(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2010-0062958
(43) 공개일자 2010년06월10일

(51) Int. Cl.
G06F 9/50 (2006.01) *G06F 9/46* (2006.01)
(21) 출원번호 10-2009-0117865
(22) 출원일자 2009년12월01일
심사청구일자 2009년12월01일
(30) 우선권주장
12/315,331 2008년12월02일 미국(US)

(71) 출원인
인텔 코오퍼레이션
미합중국 캘리포니아 산타클라라 미션 칼리지 블러바드 2200
(72) 발명자
하센플라우그, 윌리암
미국 02116 매사추세츠주 보스턴 워렌 애비뉴 넘버8 15
에머, 조엘
미국 01720 매사추세츠주 액턴 블랙 호스 디알 20
(뒷면에 계속)
(74) 대리인
양영준, 백만기

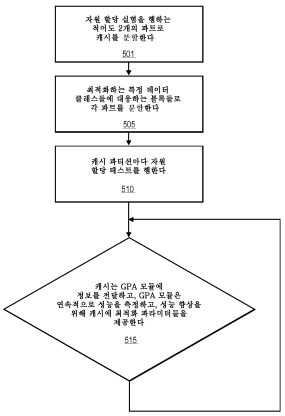
전체 청구항 수 : 총 24 항

(54) 컴퓨팅 자원들을 제어하는 기술

(57) 요 약

본원은 컴퓨터 시스템 내에서의 자원 할당 최적화를 가능하게 하는 기술에 관한 것이다. 일 실시예에서, 그레이디언트 파티션 알고리즘(GPA) 모듈은 최적의 성능을 달성하기 위해서 연속적으로 성능을 측정하고 복수의 데이터 클래스 사이에 공유된 자원에 대한 할당을 조정한다.

대 표 도 - 도5



(72) 발명자

포섬, 트리그베

미국 01532 매사추세츠주 노스보로우 인크리스 워드 드라이브 19

자릴, 아머

미국 01749 매사추세츠주 허드슨 센트럴 에스티 에이피티 2309 307

스틸리, 시몬

미국 03051 뉴햄프셔주 허드슨 안나 루이스 디알 8

특허청구의 범위

청구항 1

적어도 하나의 프로세싱 자원을 적어도 하나의 명령어 스레드에 할당하는 것을 연속적으로 제어하는 그레이디언트 파티션 알고리즘(GPA) 모듈

을 포함하며,

상기 GPA 모듈은 상기 프로세싱 자원에 대한 테스트들을 연속적으로 행하여 상기 적어도 하나의 명령어 스레드의 프로세싱 요구를 최적으로 충족시키는 동적 스롯틀 설정(dynamic throttle setting)을 결정하는 장치.

청구항 2

제1항에 있어서,

상기 프로세싱 자원은 캐시 메모리를 포함하는 장치.

청구항 3

제2항에 있어서,

상기 캐시 메모리는 상기 GPA 모듈이 상기 테스트들을 행하는 적어도 2개의 파트로 논리적으로 분할되는 장치.

청구항 4

제3항에 있어서,

상기 2개의 논리적으로 분할된 파트의 각각은 상기 적어도 하나의 명령어 스레드에 대응하는 적어도 하나의 블록을 포함하는 장치.

청구항 5

제1항에 있어서,

상기 GPA 모듈은 제1 및 제2 설정으로 설정되는 스롯틀 설정의 기능으로서 상기 적어도 하나의 프로세싱 자원의 성능을 테스트하며, 상기 제1 설정은 상기 제2 설정보다 저 레벨로 상기 프로세싱 자원을 행하게 하는 장치.

청구항 6

제5항에 있어서,

상기 GPA 모듈은 최고 성능 레벨로 되게 하는 스롯틀 설정을 지시하는 정보를 상기 프로세싱 자원에 제공하는 장치.

청구항 7

제4항에 있어서,

상기 GPA 모듈은 상기 적어도 하나의 명령어에 대응하는 클래스 데이터 비트들을 포함하는 장치.

청구항 8

제7항에 있어서,

상기 GPA 모듈은 복수의 명령어에 대응하는 글로벌 데이터 비트들을 포함하는 장치.

청구항 9

프로세서 코어와,

상기 프로세서 코어에 연결된 캐시와,

상기 캐시에 대한 테스트들을 연속적으로 행하여 상기 캐시에 액세스하는 복수의 명령어의 프로세싱 요구를 최

적으로 충족시키는 동적 스롯틀 설정을 결정하는 그레이디언트 파티션 알고리즘(GPA) 모듈

을 포함하는 시스템.

