(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2021년01월05일
(11) 등록번호 10-2197874
(24) 등록일자 2020년12월28일

(51) 국제특허분류(Int. Cl.)
G06F 9/46 (2006.01)
(21) 출원번호 10-2014-0115318
(22) 출원일자 2014년09월01일
심사청구일자 2019년04월04일
(65) 공개번호 10-2016-0027541
(43) 공개일자 2016년03월10일
(56) 선행기술조사문헌
KR1020140078394 A*
(뒷면에 계속)

(73) 특허권자
삼성전자주식회사
경기도 수원시 영통구 삼성로 129 (매탄동)
(72) 발명자
이경택
경기도 수원시 영통구 영통로 498 152동 1806호 (영통동,황골마을1단지아파트)
김승규
세종특별자치시 갈매로 480, 206동 403호 (어진동,한뜰마을아파트)
(뒷면에 계속)
(74) 대리인
특허법인 고려

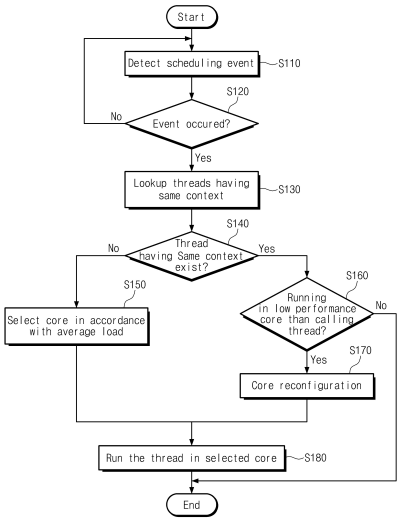
전체 청구항 수 : 총 10 항 심사관 : 유진태

(54) 발명의 명칭 **멀티-코어 프로세서를 포함하는 시스템 온 칩 및 그것의 쓰레드 스케줄링 방법**

(57) 요 약

본 발명의 실시 예에 따른 멀티-코어 프로세서를 포함하는 시스템 온 칩의 스케줄링 방법은, 상기 멀티-코어 프로세서에서 실행될 쓰레드의 스케줄링 요청을 검출하는 단계, 상기 멀티-코어 프로세서에서 실행 중인 쓰레드들 중에서 상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 콜링 쓰레드를 검출하는 단계, 그리고 상기 동일한 컨텍스트를 갖는 콜링 쓰레드가 실행되는 코어의 성능에 따라 상기 스케줄링 요청된 쓰레드를 실행할 코어를 재할당 또는 재설정하는 단계를 포함한다.

대 표 도 - 도5



(72) 발명자

박지은

경기도 시흥시 은행로 107 303동 1502호 (은행동,대우3차아파트)

장원준

경기도 성남시 분당구 서현로 181 201동 1102호 (이매동,이매촌한신아파트)

(56) 선행기술조사문헌

KR1020130093995 A*

KR1020100074920 A*

KR1020130080663 A

KR1020120017294 A

*는 심사관에 의하여 인용된 문헌

명 세 서

청구범위

청구항 1

멀티-코어 프로세서를 포함하는 시스템 온 칩의 스케줄링 방법에 있어서:

상기 멀티-코어 프로세서에서 실행될 쓰레드의 스케줄링 요청을 검출하는 단계;

상기 멀티-코어 프로세서에서 실행 중인 쓰레드들 중에서 상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 콜링 쓰레드를 검출하는 단계; 그리고

상기 동일한 컨텍스트를 갖는 콜링 쓰레드가 실행되는 코어의 성능으로 상기 스케줄링 요청된 쓰레드를 동기시키도록 상기 스케줄링 요청된 쓰레드를 실행할 코어를 재할당 또는 재설정하는 단계를 포함하는 스케줄링 방법.

청구항 2

제 1 항에 있어서,

상기 재할당 또는 재설정하는 단계에서, 상기 콜링 쓰레드가 실행되는 코어의 처리 속도가 상기 스케줄링 요청된 쓰레드가 실행되는 코어의 처리 속도보다 높은 경우, 상기 스케줄링 요청된 쓰레드는 상기 콜링 쓰레드가 실행되는 코어에서 실행되도록 마이그레이션되는 스케줄링 방법.

청구항 3

제 1 항에 있어서,

상기 재할당 또는 재설정하는 단계에서, 상기 콜링 쓰레드가 실행되는 코어의 처리 속도가 상기 스케줄링 요청된 쓰레드가 실행되는 코어의 처리 속도보다 높은 경우, 상기 스케줄링 요청된 쓰레드는 상기 콜링 쓰레드가 실행되는 코어와 동일한 처리 속도를 갖는 코어에서 실행되도록 마이그레이션되는 스케줄링 방법.

청구항 4

제 1 항에 있어서,

상기 재할당 또는 재설정하는 단계에서, 상기 콜링 쓰레드가 실행되는 코어의 처리 속도가 상기 스케줄링 요청된 쓰레드가 실행되는 코어의 처리 속도보다 높은 경우, 상기 스케줄링 요청된 쓰레드가 실행되는 코어의 처리 속도를 상기 콜링 쓰레드가 실행되는 코어의 처리 속도에 동기시키는 스케줄링 방법.

청구항 5

제 4 항에 있어서,

상기 콜링 쓰레드가 실행되는 코어와 상기 스케줄링 요청된 쓰레드가 실행되는 코어는 동종의 코어인 것을 특징으로 하는 스케줄링 방법.

청구항 6

제 1 항에 있어서,

상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 콜링 쓰레드를 검출하는 단계에서, 상기 동일한 콘텍스트를 갖는 콜링 쓰레드가 존재하지 않는 경우, 상기 스케줄링 요청된 쓰레드의 평균 부하를 계산하는 단계를 더 포함하는 스케줄링 방법.

청구항 7

제 6 항에 있어서,

상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 콜링 쓰레드가 존재하지 않는 경우, 상기 평균 부하의

크기를 참조하여 상기 스케줄링 요청된 쓰레드를 실행할 코어를 재할당하는 스케줄링 방법.

청구항 8

제 1 항에 있어서,

상기 멀티-코어 프로세서는 서로 다른 성능의 코어를 동시에 구동하는 이종 멀티-프로세싱(Heterogeneous Multi-Processing) 방식으로 구동되는 스케줄링 방법.

청구항 9

서로 다른 구동 속도를 갖는 복수의 코어들을 포함하는 멀티-코어 프로세서; 그리고

상기 멀티-코어 프로세서에 의해서 구동되는 운영 체제 및 응용 프로그램이 로드되는 워킹 메모리를 포함하되,

상기 운영 체제는 상기 응용 프로그램에서 발생하는 쓰레드의 스케줄링 요청에 따라 상기 쓰레드가 실행될 코어를 할당하는 스케줄러를 포함하며, 상기 스케줄러는 상기 멀티-코어 프로세서에서 이미 실행중인 복수의 쓰레드들 중에서 상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 쓰레드의 존재 여부에 따라 상기 스케줄링 요청된 쓰레드가 실행될 코어를 선택 또는 설정하되,

상기 스케줄러는 상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 쓰레드가 존재하지 않는 경우에는, 상기 스케줄링 요청된 쓰레드의 평균 부하에 따라 상기 복수의 코어들 중 어느 하나에 할당하는 시스템 온 칩.

청구항 10

제 9 항에 있어서,

상기 스케줄러는 상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 쓰레드의 존재 여부에 따라 상기 스케줄링 요청된 쓰레드가 현재 실행되는 코어의 처리 속도를 제어하는 시스템 온 칩.

발명의 설명

기 술 분 야

[0001] 본 발명은 반도체 장치에 관한 것으로, 좀 더 구체적으로는 멀티-코어 프로세서를 구비하는 시스템 온 칩 및 그것의 쓰레드 관리 방법에 관한 것이다.

배 경 기 술

[0002] 최근, 스마트폰, 테이블릿 PC, 디지털 카메라, MP3 플레이어, PDA 등과 같은 모바일 장치의 이용이 폭발적으로 증가하고 있다. 이러한 모바일 장치에서도 멀티미디어의 구동 및 각종 데이터의 처리량이 증가하면서, 고속 프로세서 및 대용량 저장 매체의 채용이 확대되고 있다. 모바일 장치에는 다양한 응용 프로그램(Application program)들이 구동된다. 다양한 응용 프로그램들을 구동하기 위하여, 모바일 장치에는 워킹 메모리(예를 들면, DRAM), 불휘발성 메모리, 그리고 응용 프로세서(Application Processor: 이하, AP)와 같은 반도체 장치들이 사용된다. 그리고 모바일 환경에서의 고성능화에 대한 요구에 따라 상술한 반도체 장치들의 집적도 및 구동 주파수는 날로 높아지고 있다.

[0003] 모바일 장치에는 응용 프로세서(Application Processor)에는 높은 성능을 제공하기 위한 멀티-코어 프로세서(Multi-Core Processor)가 사용되고 있다. 멀티-코어 프로세서는 동종 멀티-코어(Homogenous Multi-core)와 이종 멀티-코어(Heterogeneous Multi-core)로 구분될 수 있다. 동종 멀티-코어 프로세서(Homogenous Multi-core Processor)는 동일한 성능 또는 동일한 사이즈의 멀티-코어(Multi-Core)들이 포함될 수 있다. 이종의 멀티-코어(Heterogeneous Multi-core) 프로세서에는 서로 다른 사이즈 또는 성능의 코어들이 구비된다.

[0004] 특히, 이종 멀티-코어(Heterogeneous Multi-core)는 태스크 부하에 따라 고성능 클러스터와 저성능 클러스터의 스위칭 방식으로 운용될 수 있다. 더불어, 이종 멀티-코어(Heterogeneous Multi-core)는 서로 다른 클러스터의 코어들이 동시에 복수의 쓰레드를 처리하도록 제어될 수도 있다. 하지만, 특정 동작에서 서로 다른 성능의 코어들이 동시에 구동되는 방식은 클러스터 스위칭 방식에 비해서 상대적으로 성능 열화가 발생할 수도 있다.

발명의 내용

해결하려는 과제

[0005] 본 발명의 목적은 멀티-코어 프로세서를 사용하는 시스템 온 칩 또는 응용 프로세서에서 이종의 코어들이 사용될 때 발생하는 비효율성을 제거할 수 있는 방법을 제공하는 데 있다.

과제의 해결 수단

[0006] 상기 목적을 달성하기 위한 본 발명의 실시 예에 따른 멀티-코어 프로세서를 포함하는 시스템 온 칩의 스케줄링 방법은, 상기 멀티-코어 프로세서에서 실행될 쓰레드의 스케줄링 요청을 검출하는 단계, 상기 멀티-코어 프로세서에서 실행 중인 쓰레드들 중에서 상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 콜링 쓰레드를 검출하는 단계, 그리고 상기 동일한 컨텍스트를 갖는 콜링 쓰레드가 실행되는 코어의 성능에 따라 상기 스케줄링 요청된 쓰레드를 실행할 코어를 재할당 또는 재설정하는 단계를 포함한다.

[0007] 상기 목적을 달성하기 위한 본 발명의 실시 예에 따른 시스템 온 칩은, 서로 다른 구동 속도를 갖는 복수의 코어들을 포함하는 멀티-코어 프로세서, 그리고 상기 멀티-코어 프로세서에 의해서 구동되는 운영 체제 및 응용 프로그램이 로드되는 워킹 메모리를 포함하되, 상기 운영 체제는 상기 응용 프로그램에서 발생하는 쓰레드의 스케줄링 요청에 따라 상기 쓰레드가 실행될 코어를 할당하는 스케줄러를 포함하며, 상기 스케줄러는 상기 멀티-코어 프로세서에서 이미 실행중인 복수의 쓰레드들 중에서 상기 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 쓰레드의 존재 여부에 따라 상기 스케줄링 요청된 쓰레드가 실행될 코어를 선택 또는 설정한다.

[0008] 상기 목적을 달성하기 위한 본 발명의 실시 예에 따른 멀티-코어 프로세서의 쓰레드 스케줄링 방법은, 상기 멀티-코어 프로세서에서 실행 중인 쓰레드들 중에서 동일한 컨텍스트를 공유하는 쓰레드가 존재하는지 검출하는 단계, 동일한 컨텍스트를 공유하는 쓰레드들이 각각 서로 다른 코어들에 분산되어 실행되는지 검출하는 단계, 그리고 상기 동일한 컨텍스트를 공유하는 쓰레드들 중 저속 코어에서 실행되는 쓰레드를 상기 서로 다른 코어들 중에서 고속 코어로 마이그레이션하는 단계를 포함한다.

