 도면으로, 도 11에 도시된 협업 시스템은 두 개의 CPU와 GPU가 메모리를 공유하는 구조이다.

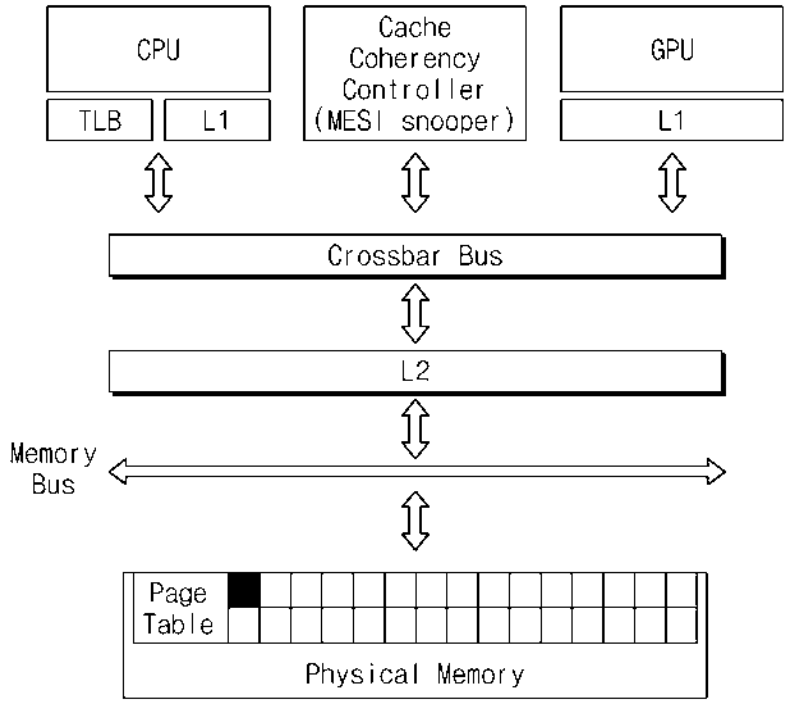
[0065] 전술한 CPU와 GPU 간의 협업 시스템의 구조는 L2 뿐만 아니라 L3 캐시를 통한 공유 구조로도 확장이 가능하며, 단일 CPU 뿐만 아니라 멀티 CPU와 GPU 간의 협업 구조로도 확장이 가능하다.

[0066] 멀티 CPU와 GPU는 L2 캐시는 각각 가지고 있으며, L3는 공유하는 구조이다. 작업관리부는 앞서 설명한 구조에서처럼 CPU와의 인터페이스를 통해 작동한다. 그러나 캐시일관성제어부는 CPU간의 메모리 공유를 위해 항상 동작하여야 한다.

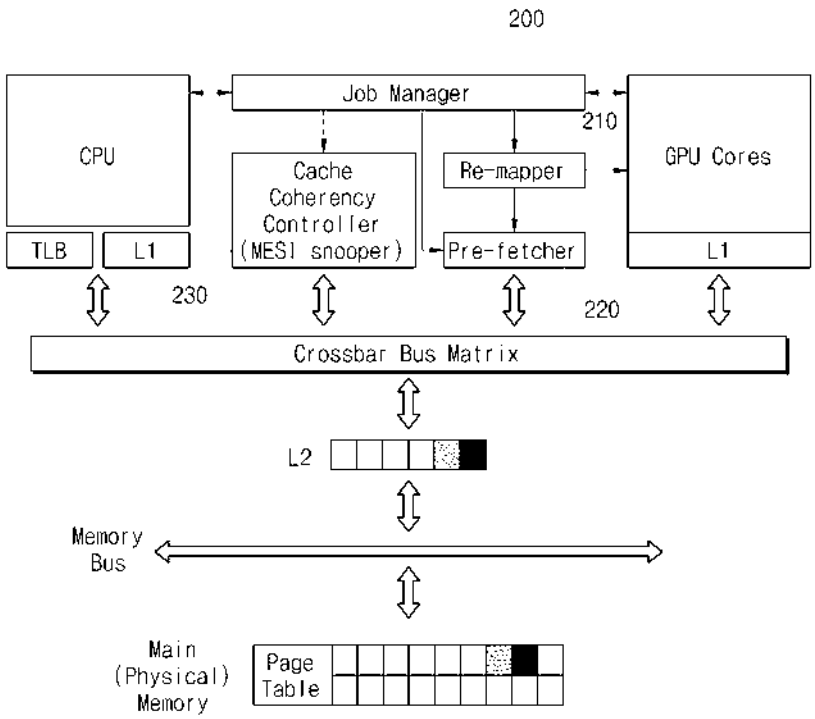
[0067] 이상의 설명은 본 발명의 기술적 사상을 예시적으로 설명한 것에 불과한 것으로서, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자라면, 본 발명의 본질적 특성을 벗어나지 않는 범위에서 다양한 수정 및 변형이 가능하다. 따라서, 본 발명에 표현된 실시예들은 본 발명의 기술적 사상을 한정하는 것이 아니라, 설명하기 위한 것이고, 이러한 실시예에 의하여 본 발명의 권리범위가 한정되는 것은 아니다. 본 발명의 보호 범위는 아래의 특허청구범위에 의하여 해석되어야 하고, 그와 동등하거나, 균등한 범위 내에 있는 모든 기술적 사상은 본 발명의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