청구항 10

제9항에 있어서,

상기 캐시는 상기 GPA 모듈이 상기 테스트들을 행하는 적어도 2개의 파트로 논리적으로 분할되는 시스템.

청구항 11

제10항에 있어서,

상기 2개의 논리적으로 분할된 파트의 각각은 상기 복수의 명령어에 대응하는 복수의 블록을 포함하는 시스템.

청구항 12

제11항에 있어서,

상기 복수의 블록은 상기 스롯틀의 값에 따라 사이즈가 증가 또는 감소하는 시스템.

청구항 13

제12항에 있어서,

상기 테스트들은 상기 스롯틀에 의해 블록 사이즈가 증가할 때 제1 블록에 대응하는 제1 스레드의 성능을 측정하는 것을 포함하는 시스템.

청구항 14

제13항에 있어서,

상기 테스트들은 상기 스롯틀에 의해 블록 사이즈가 감소할 때 상기 제1 블록에 대응하는 상기 제1 스레드의 성능을 측정하는 것을 포함하는 시스템.

청구항 15

제14항에 있어서,

상기 GPA 모듈은 상기 테스트들 중에 달성되는 성능에 따라 상기 제1 블록의 사이즈를 설정하는 시스템.

청구항 16

제15항에 있어서,

상기 GPA 모듈은 상기 복수의 명령어의 각각에 대응하는 클래스 데이터 비트들을 포함하는 시스템.

청구항 17

제16항에 있어서,

상기 GPA 모듈은 상기 복수의 명령어의 전부에 대응하는 글로벌 데이터 비트들을 포함하는 시스템.

청구항 18

자원 할당 테스트들을 행하는 적어도 2개의 파트로 캐시를 분할하는 단계와,

복수의 데이터 클래스에 대응하는 복수의 블록으로 상기 적어도 2개의 파트의 각각을 분할하는 단계와,

상기 적어도 2개의 파트의 각각에 대한 테스트들을 연속적으로 행하고 그 성능 결과들을 비교하는 단계 - 제1 스롯틀 설정에서 상기 적어도 2개의 파트 중 하나에 대한 테스트를 행하고, 제2 스롯틀 설정에서 상기 적어도 2개의 파트 중 다른 하나에 대한 테스트를 행함 - 와,

상기 성능 결과들의 비교에 응답하여 상기 복수의 블록 중 적어도 하나의 사이즈를 조정하는 단계

를 포함하는 방법.

청구항 19

제18항에 있어서,

상기 복수의 블록은 상기 스롯틀의 설정에 따라 사이즈가 증가 또는 감소하는 방법.

청구항 20

제19항에 있어서,

상기 테스트들은 상기 스롯틀 설정에 의해 블록 사이즈가 증가할 때 제1 블록에 대응하는 제1 스레드의 성능을 측정하는 것을 포함하는 방법.

청구항 21

제20항에 있어서,

상기 테스트들은 상기 스롯틀에 의해 블록 사이즈가 감소할 때 상기 제1 블록에 대응하는 상기 제1 스레드의 성능을 측정하는 것을 포함하는 방법.

청구항 22

제21항에 있어서,

상기 테스트들 중에 달성되는 성능에 따라 제1 블록의 사이즈를 설정하는 단계를 포함하는 방법.

청구항 23

제22항에 있어서,

상기 복수의 데이터 클래스의 각각에 대응하는 클래스 데이터 비트들이 유지되는 방법.

청구항 24

제23항에 있어서,

상기 복수의 데이터 클래스의 전부에 대응하는 글로벌 데이터 비트들이 유지되는 방법.

명 세 서

발명의 상세한 설명

기 술 분 야

[0001] 본 발명의 실시예들은 일반적으로 정보 처리 분야에 관한 것으로, 특히 컴퓨팅 시스템들 및 마이크로프로세서들에서 자원들을 할당하는 분야에 관한 것이다.