발명의 효과

[0009] 본 발명에 따른 모바일 장치 및 그것의 제어 방법에 따르면, 멀티-코어 프로세서에서 이종의 코어들이 활성화되는 경우에 발생하는 성능 저하 및 전력 소모를 최소화할 수 있다.

도면의 간단한 설명

[0010] 도 1은 본 발명의 실시 예에 따른 모바일 장치를 보여주는 블록도이다.

도 2는 도 1의 멀티-코어 프로세서(110)의 구조를 보여주는 블록도이다.

도 3은 도 1의 모바일 장치의 소프트웨어 구조를 보여주는 블록도이다.

도 4는 본 발명의 실시 예에 따른 스케줄러의 동작을 보여주는 블록도이다.

도 5는 본 발명의 실시 예에 따른 스케줄링 방법을 보여주는 순서도이다.

도 6 내지 도 8은 본 발명의 스케줄러에 의한 쓰레드의 코어 선택 또는 설정 방법을 간략히 보여주는 타이밍도들이다.

도 9는 본 발명의 다른 실시 예에 따른 멀티-코어 프로세서의 스케줄링 방법을 보여주는 순서도이다.

도 10은 본 발명의 실시 예에 따른 휴대용 단말기를 도시한 블록도이다.

발명을 실시하기 위한 구체적인 내용

[0011] 이하, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자가 본 발명의 기술적 사상을 용이하게 실시할 수 있을 정도로 상세히 설명하기 위하여, 본 발명의 실시 예를 첨부된 도면을 참조하여 설명하기로 한다. 동일한 구성 요소들은 동일한 참조번호를 이용하여 인용될 것이다. 유사한 구성 요소들은 유사한 참조번호들을 이용하여 인용될 것이다. 아래에서 설명될 본 발명에 따른 모바일 장치 또는 프로세서에 의해 수행되는 동작은 예를 들어 설명한 것에 불과하며, 본 발명의 기술적 사상을 벗어나지 않는 범위 내에서 다양한 변화 및 변경이 가능하다.

[0012] 본 발명은 다른 실시 예들을 통해 구현되거나 적용될 수 있을 것이다. 게다가, 상세한 설명은 본 발명의 범위,

기술적 사상 그리고 다른 목적으로부터 상당히 벗어나지 않고 관점 및 응용에 따라 수정되거나 변경될 수 있다. 이하, 본 발명에 따른 실시 예를 첨부된 도면을 참조하여 상세히 설명한다.

[0013] 도 1은 본 발명의 실시 예에 따른 모바일 장치를 보여주는 블록도이다. 도 1을 참조하면, 모바일 장치(100)는 시스템 온 칩(SoC)과 워킹 메모리(130), 액정 표시 장치(152), 터치 패널(154), 저장 장치(170) 등을 포함할 수 있다. 시스템 온 칩(SoC)은 멀티-코어 프로세서(110), 디램 컨트롤러(120), 성능 컨트롤러(140), 사용자 인터페이스 컨트롤러(150), 메모리 인터페이스(160), 가속기(180) 등을 포함할 수 있다. 모바일 장치(100)의 구성 요소들은 도시된 구성 요소들에만 국한되지 않음은 잘 이해될 것이다. 예를 들면, 모바일 장치(100)는 영상 데이터를 처리하기 위한 하드웨어 코덱, 보안 블록 등을 더 포함할 수 있다.

[0014] 멀티-코어 프로세서(110)는 모바일 장치(100)에서 수행될 소프트웨어(응용 프로그램, 운영 체제, 장치 드라이버들)를 실행한다. 멀티-코어 프로세서(110)는 워킹 메모리(130)에 로드되는 운영 체제(OS)를 실행할 것이다. 멀티-코어 프로세서(110)는 운영 체제(OS) 기반에서 구동될 다양한 응용 프로그램들(Application Program)을 실행할 것이다. 멀티-코어 프로세서(110)는 동종 멀티-코어 프로세서(Homogeneous Multi-Core Processor) 또는 이종 멀티-코어 프로세서(Heterogeneous Multi-Core Processor)로 제공될 수 있다.

[0015] 본 발명의 멀티-코어 프로세서(110)의 멀티-코어들 각각은 독립된 구동 클록 및 독립된 구동 전압에 의해서 동작하는 복수의 파워 도메인(Power Domain)으로 구성될 수 있다. 그리고 멀티-코어들 각각에 공급되는 구동 전압과 구동 클록을 코어 단위로 차단 또는 연결할 수 있다. 멀티-코어들 각각은 구동 클록의 주파수 및 구동 전압의 레벨은 코어들 각각의 처리 로드에 따라 가변될 수 있다. 즉, 각각의 코어는 부하의 크기에 따라 구동 클록의 주파수나 구동 전압의 레벨을 상승시키는 동적 전압 주파수 스케일링(Dynamic Voltage Frequency Scaling: 이하, DVFS) 방식으로 제어될 수 있다.

[0016] 특히, 본 발명의 멀티-코어 프로세서(110)에 어느 하나의 쓰레드를 스케줄링할 경우, 고성능의 코어에서 실행되는 쓰레드와 동일한 컨텍스트를 가졌는지 검출된다. 검출 결과, 고성능의 코어에서 실행되는 쓰레드와 컨텍스트를 공유하는 것으로 검출되는 쓰레드는 고성능 코어에서 실행되도록 쓰레드 마이그레이션(Thread migration)이 수행될 수 있다. 이러한 스케줄링은 운영 체제(OS)의 커널(Kernel)에서 수행될 것이다.

[0017] 디램 컨트롤러(120)는 워킹 메모리(130)와 시스템 온 칩(SoC) 사이에서 인터페이싱을 제공한다. 디램 컨트롤러(120)는 멀티-코어 프로세서(110)나 다른 기능 블록(Intellectual Property: IP)의 요청에 따라 워킹 메모리(130)를 액세스할 것이다. 예를 들면, 디램 컨트롤러(120)는 멀티-코어 프로세서(110)의 쓰기 요청에 따라 데이터를 워킹 메모리(130)에 기입할 수 있다. 또는, 디램 컨트롤러(120)는 멀티-코어 프로세서(110)의 읽기 요청에 따라 워킹 메모리(130)로부터 데이터를 읽어내어 데이터 버스(Data bus)를 통해서 멀티-코어 프로세서(110)나 메모리 인터페이스(160)에 전달할 수 있다.

[0018] 워킹 메모리(130)에는 부팅시에 운영 체제(OS)나 응용 프로그램들(Application Program)이 로드될 것이다. 예를 들면, 모바일 장치(100)의 부팅시에 저장 장치(170)에 저장된 OS 이미지가 부팅 시퀀스에 의거하여 워킹 메모리(130)로 로드된다. 운영 체제(OS)에 의해서 모바일 장치(100)의 제반 입출력 동작들이 지원될 수 있다. 마찬가지로, 사용자의 의하여 선택되거나 기본적인 서비스 제공을 위해서 응용 프로그램들이 워킹 메모리(130)에 로드될 수 있다. 워킹 메모리(130)는 이뿐 아니라, 카메라와 같은 이미지 센서로부터 제공되는 영상 데이터를 저장하는 버퍼 메모리로 사용될 수도 있을 것이다. 워킹 메모리(130)는 SRAM(Static Random Access Memory)이나 DRAM(Dynamic Random Access Memory)과 같은 휘발성 메모리이거나, PRAM, MRAM, ReRAM, FRAM, NOR 플래시 메모리 등의 비휘발성 메모리일 수 있다.

[0019] 성능 컨트롤러(140)는 운영 체제(OS)의 커널(Kernel)로부터 제공되는 제어 요청에 따라 시스템 온 칩(SoC)의 동작 파라미터들을 조정할 수 있다. 예를 들면, 성능 컨트롤러(140)는 시스템 온 칩(SoC)의 성능을 높이기 위해서 DVFS(Dynamic Voltage Frequency Scaling)의 레벨을 조정할 수 있다. 또는, 성능 컨트롤러(140)는 커널(Kernel)의 요청에 따라 멀티-코어 프로세서(110)의 빅앤리틀(Big.LITTLE)과 같은 구동 모드를 제어할 수 있다. 이때, 성능 컨트롤러(140)는 내부에 구동 전압 및 구동 클록의 주파수를 설정하는 성능 테이블(142)을 포함할 수 있다.

[0020] 사용자 인터페이스 컨트롤러(150)는 사용자 인터페이스 장치들로부터의 사용자 입력 및 출력을 제어한다. 예를 들면, 사용자 인터페이스 컨트롤러(150)는 멀티-코어 프로세서(110)의 제어에 따라 액정 표시 장치(152)에 데이터를 입력하기 위한 키보드 화면 등을 표시할 수 있다. 또는, 사용자 인터페이스 컨트롤러(150)는 사용자가 요청한 데이터를 표시하도록 액정 표시 장치(152)를 제어할 수 있다. 사용자 인터페이스 컨트롤러(150)는 터치 패

널(154)과 같은 사용자 입력 수단으로부터의 제공되는 데이터를 사용자 입력 데이터로 디코딩할 수 있다.

[0021] 메모리 인터페이스(160)는 멀티-코어 프로세서(110)의 요청에 따라 저장 장치(170)를 액세스한다. 즉, 메모리 인터페이스(160)는 시스템 온 칩(SoC)과 저장 장치(170) 사이의 인터페이스를 제공한다. 예를 들면, 멀티-코어 프로세서(110)에 의해서 처리된 데이터가 메모리 인터페이스(160)를 통해 저장 장치(170)에 저장된다. 다른 예로써, 저장 장치(170)에 저장된 데이터는 메모리 인터페이스(160)를 통해 멀티-코어 프로세서(110)에 제공될 수 있다.

[0022] 저장 장치(170)는 모바일 장치(100)의 저장 매체(Storage Medium)로서 제공된다. 저장 장치(170)는 응용 프로그램들(Application Program), 운영 체제 이미지(OS Image) 및 각종 데이터를 저장할 수 있다. 저장 장치(170)는 메모리 카드(MMC, eMMC, SD, MicroSD 등)로 제공될 수도 있다. 저장 장치(170)는 대용량의 저장 능력을 가지는 낸드 플래시 메모리(NAND-type Flash memory)를 포함할 수 있다. 또는, 저장 장치(170)는 PRAM, MRAM, ReRAM, FRAM 등의 차세대 불휘발성 메모리나 NOR 플래시 메모리를 포함할 수도 있다. 본 발명의 다른 실시 예에서는 저장 장치(170)는 시스템 온 칩(SoC)의 내부에 구비되는 내장 메모리일 수 있다.

[0023] 가속기(180)는 멀티미디어 또는 멀티미디어 데이터의 처리 속도를 향상하기 위한 별도의 기능 블록(IP)으로 제공될 수 있다. 예를 들면, 가속기(180)는 텍스트(Text), 오디오(Audio), 정지 영상들(Still images), 애니메이션(Animation), 비디오(Video), 2차원 데이터, 또는 3차원 데이터의 처리 성능을 향상시키기 위한 기능 블록(IP)으로 제공될 수 있다.

[0024] 시스템 인터커넥터(190)는 시스템 온 칩(SoC)의 내부에서 온칩 네트워크를 제공하기 위한 시스템 버스(System Bus)이다. 시스템 인터커넥터(190)는 예를 들면, 데이터 버스(Data bus), 어드레스 버스(Address bus) 및 컨트롤 버스(Control bus)를 포함할 것이다. 데이터 버스(Data bus)는 데이터가 이동하는 경로이다. 주로, 워킹 메모리(130)이나 저장 장치(170)로의 메모리 접근 경로를 제공될 것이다. 어드레스 버스(Address bus)는 기능 블록들(IPs) 간의 어드레스 교환 경로를 제공한다. 컨트롤 버스(Control bus)는 기능 블록들(IPs) 간의 제어 신호를 전달하는 경로를 제공한다. 하지만, 시스템 인터커넥터(190)의 구성은 상술한 설명에만 국한되지 않으며, 효율적인 관리를 위한 중재 수단들을 더 포함할 수 있다.

[0025] 이상의 설명에 따르면, 모바일 장치(100)는 멀티-코어 프로세서(110)에서 서로 다른 코어에서 수행되는 컨텍스트를 공유하는 쓰레드를 검출하고, 검출 결과에 따라 쓰레드의 할당을 변경시킬 수 있다. 따라서, 상호 의존적인 쓰레드들이 서로 다른 코어에서 실행됨에 따라 발생하는 처리 지연이나 전력 소모를 줄일 수 있다.

[0026] 도 2는 도 1의 멀티-코어 프로세서(110)의 구조를 예시적으로 보여주는 도면이다. 도 2를 참조하면, 이종(Heterogeneous)의 코어들(Cores)을 포함하는 멀티-코어 프로세서(110)가 예시적으로 도시되어 있다. 멀티-코어 프로세서(110)는 제 1 클러스터(112), 제 2 클러스터(114) 그리고 캐시 코히런트 인터커넥터(116)를 포함할 수 있다.