도면

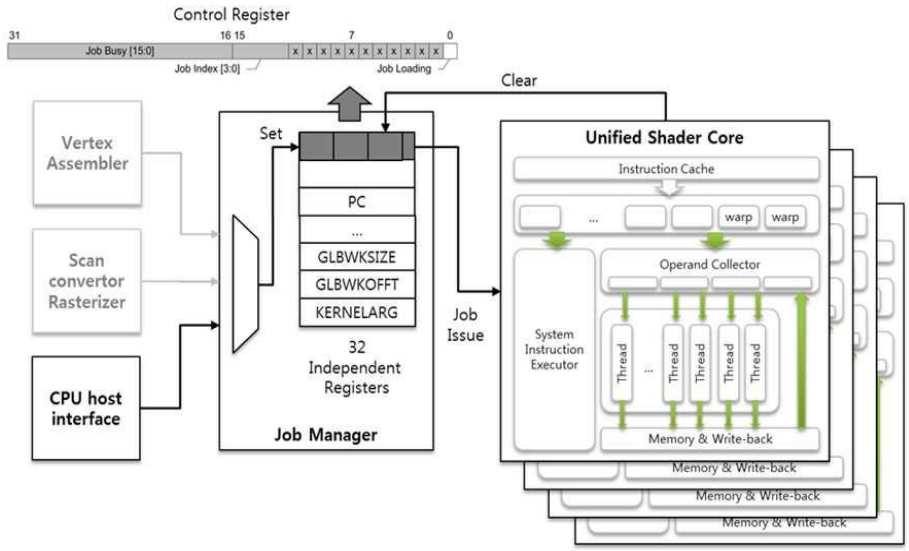
도면1


CPU
TLB
L1
Cache Coherency Controller (MESI snooper)
GPU
L1
Crossbar Bus
L2
Memory Bus
Page Table
Physical Memory

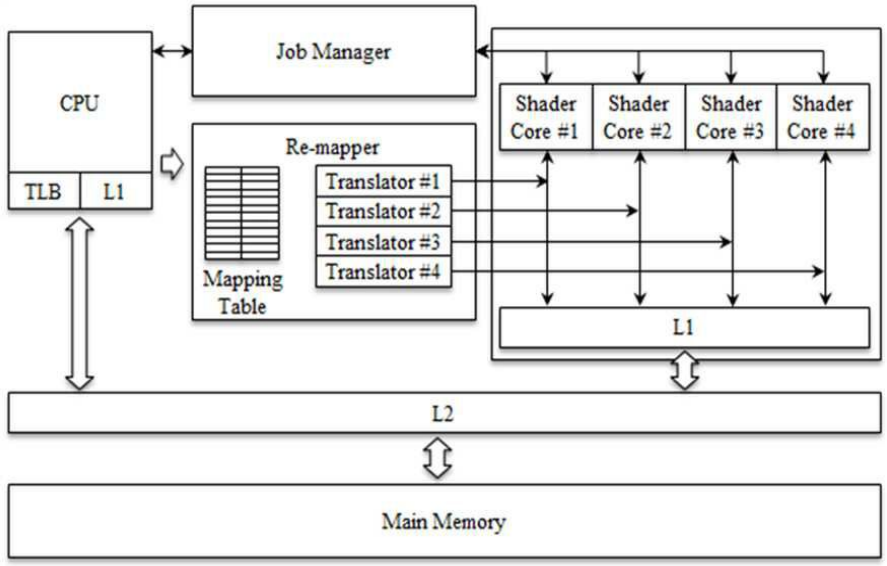
도면2


200
CPU
TLB
L1
Job Manager
210
Cache Coherency Controller (MESI snooper)
Re-mapper
Pre-fetcher
GPU Cores
L1
230
220
Crossbar Bus Matrix
L2
Memory Bus
Main (Physical) Memory
Page Table

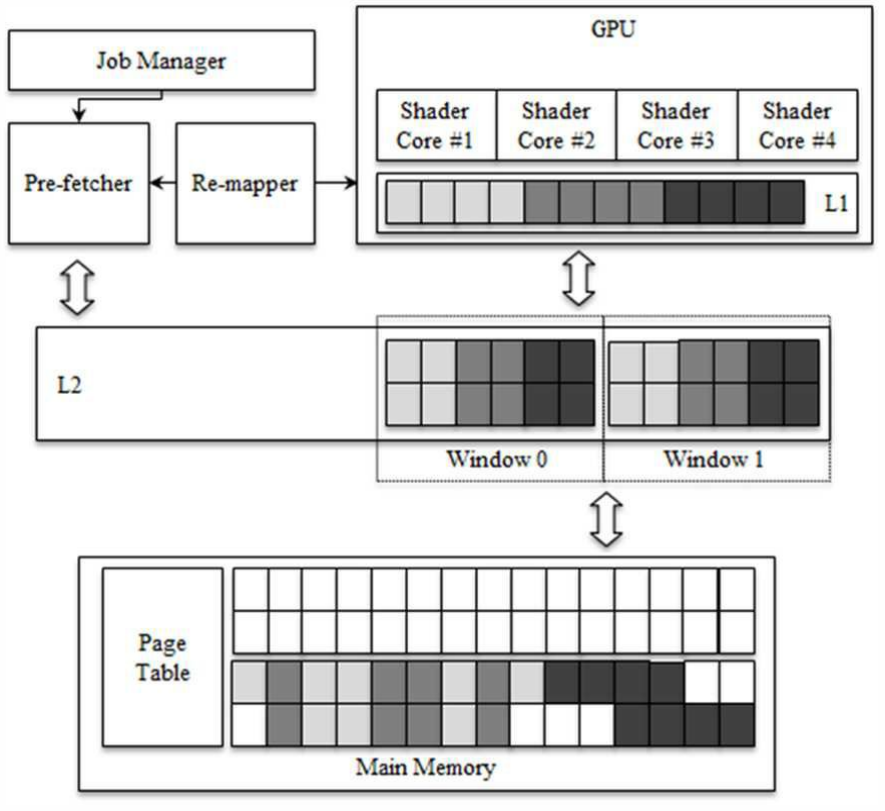
도면3


Control Register
31
16 15
7
0
Job Busy [15:0]
x x x x x x x x x x x x x x x x
Job Index [3:0]
Job Loading
Clear
Vertex Assembler
Scan convertor Rasterizer
CPU host interface
Set
PC
...
GLBWKSIZE
GLBWKOFFT
KERNELARG
32 Independent Registers
Job Manager
Job Issue
Unified Shader Core
Instruction Cache
warp
warp
Operand Collector
System Instruction Executor
Thread
Memory & Write-back
Memory & Write-back
Memory & Write-back
Memory & Write-back