[0027] 제 1 클러스터(112)는 상대적으로 높은 처리 속도를 제공하지만, 전력 소모가 큰 코어들(BCore_1, BCore_2, BCore_3, BCore_4)을 포함한다. 코어들(BCore_1, BCore_2, BCore_3, BCore_4) 각각은 독립적인 연산이 가능하며, 각각 독립적으로 전원 제어가 가능하다. 즉, 코어(BCore_1)는 전원 전압(VDD11)과 클록(CLK11)을 제공받을 수 있다. 코어(BCore_2)는 전원 전압(VDD12)과 클록(CLK12)을 제공받을 수 있다. 코어(BCore_3)는 전원 전압(VDD13)과 클록(CLK13)을 제공받을 수 있다. 코어(BCore_4)는 전원 전압(VDD14)과 클록(CLK14)을 제공받을 수 있다.

[0028] 여기서, 전원 전압들(VDD11, VDD12, VDD13, VDD14) 각각은 동일한 레벨로 또는 서로 다른 레벨로 제공될 수 있다. 즉, 어느 하나의 코어는 전원을 제공받아 구동되는 동안에 다른 하나의 코어는 대기 상태나 유휴 상태(Idle), 또는 플러그 아웃 상태로 관리될 수 있다. 클록들(CLK11, CLK12, CLK13, CLK14) 각각도 서로 다른 주파수 또는 동일한 주파수로 관리될 수 있다. 이러한 전원 전압과 클록의 주파수 관리는 DVFS 정책에 따라서 제어될 것이다.

[0029] 제 2 클러스터(114)는 상대적으로 처리 속도는 낮지만, 높은 전력 효율을 갖는 코어들(LCore_1, LCore_2, LCore_3, LCore_4)을 포함한다. 코어들(LCore_1, LCore_2, LCore_3, LCore_4) 각각은 독립적인 연산 처리가 가능하다. 그리고 코어들(LCore_1, LCore_2, LCore_3, LCore_4) 각각은 코어들(BCore_1, BCore_2, BCore_3, BCore_4)에 비해서 상대적으로 저성능의 프로세서로 구성될 수 있다.

[0030] 도시된 바와 같이 코어들(LCore_1, LCore_2, LCore_3, LCore_4) 각각은 전원 전압들(VDD21, VDD22, VDD23, VDD24)을 제공받을 수 있다. 전원 전압들(VDD21, VDD22, VDD23, VDD24) 각각은 서로 다른 레벨로 제공될 수 있

고, 스케줄링에 의해서 특정 코어의 전원 전압은 차단될 수도 있을 것이다. 코어들(LCore_1, LCore_2, LCore_3, LCore_4) 각각은 클록들(CLK21, CLK22, CLK23, CLK24)을 제공받을 수 있다. 클록들(CLK21, CLK22, CLK23, CLK24) 각각의 주파수는 대응하는 코어의 전원 전압과 연동하여 변화될 수도 있을 것이다. 즉, 클록들(CLK21, CLK22, CLK23, CLK24)의 주파수는 DVFS 방식에 따라 제어될 수 있을 것이다.

[0031] 여기서, 제 1 클러스터(112)의 적어도 하나의 코어와 제 2 클러스터(114)의 적어도 하나의 코어들은 서로 혼용될 수 있다. 즉, 높은 성능과 전력 효율을 위해서 제 1 클러스터(112)와 제 2 클러스터(114)의 어느 하나의 클러스터만이 선택되는 클러스터 스위칭(Cluster switching)뿐 아니라, 제 1 클러스터(112)의 코어 중 적어도 하나와 제 2 클러스터(114)의 코어들 중 적어도 하나를 동시에 구동할 수 있다. 이러한 방식을 이종 멀티 프로세싱(Heterogeneous Multi-Processing: 이하, HMP라 칭함) 방식이라 한다.

[0032] 더불어, 제 1 클러스터(112)와 제 2 클러스터(114)에서 활성화된 코어들은 적어도 하나의 쓰레드를 실행할 수 있다. 그리고 코어들(BCore_1, BCore_2, BCore_3, BCore_4)은 동일한 컨텍스트(Context)를 공유하는 쓰레드들을 서로 다른 코어에서 또는 동일한 코어에서 실행할 수 있는 멀티-쓰레드(Multi-thread)를 지원할 수 있다. 멀티-쓰레드는 제 1 클러스터(112)와 제 2 클러스터(114)에 관계없이 적용될 수 있다. 즉, 제 1 클러스터(112)의 어느 하나의 코어가 실행하는 쓰레드와 제 2 클러스터(114)의 다른 하나의 코어가 실행하는 쓰레드가 동일한 컨텍스트(Context)를 가질 수 있다.

[0033] 캐시 코히런트 인터커넥터(116, CCI)는 클러스터들(112, 114) 간의 태스크 마이그레이션(Task Migration)을 지원한다. 즉, 제 1 클러스터(112)가 비활성화되고, 제 2 클러스터(114)가 활성화되는 경우, 제 1 클러스터(112)의 캐시 데이터들이 제 2 클러스터(114)의 캐시로 이동해야 한다. 더불어, 컨텍스트를 공유하는 쓰레드의 마이그레이션도 동일한 절차에 의해서 수행될 수 있을 것이다. 이러한 클러스터들(112, 114) 간의 데이터 일관성을 제공하기 위한 구조로 캐시 코히런트 인터커넥터(116)가 제공될 것이다.

[0034] 쓰레드(Thread)는 일반적으로 운영 체제(OS)가 프로세서 시간을 할당하는 기본적인 단위로 간주할 수 있다. 즉, 쓰레드(Thread)는 하나의 프로세스에 포함되는 독립적 처리 단위로 스케줄링의 최소 단위에 해당한다. 일반적으로 하나의 프로세스(Process)는 복수의 쓰레드를 포함할 수 있다. 이러한 쓰레드는 예외 핸들러와 스케줄 우선순위, 시스템 스케줄에 따른 쓰레드 컨텍스트를 저장하기 위한 구조체 집합을 의미할 수 있다. 쓰레드는 레지스터(Register Set), 커널 스택(Kernel Stack), 사용자 스택 등의 여러 가지 실행 환경 정보를 보관하고 있다. 그리고 이러한 정보들을 컨텍스트(Context)라고 한다. 컨텍스트(Context)는 쓰레드가 작업을 진행하는 동안 작업에 대한 정보를 보관하고 있으며, 다른 쓰레드로 작업 순서가 넘어갈 때 저장된다.

[0035] 본 발명의 스케줄링 방식에 따르면, 이종(Heterogeneous) 멀티-코어(Multi-Core) 프로세서(110)에서 이종 멀티-프로세싱(HMP) 동작시에 발생하는 비효율성을 차단할 수 있다. 즉, 고성능의 코어에 할당된 쓰레드와 저성능 코어에 할당된 쓰레드가 동일한 컨텍스트(Context)를 공유하는 경우에 이러한 비효율성이 발생할 수 있다. 즉, 고성능의 코어에서 실행되는 쓰레드가 저성능 코어에서 실행되는 쓰레드의 실행 결과를 대기하는 경우에 고성능 코어는 실질적으로는 대기 상태로 존재하면서도 전력 소모는 일정하게 유지하게 된다. 따라서, 컨텍스트를 공유하면서도 서로 다른 성능의 코어들에 쓰레드가 분산되는 경우에는 상술한 비효율성이 발생하게 될 것이다.

[0036] 본 발명의 멀티-코어 프로세서(110)에 대한 스케줄링 방법에 따르면, 컨텍스트를 공유하는 쓰레드들이 서로 다른 성능의 코어들로 분산되는 상황을 차단할 수 있다. 따라서, 이종 멀티-프로세싱(HMP) 동작시 발생하는 전력 소모와 동작의 지연을 최소화할 수 있다.

[0037] 도 3은 도 1의 모바일 장치의 소프트웨어 구조를 보여주는 블록도이다. 도 3을 참조하면, 워킹 메모리(130)에 로드되고 멀티-코어 프로세서(110)에 의해서 구동되는 모바일 장치(100)의 소프트웨어 계층 구조는 간략히 응용 프로그램(132)과 커널(Kernel, 134)로 구분될 수 있다. 물론, 운영 체제(OS)에는 메모리나 모뎀, 이미지 처리 장치와 같은 다양한 장치들을 관리하는 장치 드라이버(Device driver)가 더 포함될 수 있을 것이다.

[0038] 응용 프로그램(132)은 기본적인 서비스로서 구동되거나, 사용자의 요청에 의해서 구동되는 상위 계층의 소프트웨어이다. 응용 프로그램(132)은 복수의 쓰레드들(TH1, TH2, TH3, TH4)을 생성할 수 있다. 응용 프로그램은 특정 작업을 위해서 프로세스(Process)를 생성할 수 있다. 프로세서는 하나 이상의 쓰레드로 구성된다. 예를 들면, 사용자에 의해서 동영상 파일의 재생이 요청되면, 동영상을 재생하기 위한 응용 프로그램(예를 들면, 비디오 플레이어)이 실행된다. 그러면, 실행된 플레이어는 사용자가 요청한 동영상 파일을 재생하기 위한 저장 장치(170)로의 읽기 요청(Read request) 또는 쓰기 요청(Write request)을 생성할 것이다. 이러한 요청들은 프로세서를 구성하고, 각각의 프로세서들은 쓰레드를 실행 단위로 수행될 것이다.

[0039] 커널(134)은 운영 체제(OS)의 구성으로, 응용 프로그램(132)과 하드웨어(Hardware) 사이에서 제어 동작을 수행한다. 커널(134)은 프로그램의 실행, 인터럽트, 멀티 태스킹, 메모리 관리, 파일 시스템, 그리고 장치 드라이버 등을 포함할 수 있다. 본 발명에서, 커널(134)의 일부로 제공되는 스케줄러(135) 및 CPU 거버너(137)에 대해서만 설명하기로 한다.

[0040] 스케줄러(135)는 응용 프로그램(132)에서 제공된 쓰레드를 멀티-코어 프로세서(110)의 각 코어들에 할당한다. 스케줄러(135)는 응용 프로그램(132)이나 운영체제에서 발생하는 스케줄링 이벤트를 모니터링할 것이다. 만일, 스케줄링 이벤트가 발생하면, 스케줄러(135)는 스케줄링할 쓰레드의 평균 부하에 따라 복수의 멀티 코어들 중 어느 하나에 할당할 것이다. 그리고 스케줄러(135)는 이미 실행중인 쓰레드들 중에서 스케줄링 요청된 쓰레드와 컨텍스트를 공유하는 것이 존재하는지 검색할 것이다. 만일, 동일한 컨텍스트를 가진 콜링 쓰레드(Calling Thread)가 고성능의 코어에서 실행중인 경우, 스케줄러(135)는 스케줄링 요청된 쓰레드를 콜링 쓰레드와 동일한 코어에서 실행되도록 코어 스위칭을 수행할 것이다. 이러한 일련의 과정을 통해서 본 발명의 스케줄러(135)는 컨텍스트가 동일한 쓰레드들을 동일한 코어 또는 동일한 성능으로 구동되는 코어로 할당할 수 있다. 따라서, 본 발명의 스케줄러(135)를 통해서 동일한 컨텍스트를 갖지만 서로 다른 성능의 코어들에 분산된 쓰레드들에 의해서 발생하는 전력 효율의 저하를 최소화할 수 있다. 본 발명의 스케줄러(135)에 대한 구체적인 구성에 대해서는 후술하는 도 4에서 상세히 설명될 것이다.

[0041] CPU 거버너(137)는 커널(134) 내에서 멀티-코어 프로세서(110)의 구동 전압 및 구동 클록을 구할 수 있다. 멀티-코어 프로세서(110)의 구동 전압 및 구동 클록에 따라 구동 클록(CLK)의 주파수나 구동 전압(VDD)의 레벨이 결정될 수 있다. 하지만, 멀티-코어 프로세서(110)의 구동 전압 및 구동 클록은 여기에 개시된 것들에만 국한되지 않음을 잘 이해될 것이다. CPU 거버너(137)는 커널(134)이나 스케줄러(135)의 요청에 따라 멀티-코어 프로세서(110)의 구동 전압과 구동 클록 주파수를 조정할 수 있다. 물론, 커널(134)에서 구동되는 CPU 거버너(137)의 제어에 따라 하드웨어적으로 멀티-코어 프로세서(110)의 구동 클록 주파수나 구동 전압의 레벨을 조정하는 구성이 존재할 수 있다. 예를 들면, 소프트웨어인 커널(134)의 호출에 따라 멀티-코어 프로세서(110)나 다양한 구성들의 구동 클록, 구동 전압을 가변하는 퍼포먼스 컨트롤러(140)가 이러한 멀티-코어 프로세서(110)의 구동 전압 및 구동 클록을 하드웨어적으로 제어할 수 있을 것이다.

[0042] 이상의 운영 체제(OS)의 커널(134) 계층에서 수행되는 본 발명의 멀티-코어 프로세서의 쓰레드 스케줄링 방법이 간략히 설명되었다. 본 발명의 실시 예에 따르면, 어느 하나의 쓰레드에 대한 스케줄링 이벤트가 발생하면, 동일한 컨텍스트의 쓰레드가 고성능 코어에서 실행되는지 검출된다. 만일, 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 쓰레드가 고성능 코어에서 구동중인 것으로 검출되면, 스케줄링 요청된 쓰레드는 고성능 코어로 재할당될 수 있다. 따라서, 동일한 컨텍스트를 갖는 쓰레드들이 서로 다른 성능의 코어로 분산되어 발생되는 비효율성을 방지할 수 있다.

[0043] 도 4는 본 발명의 실시 예에 따른 스케줄러의 동작을 보여주는 블록도이다. 도 4를 참조하면, 스케줄러(135)는 이벤트 발생 모듈(135a), 쓰레드 부하 계산 모듈(135b), 쓰레드 제어 모듈(135c)을 포함할 수 있다. 그리고 스케줄러(135)의 제어에 따라 CPU 거버너(137)는 선택된 쓰레드를 마이그레이션하고, 해당 코어에 대한 구동 전압이나 클록 주파수를 제어할 것이다.

[0044] 이벤트 발생 모듈(135a)은 응용 프로그램(132)이나 운영체제(OS)에서 발생하는 스케줄링 이벤트를 감지한다. 이벤트 발생 모듈(135a)은 스케줄링을 위한 요청에 응답하여 해당 쓰레드를 미리 정해진 스케줄링 정책에 따라 선택된 코어에 할당하게 될 것이다. 미리 정해진 스케줄링 정책에는 다양한 스케줄링 방식들이 적용될 수 있다. 예를 들면, 쓰레드의 종류나 포함되는 프로세서의 특성에 따라 스케줄링 요청된 쓰레드가 실행될 코어가 할당될 수 있다.

[0045] 쓰레드 부하 계산 모듈(135b)은 스케줄링 요청된 쓰레드의 평균 부하를 계산한다. 즉, 스케줄링 요청된 쓰레드에 의한 일차적으로 할당된 코어에서의 대기열 크기나 코어의 사용율을 참조하여 평균 부하의 크기를 계산할 수 있을 것이다. 만일, 스케줄링 요청된 쓰레드에 의해서 발생하는 부하의 크기가 기준치보다 큰 경우에는 해당 쓰레드를 고성능의 코어로 재할당해야 할 것이다. 반면, 계산된 평균 부하가 기준치와 같거나 작은 경우, 저성능의 코어가 스케줄링 요청된 쓰레드에 할당될 것이다.

[0046] 쓰레드 제어 블록(135c)은 본 발명의 실시 예에 따른 동일 컨텍스트를 공유하는 쓰레드를 검색하고, 검색 결과에 따라 스케줄링 요청된 쓰레드를 실행할 코어를 재할당한다. 쓰레드 제어 블록(135c)은 스케줄링 요청된 쓰레드와 컨텍스트를 공유하는 쓰레드를 검출하고, 검출 결과를 참조하여 스케줄링 요청된 쓰레드를 실행할 코어를 재할당하기 위해, 쓰레드 컨텍스트 분류 모듈(135c_1), 쓰레드 상태 검출 모듈(135c_2) 그리고 불균형 상태 체

크 모듈(135c_3)을 포함할 수 있다.

[0047] 쓰레드 컨텍스트 분류 모듈(135c_1)은 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 쓰레드를 검색한다. 즉, 쓰레드 컨텍스트 분류 모듈(135c_1)은 스케줄링 이벤트에 해당하는 쓰레드와 자원을 공유하는 쓰레드들이 존재하는지 검출하게 될 것이다. 이러한 검출을 위해서 쓰레드 컨텍스트 분류 모듈(135c_1)은 메모리 자원 내에서 쓰레드 레지스터, 커널 스택, 사용자 스택 등의 컨텍스트 정보를 검색할 것이다. 그리고 동일한 컨텍스트 정보를 갖는 쓰레드가 존재하는지 여부를 쓰레드 상태 검출 모듈(135c_2)에 알려줄 수 있다.

[0048] 쓰레드 상태 검출 모듈(135c_2)은 스케줄링 요청된 쓰레드와 컨텍스트를 공유하는 쓰레드의 상태를 검출한다. 쓰레드 상태 검출 모듈(135c_2)은 컨텍스트를 공유하는 쓰레드가 실행되는 코어의 종류, 코어의 전압이나 클록 주파수, 코어의 상태 등을 검출하게 될 것이다. 예를 들면, 쓰레드 상태 검출 모듈(135c_2)은 컨텍스트를 공유하는 코어가 고성능 모드로 구동되는지, 또는 저성능 모드로 구동중인지를 검출할 수 있다. 또는, 쓰레드 상태 검출 모듈(135c_2)은 이종 멀티 프로세싱 동작에서 컨텍스트를 공유하는 콜링 쓰레드(Calling Thread)가 빅 코어(big Core)에서 실행되고 있는지 또는 리틀 코어(LITTLE Core)에서 실행되고 있는지를 검출할 수 있다. 만일, 컨텍스트를 공유하는 콜링 쓰레드(Calling Thread)가 저성능의 리틀 코어(LITTLE Core)에서 실행 중인 경우에는, 스케줄링 요청된 쓰레드의 코어 재할당은 실행되지 않아도 된다. 하지만, 컨텍스트를 공유하는 콜링 쓰레드(Calling Thread)가 빅 코어(big Core) 또는 고성능 코어에서 실행중이고, 스케줄링 요청된 쓰레드가 저성능 또는 리틀 코어(LITTLE Core)에서 실행중이라면, 스케줄링 요청된 쓰레드의 코어 재할당이 발생할 것이다.

[0049] 불균형 상태 검출 모듈(135c_3)은 쓰레드 컨텍스트 분류 모듈(135c_1)과 쓰레드 상태 검출 모듈(135c_2)로부터 제공되는 정보를 참조하여 스케줄링 요청된 쓰레드가 전력 효율이나 성능 저하의 원인이 되는지 판단한다. 현재 스케줄링 요청된 쓰레드가 저성능 코어에서 실행되고, 컨텍스트를 공유하는 콜링 쓰레드가 고성능 코어에서 실행되는 것으로 검출된 것으로 가정하다. 그러면 불균형 상태 검출 모듈(135c_3)은 스케줄링 요청된 쓰레드를 고성능의 코어에서 실행되도록 스위칭을 수행할 수 있다. 반면, 스케줄링 요청된 쓰레드가 고성능 코어에서 실행되고 있는 것으로 검출되면, 불균형 상태 검출 모듈(135c_3)은 컨텍스트를 공유하는 콜링 쓰레드의 실행 위치에 관계없이 현재의 쓰레드 할당 상태를 유지하도록 CPU 거버너(137)를 제어할 수 있을 것이다.

[0050] CPU 거버너 블록(137)은 상향 마이그레이터(137a), 하향 마이그레이터(137b) 그리고 DVFS 매니저(137c) 등을 포함할 수 있다. CPU 거버너 블록(137)은 스케줄러(135)에서 결정된 방식에 따라 본 발명의 쓰레드 재할당을 수행할 수 있다. 상향 마이그레이터(137a)는 스케줄러(135)의 제어에 따라 스케줄링 요청된 쓰레드를 현재의 코어보다 고성능의 코어에서 실행되도록 쓰레드의 할당을 조정할 수 있다. 반면, 하향 마이그레이터(137b)는 스케줄러(135)의 제어에 따라 스케줄링 요청된 쓰레드를 현재의 코어보다 저성능의 코어에서 실행되도록 쓰레드의 할당을 조정할 수 있다. 더불어, DVFS 매니저(137c)는 스케줄러(135)의 제어에 따라 멀티-코어 프로세서(110)의 클러스터들 각각의 구동 전압이나 구동 클록의 주파수를 제어할 수 있다. 즉, 대기열에 누적된 태스크들의 수가 증가하는 경우, DVFS 매니저(137c)는 멀티-코어 프로세서(110)의 구동 전압과 구동 클록의 주파수를 유기적으로 상향시킬 수 있다. 하지만, DVFS 매니저(137c)는 반대의 경우에 대해서도 동일하게 적용될 수 있다.

[0051] 도 5는 본 발명의 일 실시 예에 따른 멀티-코어 프로세서의 스케줄링 방법을 간략히 보여주는 순서도이다. 도 5를 참조하면, 주기적으로 발생하는 쓰레드의 스케줄링 이벤트들 중 하나의 쓰레드에 대한 처리 과정이 간략히 설명되어 있다. 어느 하나의 쓰레드에 대한 스케줄링 이벤트가 발생하면, 컨텍스트를 공유하는 다른 쓰레드들에 대한 검색이 진행되고, 검색 결과에 따라 스케줄링 요청된 쓰레드가 실행될 코어가 결정될 수 있다.

[0052] S110 단계에서, 스케줄러(135, 도 3 참조)는 주기적으로 발생하는 쓰레드의 스케줄링 이벤트를 모니터링하고 검출하게 될 것이다. 여기서, 설명의 편의를 위해서 어느 하나의 쓰레드에 대한 스케줄링 절차를 설명하기로 한다.

[0053] S120 단계에서, 스케줄러(135)가 스케줄링 이벤트를 검출하면(Yes 방향) 절차는 S130 단계로 이동한다. 반면, 스케줄링 이벤트의 발생이 검출되지 않으면(No 방향), 절차는 S110 단계로 복귀하여 이벤트 발생을 지속적으로 모니터링할 것이다.

[0054] S130 단계에서, 스케줄러(135)는 스케줄링 요청된 쓰레드의 컨텍스트를 추출하여 동일한 컨텍스트를 갖는 쓰레드를 검색할 것이다. 스케줄러(135)는 쓰레드 풀(Thread pool)에 존재하는 복수의 쓰레드들에 대한 컨텍스트의 비교를 수행한다. 예를 들면, 스케줄러(135)는 쓰레드 레지스터, 커널 스택이나 사용자 스택 등의 쓰레드에 대한 실행 정보를 검색하여 스케줄링 요청된 쓰레드의 컨텍스트와 비교할 수 있다.

[0055] S140 단계에서, 스케줄러(135)는 쓰레드 풀(Thread pool)에 존재하는 쓰레드들의 컨텍스트 중에서 스케줄링 요

청된 쓰레드의 컨텍스트와 동일한 것이 존재하는지 판단할 것이다. 만일, 현재 실행중인 쓰레드들 중에서 스케줄링 요청된 쓰레드의 컨텍스트와 동일한 것이 존재하는 것으로 판단되면(Yes 방향), 절차는 S160 단계로 이동한다. 반면, 현재 실행중인 쓰레드들 중에서 스케줄링 요청된 쓰레드의 컨텍스트와 동일한 것이 존재하지 않는 것으로 판단되면(No 방향), 절차는 S150 단계로 이동한다.

[0056] S150 단계에서, 현재 실행중인 쓰레드들 중에서 스케줄링 요청된 쓰레드의 컨텍스트를 공유하는 것이 존재하지 않으므로, 계산된 평균 부하의 크기에 따라 코어를 할당할 것이다. 즉, 스케줄링 요청된 쓰레드의 특성을 고려하여 이미 결정된 스케줄링 정책에 따라 어느 하나의 코어를 할당하게 될 것이다.

[0057] S160 단계에서, 스케줄러(135)는 스케줄링 요청된 쓰레드를 실행하고 있는 코어가 컨텍스트를 공유하는 콜링 쓰레드를 실행하는 코어보다 저성능인지 판단하게 될 것이다. 예를 들면, 스케줄러(135)는 스케줄링 요청된 쓰레드와 동일한 컨텍스트를 갖는 콜링 쓰레드가 빅 코어(big Core)에서 구동되고, 스케줄링 요청된 쓰레드는 리틀 코어(LITTLE Core)에서 실행되고 있는지 결정할 수 있다. 만일, 스케줄링 요청된 쓰레드를 실행하고 있는 코어가 컨텍스트를 공유하는 콜링 쓰레드를 실행하는 코어보다 고성능에 해당하면(No 방향), 요청된 스케줄링을 위한 제반 절차는 종료될 것이다. 반면, 스케줄링 요청된 쓰레드를 실행하고 있는 코어가 컨텍스트를 공유하는 콜링 쓰레드를 실행하는 코어보다 저성능에 해당하면(Yes 방향), 절차는 S170 단계로 이동한다.