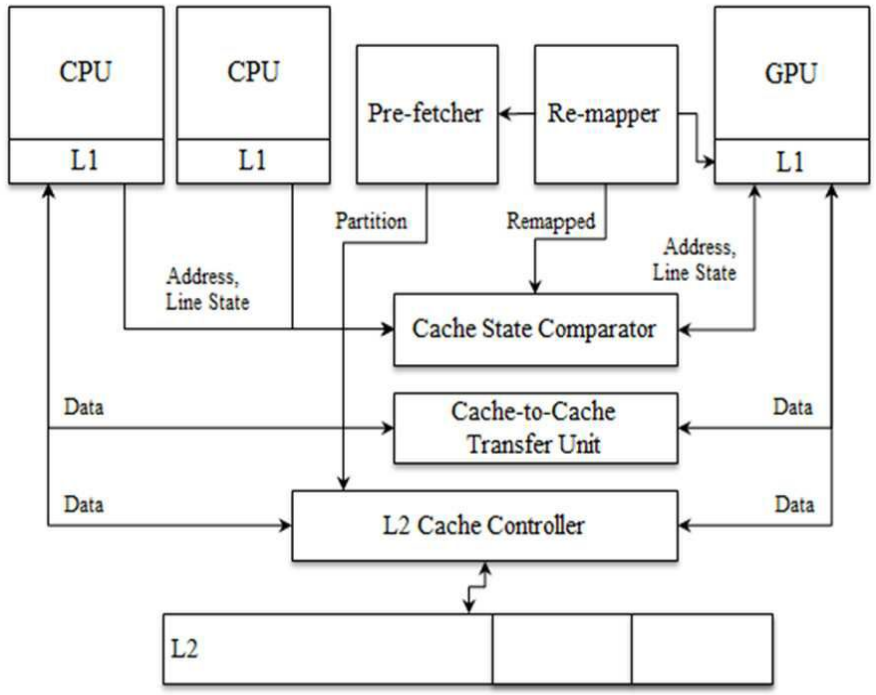
도면4


CPU
TLB
L1
Job Manager
Re-mapper
Translator #1
Translator #2
Translator #3
Translator #4
Mapping Table
Shader Core #1
Shader Core #2
Shader Core #3
Shader Core #4
L1
L2
Main Memory

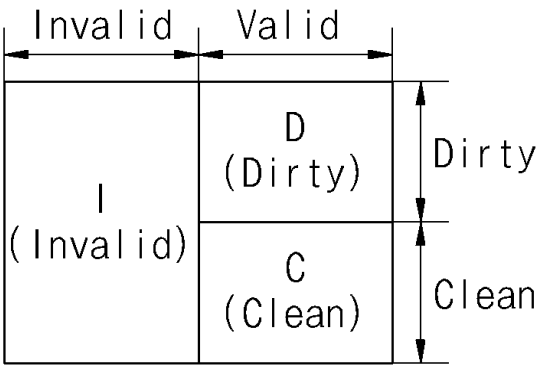
도면5


Job Manager
GPU
Shader Core #1
Shader Core #2
Shader Core #3
Shader Core #4
Pre-fetcher
Re-mapper
L1
L2
Window 0
Window 1
Page Table
Main Memory

도면6


CPU
CPU
Pre-fetcher
Re-mapper
GPU
L1
L1
L1
Partition
Remapped
Address, Line State
Address, Line State
Cache State Comparator
Data
Cache-to-Cache Transfer Unit
Data
Data
L2 Cache Controller
Data
L2

도면7



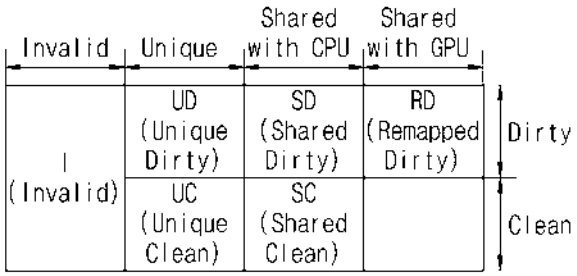
D: Dirty, must be written back
C: Clean
Invalid: Invalid

도면8

- LOAD : a read from processor
- STORE : a write from processor
- L_REQ : a load request to cache controller
- state() : change state to the next
- supply() : supply local cached data to others
- write_back() : write local cached data to next level cache

Current State	Request			
	LOAD	STORE	L_REQ	Eviction
I	L_REQ state(C)	L_REQ state(D)	-	-
C	state(C)	state(D)	supply() state(I)	state(I)
D	state(D)	state(D)	supply() state(I)	write_back() state(I)

도면9



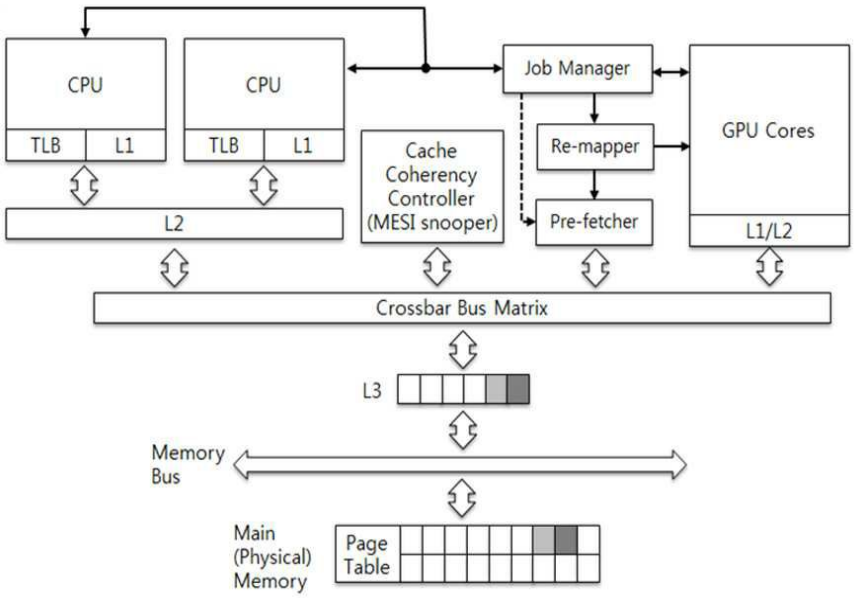
UD: Not shared, dirty, must be written back
SD: Shared with CPU, dirty, must be written back to memory
UC: Not shared, clean
SC: Shared with CPU, no need to write back, may be clean or dirty
RD: Shared with GPU, must be written back to memory
Invalid: Invalid