[0058] S170 단계에서, 스케줄러(135)는 스케줄링 요청된 쓰레드를 실행할 코어에 대한 재할당 또는 재설정을 수행할 수 있다. 예를 들면, 스케줄러(135)는 스케줄링 요청된 쓰레드를 저성능의 코어에서 고성능의 코어로 스위칭할 수 있을 것이다. 바람직하게는 콜링 쓰레드가 실행되는 고성능 코어에 스케줄링 요청된 쓰레드를 할당할 수도 있을 것이다. 또는, 스케줄링 요청된 쓰레드를 실행하고 있는 코어의 성능을 향상시키는 조작도 가능하다. 즉, 스케줄링 요청된 쓰레드를 현재 실행하고 있는 코어의 전압이나 클록 주파수를 콜링 쓰레드를 수행하는 코어와 동일하게 조정할 수도 있을 것이다. 하지만, 스케줄링 요청된 쓰레드에 대한 재설정 방법은 여기의 개시에만 국한되지 않음은 잘 이해될 것이다.

[0059] S180 단계에서, 스케줄링 요청된 쓰레드에 대한 코어 재할당 또는 재설정이 완료되면, 재할당 또는 재설정된 코어에서 스케줄링 요청된 쓰레드를 실행할 것이다. 만일, 스케줄링 요청된 쓰레드와 콜링 쓰레드가 동일한 고성능 코어에서 실행된다면, 어느 하나의 쓰레드에서 다른 하나의 쓰레드의 동작 완료 시점을 기다리는 상황은 발생하지 않을 것이다.

[0060] 이상에서 본 발명의 개략적인 스케줄링 방법이 설명되었다. 이종 멀티-쓰레드 방식의 코어를 제어하기 위해서, 스케줄링 이벤트가 발생하면 컨텍스트를 공유하는 쓰레드의 존재 유무가 검출될 것이다. 그리고 검출 결과에 따라 스케줄링 요청된 쓰레드를 고성능 코어로 재할당할지가 결정된다. 이러한 쓰레드의 관리 방법에 따라 컨텍스트를 공유하는 쓰레드들이 서로 다른 성능의 코어들에 분산되어 발생하는 비효율성은 차단될 수 있다.

[0061] 도 6 내지 도 8은 본 발명의 스케줄러에 의한 쓰레드의 코어 선택 또는 설정 방법을 간략히 보여주는 타이밍도들이다. 도 6 내지 도 7은 스케줄링 요청된 쓰레드에 대한 코어 재할당 방법을 도시하고 있고, 도 8은 스케줄링 요청된 쓰레드가 실행되는 코어의 재설정 방법을 도시하고 있다. 여기서, 멀티-코어 프로세서는 예시적으로 빅 앤리틀(big.LITTLE) 구조의 이종 멀티-코어 프로세서를 예시적으로 나타내었다. 하지만, 본 발명의 이점은 서로 다른 성능이나 구동 속도로 제어될 수 있는 다양한 멀티-코어 프로세서에 적용될 수 있음은 잘 이해될 것이다.

[0062] 도 6은 스케줄링 요청된 쓰레드의 상향 마이그레이션(Up Migration)을 예시적으로 보여준다. 도 6을 참조하면, 스케줄링 이벤트에 의해서 스케줄링 요청된 제 3 쓰레드(TH3)가 저성능의 코어에서 고성능의 코어로 상향 마이그레이션 절차에 따라서 재할당된다.

[0063] 제 1 쓰레드(TH1)는 고속의 빅 코어(BCore1)에서 실행되고, 제 2 쓰레드(TH2)는 고속으로 구동되는 빅 코어(BCore2)에서, 그리고 제 4 쓰레드(TH4)는 저속으로 구동되는 리틀 코어(LCore2)에서 각각 실행되고 있다고 가정하자. 그리고 스케줄링 요청된 제 3 쓰레드(TH3)는 제 1 쓰레드(TH1)와 컨텍스트를 공유하는 것으로 가정하기로 한다.

[0064] 시점 (t0)에서 스케줄링 이벤트가 발생하고, 스케줄링 정책에 따라 제 3 쓰레드(TH3)가 생성된다. 생성된 제 3 쓰레드(TH3)는 우선적으로 리틀 코어(LCore1)에 할당되어 실행될 것이다. 그러면, 스케줄러(135)의 쓰레드 부하 계산 모듈(135b)은 리틀 코어(LCore1)에서 실행되는 제 3 쓰레드(TH3)의 평균 부하를 계산할 것이다. 더불어, 스케줄러(135)는 제 3 쓰레드(TH3)와 동일한 컨텍스트를 갖는 쓰레드가 현재 실행중인 쓰레드들 중에서 존재하는지 검색할 것이다. 스케줄러(135)는 제 3 쓰레드(TH3)와 동일한 컨텍스트를 갖는 제 1 쓰레드(TH1)가 존재함을 검출하게 될 것이다. 스케줄러(135)는 제 1 쓰레드(TH1)가 실행되는 코어가 고속의 코어인지, 저속의 코어인

지 검출할 것이다. 그리고 스케줄러(135)는 제 1 쓰레드(TH1)와 제 3 쓰레드(TH3)가 서로 다른 코어에서 실행됨에 따라 발생하는 동작의 비효율성이 존재하는지 판단할 것이다. 판단 결과에 따라, 스케줄러(135)는 제 3 쓰레드(TH3)가 실행될 코어를 고속의 빅코어(BCore1)로 재할당하게 될 것이다.

[0065] 시점(t1)부터는 제 3 쓰레드(TH3)가 제 1 쓰레드(TH1)와 함께 빅 코어(BCore1)에서 실행될 것이다. 즉, 제 3 쓰레드(TH3)는 저속의 리틀 코어(LCore1)에서 고속의 빅 코어(BCore1)로 상향 마이그레이션 처리된다. 동일한 컨텍스트를 갖는 제 1 쓰레드(TH1)와 제 3 쓰레드(TH3)는 동일한 코어에서 실행되기 때문에, 제 3 쓰레드(TH3)의 응답 지연에 따른 빅 코어(BCore1)의 대기 상황은 방지될 수 있다. 더불어, 리틀 코어(LCore1)에서 실행되는 쓰레드가 존재하지 않는다면, 리틀 코어(LCore1)는 핫플러그 아웃으로 관리될 수 있을 것이다.

[0066] 이상의 도 6의 타이밍도에 따르면, 컨텍스트를 공유하는 쓰레드들이 고속의 코어와 저속의 코어에서 분산되어 처리되는 경우, 본 발명의 스케줄러(135)는 저속의 코어에서 실행되는 쓰레드를 고속의 코어에서 실행되도록 스케줄링한다. 특히, 동일한 코어에 컨텍스트를 공유하는 쓰레드들이 실행되도록 설정하면, 동작 속도의 향상 및 전력 효율이 극대화될 수 있다.

[0067] 도 7은 스케줄링 요청된 쓰레드가 콜링 쓰레드가 실행되는 코어가 아닌 다른 코어로의 상향 마이그레이션(Up Migration)을 예시적으로 보여준다. 도 7을 참조하면, 스케줄링 요청된 제 3 쓰레드(TH3)는 저성능의 코어(LCore1)에서 고성능의 코어(BCore2)로 상향 마이그레이션된다. 즉, 콜링 쓰레드에 대응하는 제 1 쓰레드(TH1)와는 다른 코어로 제 3 쓰레드(TH3)가 이동한다.

[0068] 제 1 쓰레드(TH1)는 고속의 빅 코어(BCore1)에서 실행되고, 제 2 쓰레드(TH2)는 고속으로 구동되는 빅 코어(BCore2)에서, 그리고 제 4 쓰레드(TH4)는 저속으로 구동되는 리틀 코어(LCore2)에서 각각 실행되고 있다고 가정하자. 그리고 스케줄링 요청된 제 3 쓰레드(TH3)는 제 1 쓰레드(TH1)와 컨텍스트를 공유하는 것으로 가정하기로 한다.

[0069] 시점(t0)에서 스케줄링 이벤트가 발생하고, 스케줄링 정책에 따라 제 3 쓰레드(TH3)가 생성된다. 생성된 제 3 쓰레드(TH3)는 우선적으로 리틀 코어(LCore1)에 할당되어 실행될 것이다. 그러면, 스케줄러(135)는 제 3 쓰레드(TH3)와 동일한 컨텍스트를 갖는 쓰레드가 현재의 쓰레드 풀에 존재하는지 검색할 것이다. 스케줄러(135)는 제 3 쓰레드(TH3)와 동일한 컨텍스트를 갖는 제 1 쓰레드(TH1)가 존재함을 검출하게 될 것이다. 스케줄러(135)는 제 1 쓰레드(TH1)가 실행되는 코어가 고속의 코어인지, 저속의 코어인지 검출할 것이다. 그리고 스케줄러(135)는 제 1 쓰레드(TH1)와 제 3 쓰레드(TH3)가 서로 다른 코어에서 실행됨에 따라 발생하는 동작의 비효율성이 존재하는지 판단할 것이다. 판단 결과에 따라, 스케줄러(135)는 제 3 쓰레드(TH3)를 실행할 코어로 고속의 빅 코어(BCore2)로 재할당하게 될 것이다.

[0070] 시점(t1)부터, 제 3 쓰레드(TH3)는 빅 코어(BCore2)에서 실행될 것이다. 즉, 제 3 쓰레드(TH3)는 저속의 리틀 코어(LCore1)에서 고속의 빅 코어(BCore2)로 상향 마이그레이션된다. 하지만, 제 3 쓰레드와 제 1 쓰레드(TH1)는 처리 속도는 동일하지만, 서로 다른 빅 코어들(BCore1, BCore2)에서 각각 실행될 것이다.

[0071] 동일한 컨텍스트를 갖는 제 1 쓰레드(TH1)와 제 3 쓰레드(TH3)는 동일한 성능 또는 구동 속도를 갖는 코어들(BCore1, BCore2)에서 실행되기 때문에, 제 3 쓰레드(TH3)의 응답 지연에 따른 처리 지연은 최소화될 수 있다. 더불어, 리틀 코어(LCore1)에서 실행되는 쓰레드가 더 이상 남아있지 존재하지 않는다면, 리틀 코어(LCore1)는 핫플러그 아웃으로 관리될 수 있을 것이다.

[0072] 컨텍스트를 공유하는 쓰레드가 콜링 쓰레드와 다른 코어에 상향 마이그레이션되는 예가 설명되었다. 이러한 경우는 예를 들면, 콜링 쓰레드(TH1)가 실행되는 고속의 코어에 이미 상대적으로 많은 수의 멀티-쓰레드가 실행되고 있는 경우일 수 있다. 즉, 콜링 쓰레드(TH1)와 동일한 코어에 스케줄링 요청된 쓰레드(TH3)를 마이그레이션하기 어려운 상태일 때, 스케줄러(135)는 차선책으로 동일한 속도의 다른 코어를 선택할 수 있을 것이다. 동일한 속도의 다른 코어에 스케줄링 요청된 쓰레드(TH3)가 이동하면, 제 3 쓰레드(TH3)의 처리 지연에 따른 문제는 해결될 수 있다.

[0073] 이상에서는 컨텍스트를 공유하는 쓰레드들이 고속의 코어와 저속의 코어에서 분산되어 처리되는 경우, 저속의 코어에서 실행되는 쓰레드가 콜링 쓰레드가 실행되는 코어가 아닌 다른 코어로 상향 마이그레이션될 수 있음이 설명되었다. 특히, 동일한 성능의 코어에서 컨텍스트를 공유하는 쓰레드들이 실행되도록 설정하면, 어느 하나의 쓰레드의 응답 지연에 따른 성능 저하나 전력 비효율성은 해결될 수 있을 것이다.

[0074] 도 8은 스케줄링 요청된 쓰레드의 마이그레이션 없이 응답 속도를 높이기 위한 예시적인 방법을 보여주는 타이밍도이다. 도 8을 참조하면, 스케줄링 요청된 제 2 쓰레드(TH2)는 이미 빅 코어(BCore2)에 할당되고, 컨텍스트

를 공유하는 제 1 쓰레드(TH1)는 빅 코어(BCore1)에서 실행되고 있다. 하지만, 제 2 쓰레드(TH2)가 실행되는 빅 코어(BCore2)의 동작 모드가 저속 모드로 구동되고 있다고 가정한다.