도면10

- LOAD : a read from processor
- STORE : a write from processor
- L_REQ : a load request to cache controller
- UP_REQ : an update request to cache controller
- INV_REQ : an invalidate request to cache controller
- s : condition, true if shared
- r : condition, true if remapped
- state() : change state to the next
- supply() : supply local cached data to others
- update() : update local copy from next level cache or others
- write_back() : write local cached data to next level cache

Current State	Request					
	LOAD	STORE	L_REQ	UP_REQ	INV_REQ	Eviction
I	L_REQ If ~s, state(UC) If s, state(SC)	L_REQ If ~s, state(UD) If s and ~r, UP_REQ and state(SD) If s and r, INV_REQ and state(RD)	-	-	-	-
SC	state(SC)	UP_REQ If ~s, state(UD) If s and ~r, UP_REQ and state(SD) If s and r, INV_REQ and state(RD)	state(SC)	update() state(SC)	state(I)	state(I)
UC	state(UC)	state(UD)	supply() state(SC)	-	-	state(I)
SD	state(SD)	UP_REQ If ~s, state(UD) If s and ~r, UP_REQ and state(SD) If s and r, INV_REQ and state(RD)	supply() state(SD)	update() state(SC)	write_back() state(I)	write_back() state(I)
UD	state(UD)	state(UD)	supply() state(SD)	-	-	write_back() state(I)
RD	state(RD)	UP_REQ state(RD)	supply() state(RD)	update() state(RD)	-	write_back() state(I)

도면11


CPU
CPU
TLB
L1
TLB
L1
L2
Job Manager
Re-mapper
Pre-fetcher
Cache
Coherency
Controller
(MESI snooper)
GPU Cores
L1/L2
Crossbar Bus Matrix
L3
Memory
Bus
Main
(Physical)
Memory
Page
Table

(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2016년03월15일
(11) 등록번호 10-1603711
(24) 등록일자 2016년03월09일

(51) 국제특허분류(Int. Cl.)
G06F 9/50 (2006.01) *G06F 13/14* (2006.01)
(21) 출원번호 10-2014-0049973
(22) 출원일자 2014년04월25일
심사청구일자 2014년04월25일
(65) 공개번호 10-2015-0123519
(43) 공개일자 2015년11월04일
(56) 선행기술조사문헌
KR1020120009919 A*
KR1020140006351 A*
KR101471303 B1
KR1020120031759 A
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
전자부품연구원
경기도 성남시 분당구 새나리로 25 (야탑동)
(72) 발명자
황태호
서울특별시 송파구 동남로 225 래미안파크팰리스 108동 601호
김동순
경기도 성남시 분당구 정자일로 248 파크뷰 606동 3203호
김선욱
경기도 남양주시 도농로 34 부영아파트 214동 2003호
(74) 대리인
특허법인지명

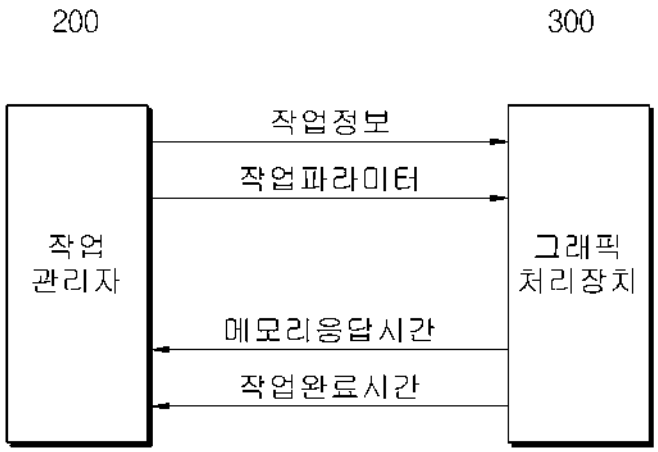
전체 청구항 수 : 총 8 항 심사관 : 유진태

(54) 발명의 명칭 **그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 방법**

(57) 요 약

본 발명은 그래픽 처리 장치(GPU)에 작업을 할당하고 그래픽 처리 장치가 할당받은 작업을 처리함에 있어서, 그래픽 처리 장치로부터 수신한 메모리 응답 시간에 기초하여 그래픽 처리 장치의 최적화된 코어 수를 조절하는 그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 방법을 제공한다. 본 발명에 따르면, 그래픽 처리 장치에서 최적화된 수의 코어가 작동하도록 함으로써 그래픽 처리 장치의 작업 처리 속도는 유지하면서 메모리 병목 현상으로 인한 작업 처리 지연을 감소시킬 수 있도록 한다.

대 표 도 - 도2



이 발명을 지원한 국가연구개발사업
과제고유번호 10041664
부처명 산업통산자원부
연구관리전문기관 산업기술평가관리원
연구사업명 산업기술원천기술개발사업
연구과제명 멀티 Shader GPU 통합형 멀티 코어 퓨전 프로세서 원천 기술 개발
기 여 율 1/1
주관기관 전자부품연구원
연구기간 2013.06.01 ~ 2014.05.31

명 세 서

청구범위

청구항 1

복수의 코어를 포함하며 중앙처리장치가 요청한 작업을 처리하는 그래픽처리장치; 및

상기 중앙처리장치가 요청한 작업을 상기 그래픽처리장치에 포함된 코어에 할당하고, 상기 그래픽처리장치로부터 일정 시간 간격으로 메모리 응답 시간 정보를 수신하며, 수신한 메모리 응답 시간 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 작업관리자를 포함하되,

상기 작업관리자는 상기 지정된 그래픽처리장치의 목표 코어 수가 상기 그래픽처리장치의 작동 코어 수보다 크면 상기 목표 코어 수와 상기 작동 코어 수의 차이에 해당하는 수의 작업이 할당되지 않은 코어에 작업을 할당하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 시스템.