[0075] 시점(t0)에서 스케줄링 이벤트가 발생하고, 스케줄링 정책에 따라 제 2 쓰레드(TH2)가 생성된다. 생성된 제 2 쓰레드(TH2)는 우선적으로 빅 코어(BCore2)에 할당되어 실행될 것이다. 그러면, 스케줄러(135)는 제 2 쓰레드(TH2)와 동일한 컨텍스트를 갖는 쓰레드가 현재의 쓰레드 풀에 존재하는지 검색할 것이다. 스케줄러(135)는 제 2 쓰레드(TH2)와 동일한 컨텍스트를 갖는 제 1 쓰레드(TH1)가 존재함을 검출하게 될 것이다. 스케줄러(135)는 제 1 쓰레드(TH1)가 실행되는 코어가 고속의 코어인지, 저속의 코어인지 검출할 것이다. 그리고 스케줄러(135)는 제 1 쓰레드(TH1)와 제 2 쓰레드(TH2)가 서로 다른 코어에서 실행됨에 따라 발생하는 동작의 비효율성이 존재하는지 판단할 것이다. 판단 결과에 따라, 스케줄러(135)는 제 2 쓰레드(TH2)를 실행할 다른 코어를 선택할지, 현재의 코어를 유지할지 결정할 것이다. 제 2 쓰레드(TH2)가 빅 코어(BCore2)에서 실행되고 있으므로 스케줄러(135)는 동작의 비효율성을 개선하기 위해서 저속 모드로 구동되는 빅 코어(BCore2)의 모드를 고속 모드로 스위칭하도록 결정할 것이다.

[0076] 시점(t1)부터, 제 2 쓰레드(TH2)는 마이그레이션 없이 빅 코어(BCore2)에서 실행될 것이다. 다만, 스케줄러(135)의 제어에 따라 CPU 거버너(137)는 빅 코어(BCore2)의 클록 주파수 및 전원 전압의 레벨을 상향시킬 것이다. 즉, 스케줄러(135)는 DVFS 제어를 통해서 빅 코어(BCore2)의 처리 속도를 빅 코어(BCore1)와 동일하게 전환하게 될 것이다.

[0077] 동일한 컨텍스트를 갖는 제 1 쓰레드(TH1)와 제 2 쓰레드(TH2)가 동일한 성능 또는 처리 속도를 갖는 코어들(BCore1, BCore2)에서 각각 실행되기 때문에, 제 2 쓰레드(TH2)의 응답 지연에 따른 처리 지연은 최소화될 수 있다.

[0078] 도 9는 본 발명의 다른 실시 예에 따른 멀티-코어 프로세서의 스케줄링 방법을 간략히 보여주는 순서도이다. 도 9를 참조하면, 하나의 쓰레드에 대한 스케줄링 이벤트를 처리하는 절차를 통해서 본 발명의 이점이 설명될 것이다. 하나의 쓰레드에 대한 스케줄링 이벤트가 발생하면, 컨텍스트를 공유하는 쓰레드들의 존재 여부가 검출되고, 검출 결과에 따라 스케줄링 요청된 쓰레드의 마이그레이션 여부가 결정될 수 있다. 좀더 자세히 설명하면 다음과 같다.

[0079] S205 단계에서, 스케줄러(135, 도 3 참조)는 주기적으로 발생하는 쓰레드의 스케줄링 이벤트를 모니터링하고 검출하게 될 것이다.

[0080] S210 단계에서, 스케줄러(135)가 스케줄링 이벤트가 발생한 것을 검출하면(Yes 방향) 절차는 S215 단계로 이동한다. 반면, 스케줄링 이벤트의 발생이 검출되지 않으면(No 방향), 절차는 S205 단계로 복귀하여 이벤트 발생을 지속적으로 모니터링할 것이다.

[0081] S215 단계에서, 스케줄러(135)는 스케줄링 이벤트가 발생한 쓰레드들 중에서 어느 하나를 선택할 것이다. 그리고 선택된 쓰레드는 특정 스케줄링 정책에 따라 또는 미리 결정된 우선 순위에 따라 복수의 코어들 중 어느 하나에 할당될 것이다.

[0082] S220 단계에서, 스케줄러(135)는 스케줄링 요청된 어느 하나의 쓰레드에 대해서 평균 부하를 계산할 것이다. 평균 부하를 계산하는 방식에는 다양한 방법이 사용될 수 있으며, 이러한 평균 부하의 계산 예는 도 5에서 예시적으로 설명되었으므로 이하에서는 생략하기로 한다.

[0083] S225 단계에서, 스케줄러(135)는 현재 스케줄링 요청된 쓰레드가 저속 코어 또는 리틀 코어(LITTLE Core)에서 실행되고 있는지 판단한다. 만일, 스케줄링 요청된 쓰레드가 저속 코어 또는 리틀 코어(LITTLE Core)에서 실행중인 경우, 절차는 S245 단계로 이동한다. 반면, 스케줄링 요청된 쓰레드가 고속 코어 또는 빅 코어(big Core)에서 실행중인 경우, 평균 부하에 따라 코어 재할당을 수행하기 위한 S245 단계로 이동한다.

[0084] S230 단계에서, 스케줄러(135)는 스케줄링 요청된 쓰레드의 컨텍스트와 동일한 컨텍스트를 갖는 쓰레드를 검색할 것이다. 스케줄러(135)는 쓰레드 풀(Thread pool)에 존재하는 복수의 쓰레드들에 대한 컨텍스트의 비교를 수행한다. 예를 들면, 스케줄러(135)는 멀티-쓰레딩을 수행하기 위한 동일한 쓰레드 그룹(또는, 쓰레드 풀)에서 쓰레드 레지스터, 커널 스택이나 사용자 스택 등의 쓰레드에 대한 실행 정보를 검색하여 스케줄링 요청된 쓰레드의 컨텍스트와 비교할 수 있다.

[0085] S235 단계에서, 스케줄러(135)는 실행중인 쓰레드들 중에서 스케줄링 요청된 쓰레드의 컨텍스트를 공유하는 것이 존재하는지 판단할 것이다. 만일, 현재 실행중인 쓰레드들 중에서 스케줄링 요청된 쓰레드의 컨텍스트와 동

일한 것이 존재하는 것으로 판단되면(Yes 방향), 절차는 S240 단계로 이동한다. 반면, 현재 실행중인 쓰레드들 중에서 스케줄링 요청된 쓰레드의 컨텍스트와 동일한 것이 존재하지 않는 것으로 판단되면(No 방향), 절차는 S245 단계로 이동한다.

[0086] S240 단계에서, 스케줄러(135)는 스케줄링 요청된 쓰레드의 컨텍스트를 공유하는 쓰레드가 고속 코어 또는 빅 클러스터에서 실행되는지 판단한다. 만일, 컨텍스트를 공유하는 쓰레드가 고속 코어나 빅 클러스터가 아닌 코어에서 실행되고 있다면, 절차는 추가적으로 쓰레드를 검출하기 위한 S230 단계로 이동한다. 반면, 컨텍스트를 공유하는 쓰레드가 고속 코어나 빅 클러스터에서 실행되고 있다면, 절차는 S270 단계로 이동한다.

[0087] S245 단계에서, 스케줄러(135)는 S220 단계에서 계산된 스케줄링 요청된 쓰레드에 대해서 평균 부하의 값을 기초로 실행 코어를 선택할 것이다. 예를 들면, 평균 부하가 상한 기준치(uTH)를 초과하면, 절차는 고속 또는 빅 코어를 선택하기 위한 S270 단계로 이동한다. 반면, 평균 부하가 상한 기준치(uTH) 이하이면, 절차는 S250 단계로 이동한다.

[0088] S250 단계에서, 스케줄링 요청된 쓰레드의 평균 부하가 하한 기준치(dTH)와 같거나 크면(No), 절차는 현재 할당된 코어나 클러스터를 유지하기 위하여 S255 단계로 이동한다. 반면, 스케줄링 요청된 쓰레드의 평균 부하가 하한 기준치(dTH) 미만(Yes)이면, 절차는 저속 또는 리틀 코어를 선택하기 위한 S260 단계로 이동한다.

[0089] S255 단계는 특정 정책에 따라 스케줄링 요청된 쓰레드가 임시적으로 실행되는 코어 또는 클러스터의 할당을 유지하는 것을 나타낸다. S260 단계는 특정 정책에 따라 스케줄링 요청된 쓰레드를 실행하기 위한 코어 또는 클러스터로 저속 또는 리틀 클러스터를 선택하는 절차를 보여준다. S270 단계는 특정 정책에 따라 스케줄링 요청된 쓰레드를 실행하기 위한 코어 또는 클러스터로 고속 또는 빅 클러스터를 선택하는 절차를 보여준다.

[0090] S280 단계에서, 스케줄러(135)는 컨텍스트를 공유하는 쓰레드를 참조하여 선택된 코어 또는 클러스터에서 스케줄링 요청된 쓰레드를 실행하도록 제어할 것이다.

[0091] 이상에서 본 발명의 개략적인 스케줄링 방법이 설명되었다. 이종 멀티-쓰레드 방식의 코어 제어에 있어서, 스케줄링 이벤트가 발생하면 컨텍스트를 공유하는 쓰레드의 존재 유무가 검출될 것이다. 그리고 검출 결과에 따라 스케줄링 요청된 쓰레드를 고속 또는 빅 클러스터로 재할당할지가 결정된다. 이러한 쓰레드의 관리 방법에 따라 컨텍스트를 공유하는 쓰레드들이 서로 다른 성능의 코어들에 분산되어 발생하는 비효율성은 차단될 수 있다.

[0092] 도 10은 본 발명의 실시 예에 따른 휴대용 단말기를 나타내는 블록도이다. 도 14를 참조하면, 본 발명의 실시 예에 따른 휴대용 단말기(1000)는 이미지 처리부(1100), 무선 송수신부(1200), 오디오 처리부(1300), 이미지 파일 생성부(1400), 불휘발성 메모리 장치(1500), 사용자 인터페이스(1600), 그리고 컨트롤러(1700)를 포함한다.

[0093] 이미지 처리부(1100)는 렌즈(1110), 이미지 센서(1120), 이미지 프로세서(1130), 그리고 디스플레이부(1140)를 포함한다. 무선 송수신부(1210)는 안테나(1210), 트랜시버(1220), 모뎀(1230)을 포함한다. 오디오 처리부(1300)는 오디오 프로세서(1310), 마이크(1320), 그리고 스피커(1330)를 포함한다.

[0094] 여기서, 불휘발성 메모리 장치(1500)는 본 발명의 실시 예에 따른 메모리 카드(MMC, eMMC, SD, micro SD) 등으로 제공될 수 있다. 더불어, 컨트롤러(1700)는 응용 프로그램, 운영 체제 등을 구동하는 시스템 온 칩(SoC)으로 제공될 수 있다. 시스템 온 칩(SoC)에서 구동되는 운영 체제의 커널(Kernel)에는 본 발명의 실시 예에 따라 쓰레드를 관리하는 스케줄러와 CPU 거버너가 포함될 수 있다. 커널에 의해서 서로 다른 성능의 코어들이 동시에 활성화되는 프로세서들에 컨텍스트를 공유하는 쓰레드들이 분산되는 동작 상태를 차단하거나 바로잡을 수 있다. 따라서, 이종 멀티-프로세싱을 수행하는 응용 프로세서의 전력 또는 처리 속도의 효율을 높일 수 있을 것이다.

[0095] 본 발명에 따른 시스템 온 칩 그리고/또는 멀티-코어 프로세서는 다양한 형태들의 패키지를 이용하여 실장될 수 있다. 예를 들면, 본 발명에 따른 플래시 메모리 장치 그리고/또는 메모리 컨트롤러는 PoP(Package on Package), Ball grid arrays(BGAs), Chip scale packages(CSPs), Plastic Leaded Chip Carrier(PLCC), Plastic Dual In-Line Package(PDIP), Die in Waffle Pack, Die in Wafer Form, Chip On Board(COB), Ceramic Dual In-Line Package(CERDIP), Plastic Metric Quad Flat Pack(MQFP), Thin Quad Flatpack(TQFP), Small Outline(SOIC), Shrink Small Outline Package(SSOP), Thin Small Outline(TSOP), System In Package(SIP), Multi Chip Package(MCP), Wafer-level Fabricated Package(WFP), Wafer-Level Processed Stack Package(WSP), 등과 같은 패키지들을 이용하여 실장 될 수 있다.

[0096] 이상에서와 같이 도면과 명세서에서 실시 예가 개시되었다. 여기서 특정한 용어들이 사용되었으나, 이는 단지 본 발명을 설명하기 위한 목적에서 사용된 것이지 의미 한정이나 특허 청구범위에 기재된 본 발명의 범위를 제

한하기 위하여 사용된 것은 아니다. 그러므로 본 기술분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시 예가 가능하다는 점을 이해할 것이다. 따라서 본 발명의 진정한 기술적 보호 범위는 첨부된 특허 청구범위의 기술적 사상에 의해 정해져야 할 것이다.