청구항 2

제1항에 있어서, 상기 작업관리자는

상기 수신한 메모리 응답 시간 정보의 개수가 기설정된 개수 이상이 되면 수신한 메모리 응답 시간의 평균을 산출하고, 산출된 평균에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 시스템.

청구항 3

제2항에 있어서, 상기 작업관리자는

상기 산출된 평균이 제1 임계값보다 크면 상기 그래픽처리장치의 목표 코어 수를 감소시키고, 상기 산출된 평균이 제2 임계값보다 작으면 상기 그래픽처리장치의 목표 코어 수를 증가시키는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 시스템.

청구항 4

삭제

청구항 5

제1항에 있어서, 상기 작업관리자는

상기 지정된 그래픽처리장치의 목표 코어 수가 상기 그래픽처리장치의 작동 코어 수보다 작으면 작동 중인 코어 중 상기 목표 코어 수와 상기 작동 코어 수의 차이에 해당하는 수의 코어를 작업 할당에서 제외하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 시스템.

청구항 6

그래픽처리장치의 동작을 위한 작업관리자의 작업 할당 방법에 있어서,

복수의 코어를 포함하는 그래픽처리장치에 작업을 할당하는 단계;

상기 그래픽처리장치로부터 일정 시간 간격으로 메모리 응답 시간에 대한 정보를 수신하는 단계;

상기 수신한 메모리 응답 시간에 대한 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 단계; 및

상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계

를 포함하되,

상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계는,

상기 지정된 목표 코어 수가 작동 중인 코어 수보다 크면 작업이 할당되지 않은 코어에 작업을 할당하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 방법.

청구항 7

제6항에 있어서, 상기 수신한 메모리 응답 시간에 대한 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 단계는

상기 수신한 메모리 응답 시간의 평균을 산출하는 단계; 및

상기 평균을 기설정된 임계값과 비교하고, 비교 결과에 기초하여 목표 코어 수를 지정하는 단계를 포함하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 방법.

청구항 8

삭제

청구항 9

제6항에 있어서, 상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계는

상기 지정된 목표 코어 수가 작동 중인 코어 수보다 작으면 작업이 할당된 코어 중 할당된 작업이 적은 순서대로 코어를 작업 할당에서 제외하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 방법.

청구항 10

제6항에 있어서, 상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계는

상기 지정된 목표 코어 수가 작동 중인 코어 수와 동일하면 작업이 가장 적게 할당된 코어에 작업을 할당하거나 작업 완료 신호를 전송한 코어에 작업을 할당하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 그래픽 처리 장치(GPU, Graphics Processing Unit)의 작동을 제어하는 시스템 및 방법에 관한 것으로서, 구체적으로는, 중앙 처리 장치가 요청한 작업을 그래픽 처리 장치에 효율적으로 할당하는 그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 그 방법에 관한 것이다.

배 경 기 술

[0002] 그래픽 처리 장치(GPU, Graphics Processing Unit)는 컴퓨터에서 그래픽 연산 처리를 전담하는 반도체 코어 칩 또는 장치를 의미하는 것으로서, 현재 가장 널리 쓰이고 있는 NVIDIA 사의 Fermi GPU 구조에서는 GPU 자원을 최대한 활용하기 위해 모든 GPU 코어에 가능한 한 많은 작업을 할당하고 있다. 즉, 종래의 GPU 구조에서는 모든 코어에 최대 개수의 작업을 할당함으로써 GPU의 TLP(Thread Level Parallelism)을 최대로 끌어내고자 한다.

[0003] 종래의 GPU 구조에서의 이러한 작업 할당 정책은 모든 코어를 최대한 활용하는 것으로서, 이는 메모리 접근이 빈번하지 않은 GPU 프로그램을 수행할 때는 좋은 효과를 보여주나 메모리 접근이 빈번한 프로그램을 수행할 때는 메모리의 병목 현상으로 인하여 좋지 못한 성능을 보인다.

[0004] 즉, 메모리 접근 작업이 많은 GPU 프로그램을 수행할 경우 모든 코어에서 많은 수의 메모리 요청이 동시에 일어나므로 메모리에 병목 현상이 일어나게 되고, 메모리 병목현상으로 인해 GPU 코어는 요청한 데이터가 도착할 때까지 프로그램을 진행할 수 없으며, 이러한 지연(stall) 현상은 GPU 코어의 개수가 늘어날수록 증가한다. 이는 결국 개별 GPU 코어 성능을 크게 떨어뜨리고 개별 GPU 코어를 효율적으로 사용하지 못하게 하므로 전체적인 전력 소모량이 늘어나게 하는 문제점이 존재한다.

발명의 내용

해결하려는 과제

[0005] 본 발명은 전술한 문제점을 해결하기 위하여, 메모리 접근이 빈번한 프로그램에서 메모리 응답 시간(Memory Latency)을 기준으로 최적화된 GPU 코어 개수를 계산하여 활용함으로써 메모리 병목 현상을 줄이고 개별 GPU 코어의 성능을 높이는 그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 방법을 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0006] 본 발명은 복수의 코어를 포함하며 중앙처리장치가 요청한 작업을 처리하는 그래픽처리장치; 및 상기 중앙처리장치가 요청한 작업을 상기 그래픽처리장치에 포함된 코어에 할당하고, 상기 그래픽처리장치로부터 일정 시간 간격으로 메모리 응답 시간 정보를 수신하며, 수신한 메모리 응답 시간 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 작업관리자를 포함하는 그래픽 처리 장치의 동작을 위한 작업 할당 시스템을 제공한다.

[0007] 상기 작업관리자는 상기 수신한 메모리 응답 시간 정보의 개수가 기설정된 개수 이상이 되면 수신한 메모리 응답 시간의 평균을 산출하고, 산출된 평균에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정한다.

[0008] 상기 작업관리자는 상기 산출된 평균이 제1 임계값보다 크면 상기 그래픽처리장치의 목표 코어 수를 감소시키고, 상기 산출된 평균이 제2 임계값보다 작으면 상기 그래픽처리장치의 목표 코어 수를 증가시킨다.

[0009] 상기 작업관리자는 상기 지정된 그래픽처리장치의 목표 코어 수가 상기 그래픽처리장치의 작동 코어 수보다 크면 상기 목표 코어 수와 상기 작동 코어 수의 차이에 해당하는 수의 작업이 할당되지 않은 코어에 작업을 할당한다.

[0010] 상기 작업관리자는 상기 지정된 그래픽처리장치의 목표 코어 수가 상기 그래픽처리장치의 작동 코어 수보다 작으면 작동 중인 코어 중 상기 목표 코어 수와 상기 작동 코어 수의 차이에 해당하는 수의 코어를 작업 할당에서 제외한다.

[0011] 본 발명의 다른 일면에 따르면, 복수의 코어를 포함하는 그래픽처리장치에 작업을 할당하는 단계; 상기 그래픽처리장치로부터 일정 시간 간격으로 메모리 응답 시간에 대한 정보를 수신하는 단계; 상기 수신한 메모리 응답 시간에 대한 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 단계; 및 상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계를 포함하는 그래픽 처리 장치의 동작을 위한 작업 할당 방법을 제공한다.

발명의 효과

[0012] 본 발명은 최적화된 개수의 GPU 코어만을 활용하여 메모리 병목 현상을 줄이고 개별 GPU 코어의 성능을 향상시킬 수 있도록 한다. 따라서 향상된 개별 GPU 코어 성능으로 더 적은 수의 GPU 코어를 사용하면서도 종래기술과 같은 수준을 실행 속도를 유지하도록 하고 필요없는 GPU 코어는 사용하지 않음으로써 종래기술에 비해 전력 소모량을 감소시킬 수 있도록 한다.

도면의 간단한 설명

[0013] 도 1은 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 전체적인 구성을 나타낸 도면.

도 2는 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 작업관리자와 그래픽 처리 장치의 인터페이스를 나타낸 도면.

도 3과 도 4는 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 방법의 과정을 나타낸 도면.

발명을 실시하기 위한 구체적인 내용

[0014] 본 발명의 이점 및 특징, 그리고 그것들을 달성하는 방법은 첨부되는 도면과 함께 상세하게 후술 되어 있는 실시예들을 참조하면 명확해질 것이다. 그러나 본 발명은 이하에서 개시되는 실시예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 것이며, 단지 본 실시예들은 본 발명의 개시가 완전하도록 하며, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 발명의 범주를 완전하게 알려주기 위해 제공되는 것이며, 본 발명은 청구항의 기재에 의해 정의된다.

[0015] 한편, 본 명세서에서 사용된 용어는 실시예들을 설명하기 위한 것이며 본 발명을 제한하고자 하는 것은 아니다. 본 명세서에서, 단수형은 문구에서 특별히 언급하지 않는 한 복수형도 포함한다. 명세서에서 사용되는 "포함한다(comprises)" 및/또는 "포함하는(comprising)"은 언급된 구성소자, 단계, 동작 및/또는 소자에 하나 이상의 다른 구성소자, 단계, 동작 및/또는 소자의 존재 또는 추가함을 배제하지 않는다. 이하, 첨부된 도면을 참조하여 본 발명의 실시예를 상세히 설명하기로 한다.

[0016] 도 1은 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 전체적인 구성을 나타낸 것이다.

[0017] 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템은 도 1에 도시된 바와 같이 중앙 처리 장치(100), 그래픽 처리 장치(300) 및 DRAM(400)을 포함하며, 중앙 처리 장치(100)와 그래픽 처리 장치(300)의 사이에 작업 실행을 전반적으로 제어하는 작업관리자(200)를 포함한다.

[0018] 작업관리자(200)는 중앙 처리 장치(100)로부터 작업 처리를 위임받고 위임받은 작업을 그래픽 처리 장치(300)의 코어에 할당하며 그래픽 처리 장치(300)의 구동을 제어한다. 작업관리자(200)는 그래픽 처리 장치(300)의 연산 자원을 관리하는 기능을 탑재하고 있으며, 이를 이용하여 본 발명이 제안하는 그래픽 처리 장치(300)의 코어 개수 조절을 수행한다.

[0019] 도 2는 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 작업관리자(200)와 그래픽 처리 장치(300) 간의 인터페이스를 나타낸 것이다.

[0020] 작업관리자(200)는 중앙 처리 장치(100)로부터 위임받은 작업정보를 그래픽 처리 장치(300)에 전달하고, 그래픽 처리 장치(300)의 각 코어에 작업을 할당한다. 그리고 할당한 작업의 파라미터를 그래픽 처리 장치(300)로 전달한다.

[0021] 그래픽 처리 장치(300)는 작업관리자(200)로부터 할당받은 작업을 처리하고, 할당받은 작업의 처리가 완료되면 작업관리자(200)에게 작업 완료 신호를 전송한다. 또한, 그래픽 처리 장치(300)는 작업관리자(200)로부터 할당받은 작업을 수행하면서 일정 주기마다 메모리 응답 시간(Memory Latency)에 대한 정보를 작업관리자(200)에게 전달한다.

[0022] 작업관리자(200)는 그래픽 처리 장치(300)의 각 코어에서 전달받은 메모리 응답 시간을 일정 개수만큼 저장하고 저장된 메모리 응답 시간의 평균(AML, Average Memory Latency)을 계산한다. 작업관리자(200)는 계산된 평균 메모리 응답 시간(AML)에 기초하여 그래픽 처리 장치(300)의 작동 코어 개수를 조절하여 그래픽 처리 장치(300)의 코어가 최적화된 개수만큼 작동하도록 하며, 작업관리자(200)가 그래픽 처리 장치(300)의 작동 코어 개수를 조절하는 과정은 도 3과 도 4를 통해 구체적으로 설명한다.