부호의 설명

[0097] 100 : 모바일 장치 110 : CPU

112 : 제 1 클러스터 114 : 제 2 클러스터

116 : 캐시 코히런트 인터페이스

120 : 디램 컨트롤러 130 : 워킹 메모리

132 : 응용 프로그램 134 : 커널

135 : 스케줄러 137: CPU 거버너

140 : 성능 컨트롤러 150 : UI 컨트롤러

152 : 액정 표시 장치 154 : 터치 패널

160 : 메모리 인터페이스 170 : 저장 장치

180 : 가속기 190 : 시스템 인터커넥터

1000 : 휴대용 단말기 1100 : 이미지 처리부

1110 : 렌즈 1120 : 이미지 센서

1130 : 이미지 프로세서 1140 : 디스플레이 유닛

1200 : 무선 송수신부 1210 : 안테나

1220 : 송수신기 1230 : 모뎀

1300 : 오디오 처리부 1310 : 오디오 프로세서

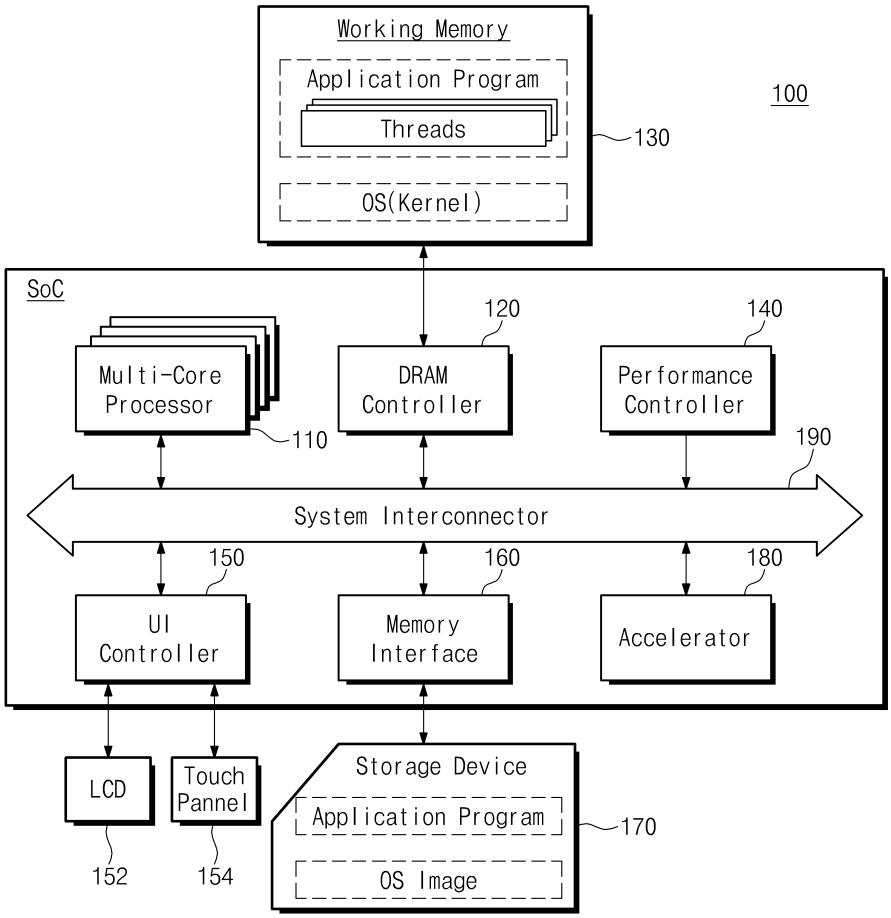
1320 : 마이크 1330 : 스피커

1400 : 이미지 파일 생성부 1500 : 불휘발성 메모리 장치

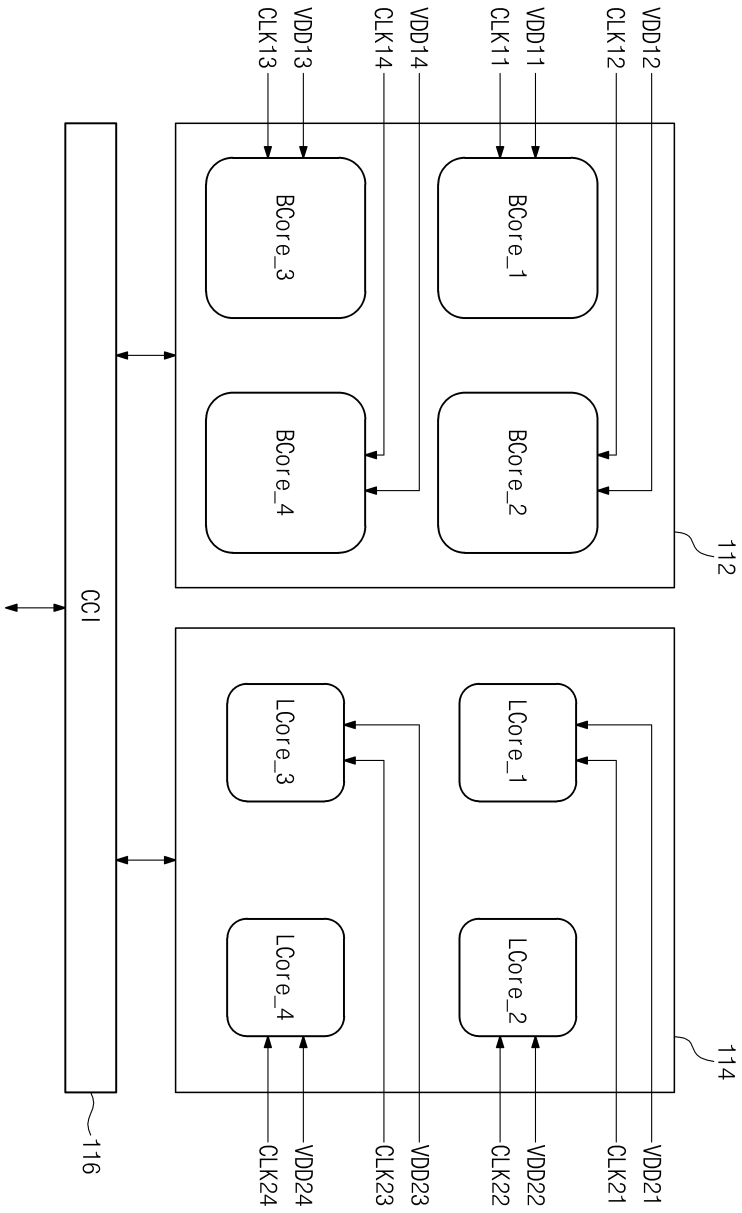
1600 : 유저 인터페이스 1700 : 컨트롤러

도면

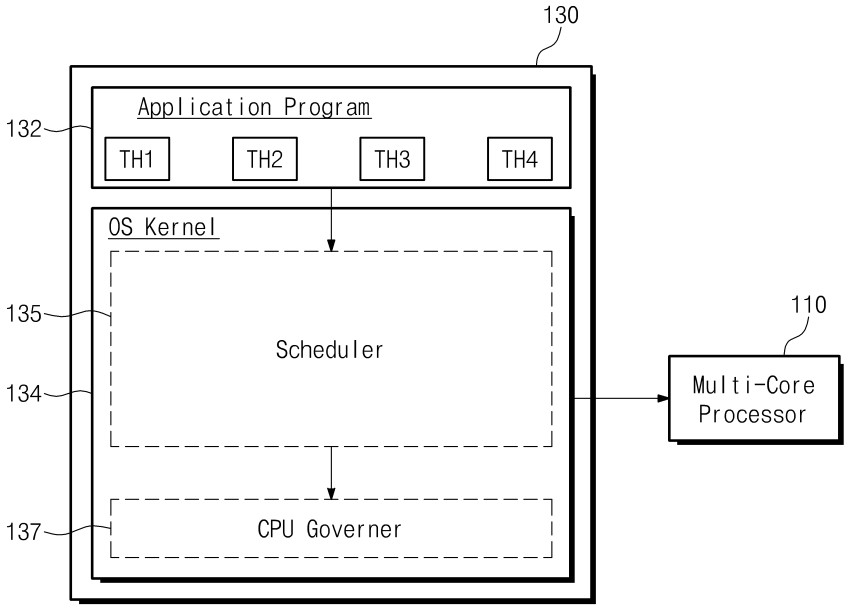
도면1


Working Memory
Application Program
Threads
OS(Kernel)
130
100
SoC
Multi-Core Processor
110
120
DRAM Controller
140
Performance Controller
190
System Interconnector
150
160
180
UI Controller
Memory Interface
Accelerator
LCD
Touch Pannel
152
154
Storage Device
Application Program
OS Image
170

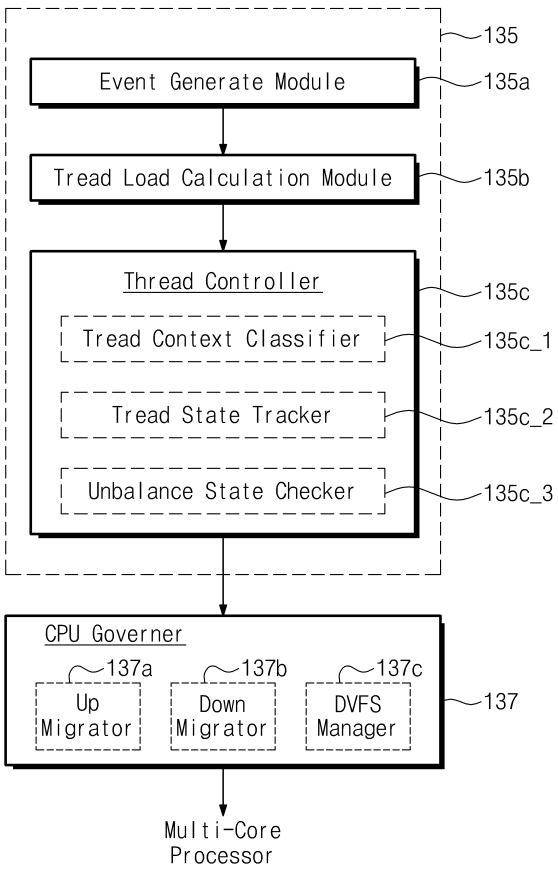
도면2


VDD12
CLK12
VDD11
CLK11
VDD14
CLK14
VDD13
CLK13
BCore_1
BCore_2
BCore_3
BCore_4
112
CCI
116
LCore_1
LCore_2
LCore_3
LCore_4
114
VDD21
CLK21
VDD22
CLK22
VDD23
CLK23
VDD24
CLK24