[0023] 도 3과 도 4는 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 방법의 과정을 나타낸 것으로서, 도 3은 작업관리자가 최적화된 목표 코어 수를 지정하는 과정을 나타낸 것이고 도 4는 최적화된 목표 코어 수와 현재 작동 코어 수에 따라 그래픽 처리 장치에 작업 할당을 제어하는 과정을 나타낸 것이다.

[0024] 도 3에 도시된 바와 같이, 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 작업관리자는 중앙 처리 장치로 위임받은 작업을 그래픽 처리 장치의 각 코어에 할당하고, 일정 주기마다 그래픽 처리 장치로부터 각 코어의 메모리 응답 시간을 전달받는다.

[0025] 작업관리자는 그래픽 처리 장치의 각 코어에서 전달받은 메모리 응답 시간을 가지고 전체 그래픽 처리 장치에서의 평균 메모리 응답 시간(AML)을 계산한다(S300). 이때 작업관리자는 그래픽 처리 장치로부터 전달받은 메모리 응답 시간의 개수가 일정 개수 이상이 되면 평균 메모리 응답 시간(AML)을 계산할 수도 있다.

[0026] 작업관리자는 계산된 평균 메모리 응답 시간(AML)을 미리 지정된 응답 시간의 임계값과 비교하고 비교 결과에 따라 목표 코어 수를 조정하여 최적화된 개수의 코어가 작동할 수 있도록 한다.

[0027] 구체적으로, 작업관리자는 평균 메모리 응답 시간(AML)을 응답 시간의 최대 임계값인 제1 임계값과 비교하고(S320), 평균 메모리 응답 시간(AML)이 제1 임계값보다 크면 목표 코어 수를 감소시킨다(S340).

[0028] 그리고 평균 메모리 응답 시간(AML)이 제1 임계값보다 크지 않으면 평균 메모리 응답 시간(AML)을 응답 시간의 최소 임계값인 제2 임계값과 비교하고(S360), 평균 메모리 응답 시간(AML)이 제2 임계값보다 작으면 목표 코어 수를 증가시킨다(S380).

[0029] 평균 메모리 응답 시간(AML)이 제1임계값보다 크지 않고 제2 임계값보다 작지 않으면 목표 코어 수는 현재 목표 코어 수로 유지한다.

[0030] 즉, 본 발명은 평균 메모리 응답 시간(AML)을 미리 지정된 응답 시간의 임계값과 비교하고 비교 결과에 따라 목표 코어 수를 조정함으로써 그래픽 처리 장치의 코어가 최적화된 개수만큼 작동할 수 있도록 하며, 목표 코어 수 결정을 위한 제1 임계값과 제2 임계값은 실험적으로 구해질 수 있고 사용자에 의하여 임의로 설정될 수도 있다.

[0031] 도 4는 최적화된 목표 코어 수에 따라 그래픽 처리 장치의 코어에 작업을 할당하는 과정을 나타낸 것이다.

[0032] 작업관리자는 목표 코어 수가 지정되면 지정된 목표 코어 수를 현재 작동 중인 그래픽 처리 장치의 코어 수와 비교한다(S400).

[0033] 목표 코어 수가 현재 작동 중인 코어 수보다 크면(S410), 작업이 할당되지 않은 코어에 작업을 할당하여(S420) 작동 중인 코어 수가 최적화된 목표 코어 수와 동일하게 하거나 목표 코어 수에 근접하도록 한다.

[0034] 그리고 목표 코어 수가 현재 작동하는 코어 수보다 작으면(S430), 현재 작동하고 있는 코어 중 작업이 가장 적게 할당된 코어를 선택하고 선택된 코어를 작업 할당에서 제외한다(S440). 따라서 작업이 가장 적게 할당된 코어에는 추가적인 작업 할당을 하지 않고 해당 코어가 현재 할당된 작업만을 완료하고 작동을 중지하도록 한다.

[0035] 이때 목표 코어 수와 작동 중인 코어 수의 차이에 해당하는 수만큼의 코어를 작업이 적게 할당된 순서대로 선택하고 선택된 코어들을 작업 할당에서 제외할 수도 있다.

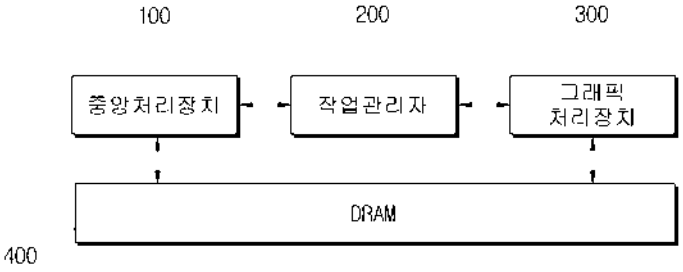
[0036] 현재 작동하고 있는 코어 수와 목표 코어 수가 동일하면 현재 작동 중인 코어 중에서 가장 적은 작업을 할당받은 코어를 지정한다(S450). 그리고 지정된 코어에 작업 할당이 바로 가능하다면(S460) 작업을 바로 할당한다. 지정된 코어에 이미 너무 많은 작업이 할당되어 있어 추가적인 작업 할당이 가능하지 않으면(460) 그래픽 처리 장치의 코어로부터 작업 완료 신호를 수신할 때까지 대기하고(S480), 작업 완료 신호를 수신하면 작업을 할당한다.

[0037] 이상의 설명은 본 발명의 기술적 사상을 예시적으로 설명한 것에 불과한 것으로서, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자라면, 본 발명의 본질적 특성을 벗어나지 않는 범위에서 다양한 수정 및 변형이 가

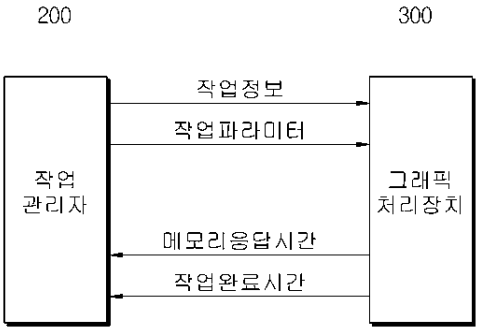
능하다. 따라서, 본 발명에 표현된 실시예들은 본 발명의 기술적 사상을 한정하는 것이 아니라, 설명하기 위한 것이고, 이러한 실시예에 의하여 본 발명의 권리범위가 한정되는 것은 아니다. 본 발명의 보호 범위는 아래의 특허청구범위에 의하여 해석되어야 하고, 그와 동등하거나, 균등한 범위 내에 있는 모든 기술적 사상은 본 발명의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

도면

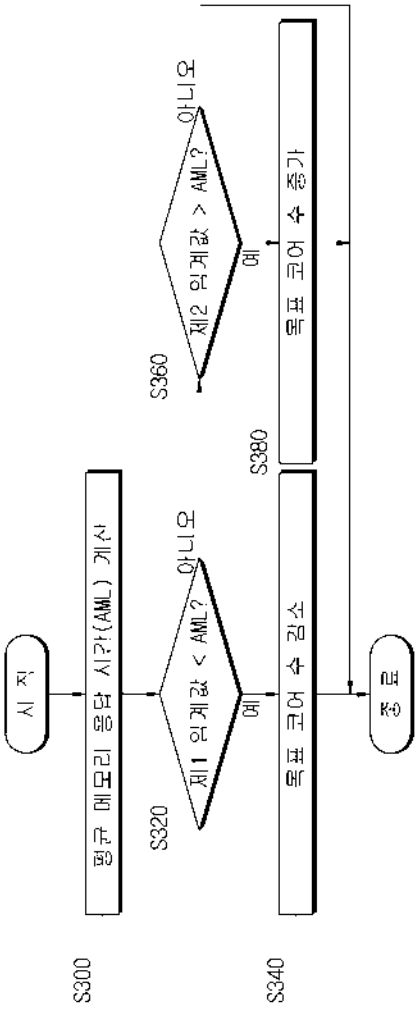
도면1



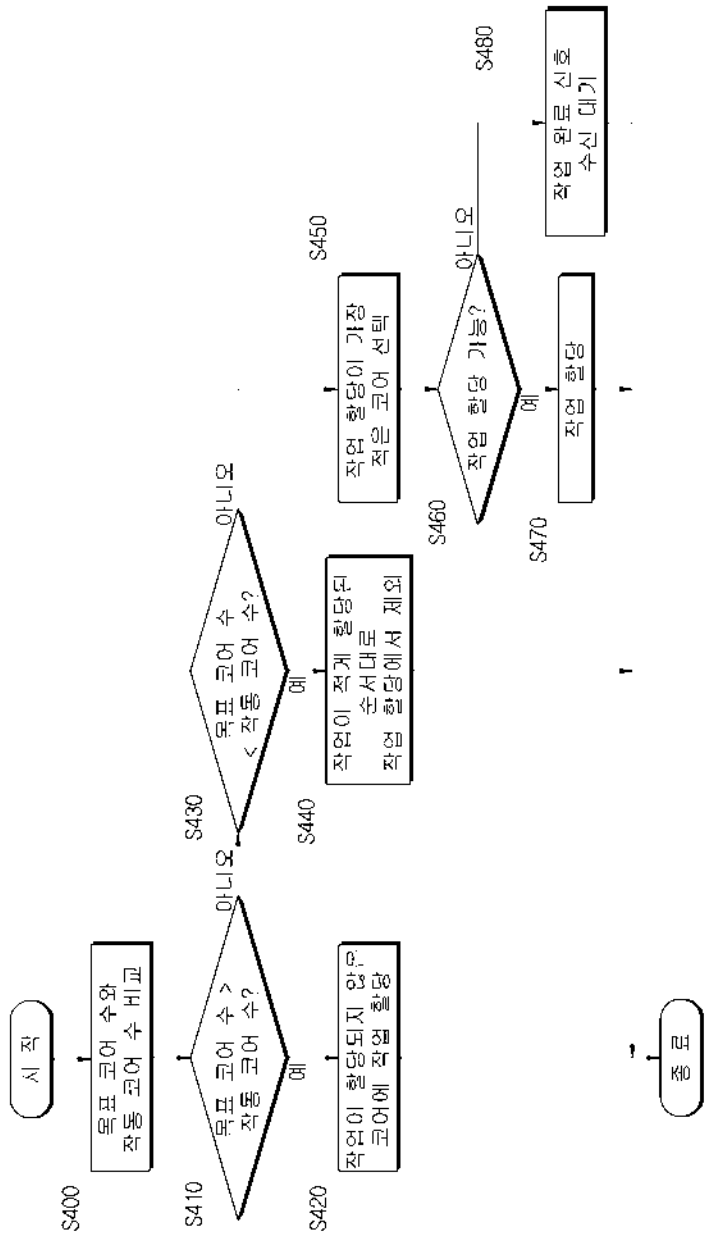
도면2



도면3


시 작
평균 메모리 응답 시간(AML) 계산
S300
제1 임계값 < AML?
S320
아니오
예
목표 코어 수 감소
S340
제2 임계값 > AML?
S360
아니오
예
목표 코어 수 증가
S380
종 료

도면4


시 작
목표 코어 수와
작동 코어 수 비교
S400
목표 코어 수 >
작동 코어 수?
S410
예
작업이 할당되지 않은
코어에 작업 할당
S420
아니오
목표 코어 수
< 작동 코어 수?
S430
예
작업이 적게 할당된
순서대로
작업 할당에서 제외
S440
아니오
작업 할당이 가장
적은 코어 선택
S450
작업 할당 가능?
S460
예
작업 할당
S470
아니오
작업 완료 신호
수신 대기
S480
종 료