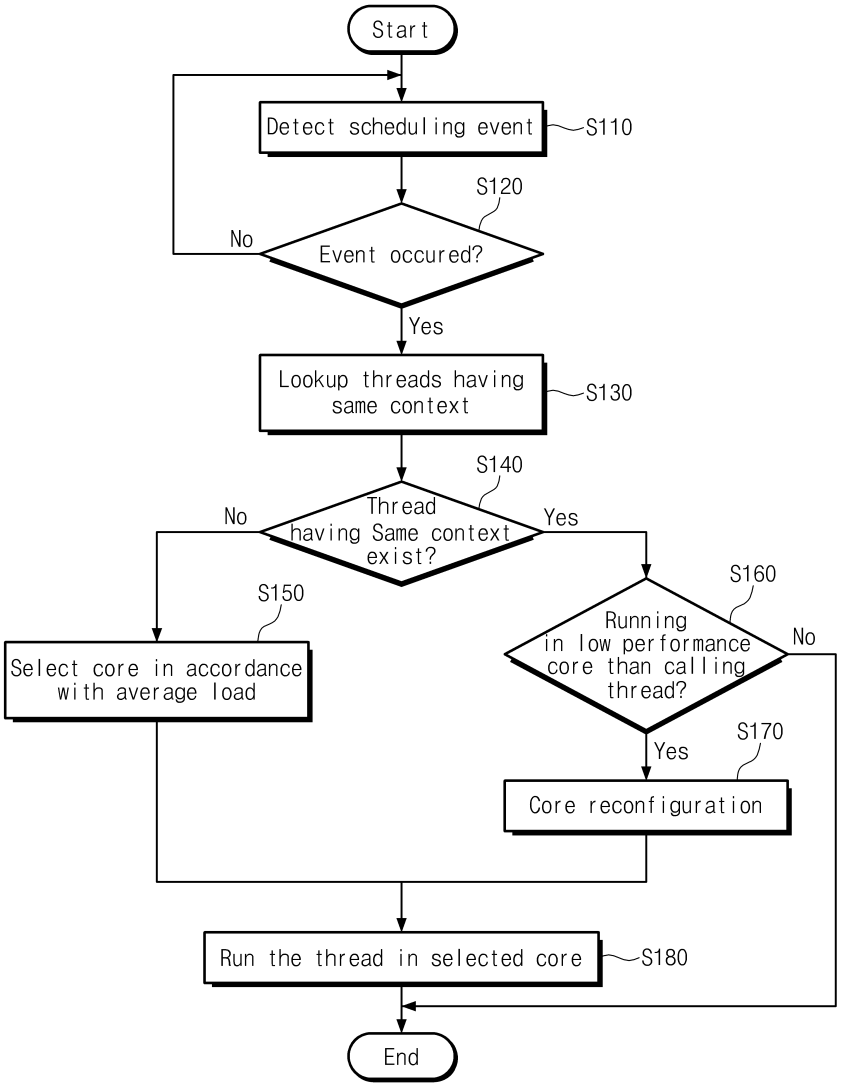
도면3


130
Application Program
132
TH1
TH2
TH3
TH4
OS Kernel
135
Scheduler
134
110
Multi-Core Processor
137
CPU Governer

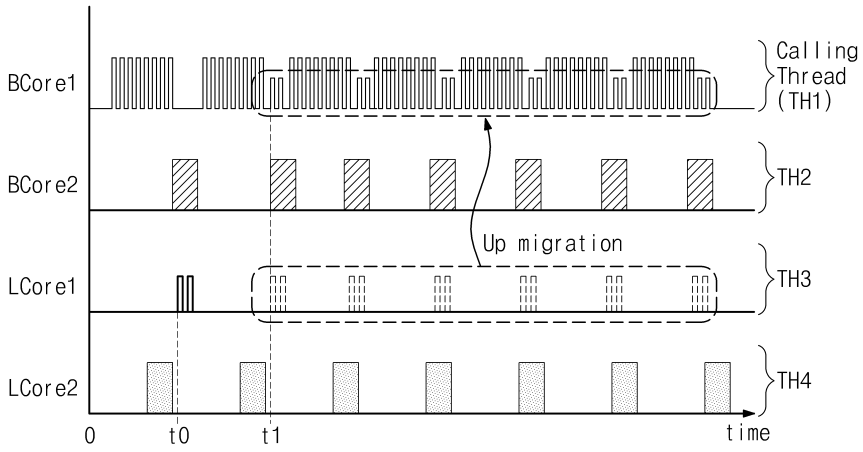
도면4


135
Event Generate Module
135a
Tread Load Calculation Module
135b
Thread Controller
135c
Tread Context Classifier
135c_1
Tread State Tracker
135c_2
Unbalance State Checker
135c_3
CPU Governer
137a
137b
137c
Up Migrator
Down Migrator
DVFS Manager
137
Multi-Core Processor

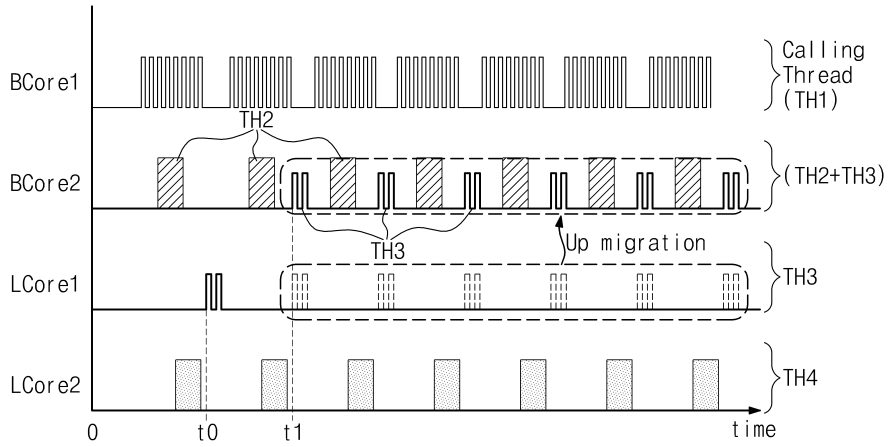
도면5


Start
Detect scheduling event
S110
S120
No
Event occured?
Yes
Lookup threads having same context
S130
S140
No
Thread having Same context exist?
Yes
S150
Select core in accordance with average load
S160
Running in low performance core than calling thread?
No
Yes
S170
Core reconfiguration
Run the thread in selected core
S180
End

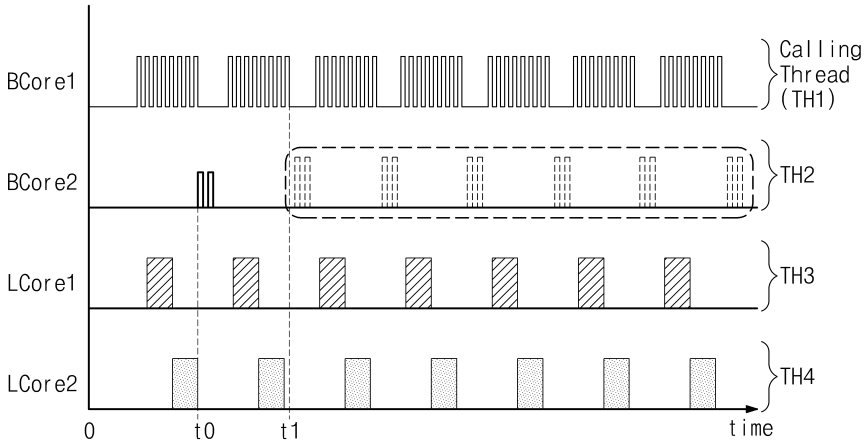
도면6


BCore1
BCore2
LCore1
LCore2
Calling Thread (TH1)
TH2
Up migration
TH3
TH4
0
t0
t1
time

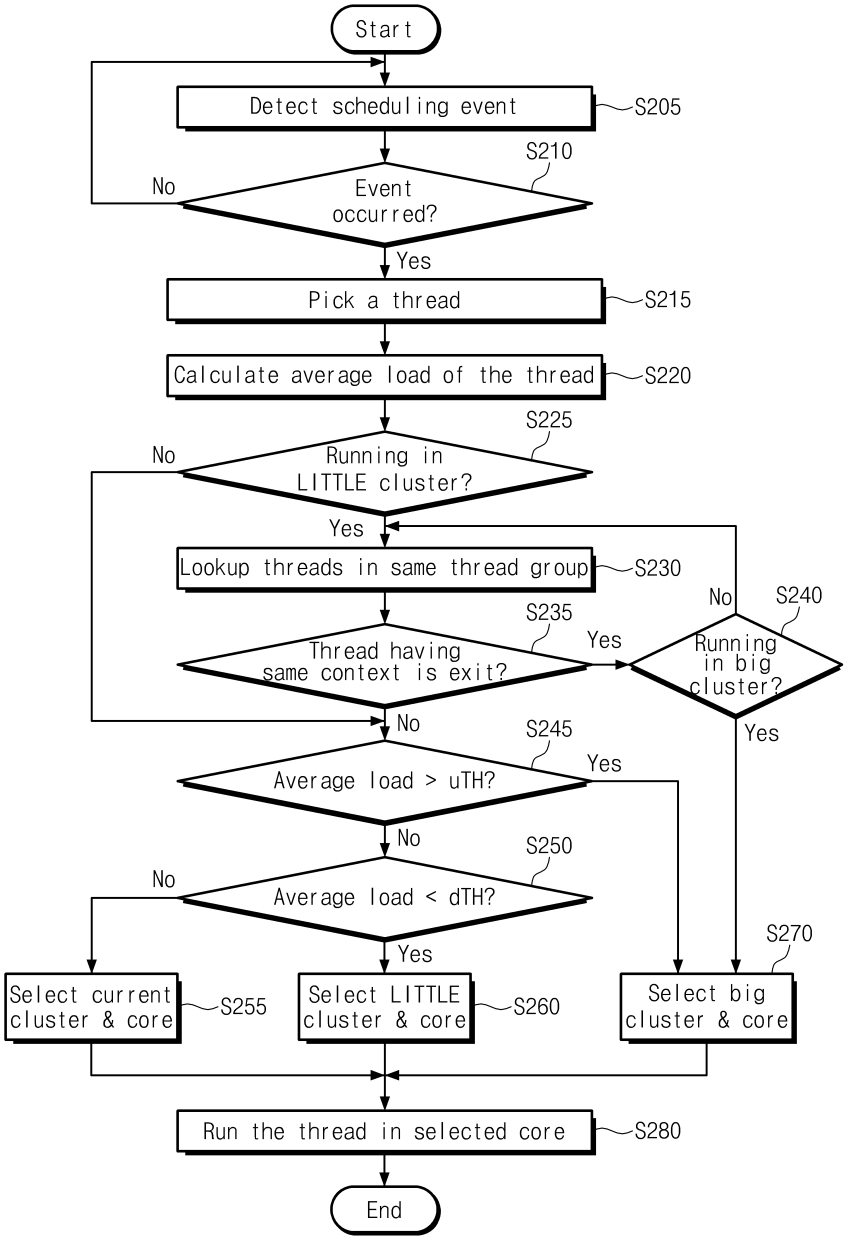
도면7


BCore1
BCore2
LCore1
LCore2
Calling
Thread
(TH1)
TH2
(TH2+TH3)
TH3
Up migration
TH3
TH4
0
t0
t1
time

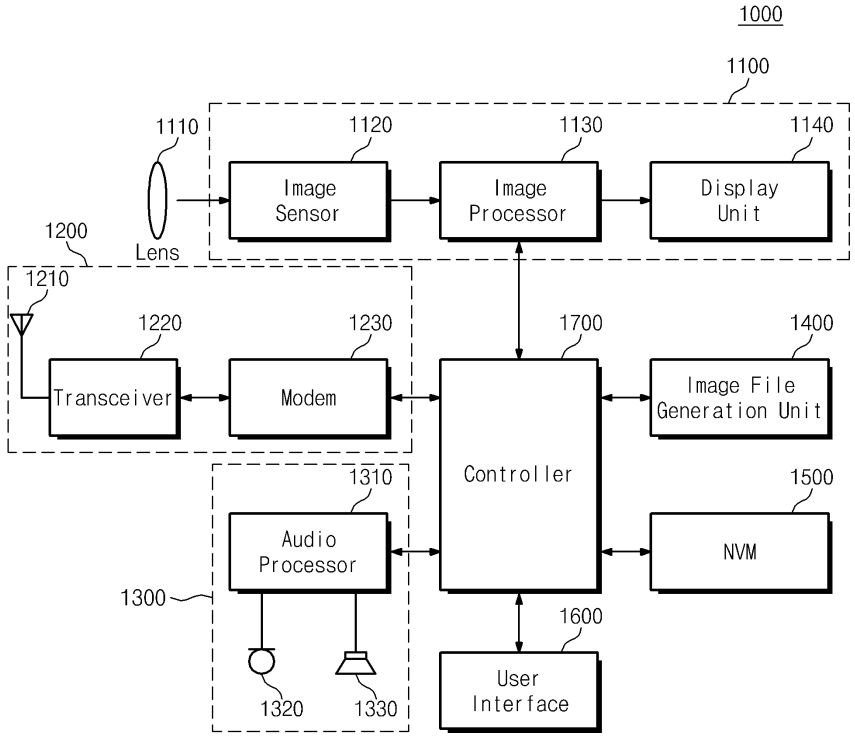
도면8


BCore1
BCore2
LCore1
LCore2
Calling
Thread
(TH1)
TH2
TH3
TH4
0
t0
t1
time

도면9


Start
Detect scheduling event
S205
S210
No
Event occurred?
Yes
Pick a thread
S215
Calculate average load of the thread
S220
S225
No
Running in LITTLE cluster?
Yes
Lookup threads in same thread group
S230
No
S240
S235
Thread having same context is exit?
Yes
Running in big cluster?
Yes
No
S245
Average load > uTH?
Yes
No
S250
No
Average load < dTH?
Yes
S270
Select current cluster & core
S255
Select LITTLE cluster & core
S260
Select big cluster & core
Run the thread in selected core
S280
End

도면10


1000
1100
1110
1120
1130
1140
Image Sensor
Image Processor
Display Unit
Lens
1200
1210
1220
1230
1700
1400
Transceiver
Modem
Controller
Image File Generation Unit
1310
1500
Audio Processor
NVM
1300
1600
User Interface
1320
1330