배 경 기 술

[0002] 컴퓨터 시스템 또는 프로세서에서 자원들을 할당하는 것은 어려운 일일 수 있다. 예를 들면, 캐시 공간, 메모리, 실행 자원들 등과 같은 자원들이 "정적" 방식(즉, 변화하는 자원 요구에 응답하여 변화하지 않음)으로 할당되는 일부 컴퓨터 시스템들에서는, 컴퓨팅 시스템 또는 프로세서는 특정 프로세스들 또는 스레드(thread)들에는 적게 서비스하고(under-service) 그외의 프로세스들 또는 스레드들에는 과도하게 서비스할(over-service) 수 있다. 그러나, 일부 종래기술의 "동적" 자원 할당 방식들(즉, 그들이 서비스하는 스레드들, 프로세스들 등의 변화하는 요구에 응답하기를 시도하는 방식들)에서 조차도, 자원들의 동적 할당과 연관된 오버헤드는 그러한 할당에 의해 제공되는 성능상의 이익의 가치가 없을 수 있다. 따라서, 개선된 자원 할당 메카니즘은 할당 방식의 이익을 방해할 수 있는 오버헤드를 발생시키지 않으면서 프로세서 또는 컴퓨팅 시스템 성능을 모두 개선시킬 수 있다.

발명의 내용

발명의 실시를 위한 구체적인 내용

[0003] 본 발명의 실시예들은 본 명세서에서 기술되는 자원 할당 기술들의 이점을 방해하는 오버헤드를 발생시키지 않으면서 전반적인 시스템 또는 프로세싱 성능을 개선시키기 위한 동적 자원 할당 기술에 관한 것이다. 일부 실시예들에 있어서, 동적 자원 할당 방식은 스레드들, 메모리 자원들, 및 실행 자원들 중에서 캐시 할당과 같은 다수의 상이한 프로세싱 자원들에 적용될 수 있다. 예시를 위해, 이하의 설명은 적어도 하나의 실시예에 따라, 두개 이상의 스레드들에 의해 공유되는 캐시 자원들에 자원 할당 기술을 적용하는 예시에 주로 촛점을 맞출 것이다. 그러나, 본 명세서에 기술되는 실시예들은 본 명세서에서 특히 논의되는 것 이외의 그외의 다른 컴퓨팅 자원들 및 다른 수의 스레드들에 적용될 수 있다.

[0004] 일 실시예에 있어서, 복수의 스레드 중에서 캐시 공간을 할당하기 위해 그레이디언트 기반 파티션 알고리즘(GPA)이 이용된다. 일 실시예에 있어서, GPA는 상태 머신, 데이터 클래스 당 3개의 레지스터들(예를 들어, 스트리밍된 데이터, 재사용된 데이터 등) 및 데이터 클래스에 의존하지 않는 4개의 글로벌 레지스터들을 사용한다. 일 실시예에 있어서, GPA는, 데이터 클래스에 대한 리소스를 점진적으로 더 많이 그 다음에 점진적으로 더 적게 하는 것으로 하여 데이터 클래스마다 실험을 행하는 그레이디언트 디센트(gradient descent)(또는 "힐 클라이밍(hill climbing)") 알고리즘의 변화를 이용하여 공유 자원의 최적 할당을 찾아낸다. 그 다음에 GPA는 양 시나리오에 대한 "글로벌 굿니스(global goodness)" 메트릭을 측정하고 실험에 따라 해당 데이터 클래스에 공칭 정량(nominal ration)의 자원을 할당한다. 일 실시예에 있어서, GPA는 체르노프 한계(Chernoff bounds)를 이용하여 자원의 분할을 조정하는 시기를 결정하고, 다른 실시예에서는, 다른 알고리즘을 이용하여 자원의 분할 조정 시기를 결정할 수 있다. 또한, 일 실시예에서는, 프로세싱 사이클들에 걸쳐 실험 프로세스를 분할하는 것에 의해 동시에 데이터 클래스마다 전술한 실험을 행할 수 있다(예를 들면, 대역폭 또는 전력 관리에 대한 시간 다중화, 캐시 또는 프리페치 관리에 대한 메모리 공간 분할).

[0005] 일부 실시예들에 있어서, 자원을 공유하는 멀티프로세서 시스템들 또는 멀티코어 프로세서들에 GPA를 적용하는 것은, 일부 실시예들이 계속적으로 자원 관리를 능동적으로 최적화한다는 사실에 부분적으로 기인하여 종래 기술의 시스템들보다 더욱 높은 성능 및 더욱 낮은 전력 소모를 달성할 수 있게 한다. 따라서, 실시예들은 보다 긴 배터리 수명, 더욱 우수한 블레이드당 성능, 고밀도의 클라우드 컴퓨팅 등이 되게 할 수 있다.

[0006] 예를 들면, 서버 프로세서들을 사용하는 환경과 같은 일부 컴퓨팅 환경들에서는, 어떤 데이터는 재사용되고(예를 들면, 패킷 헤더들, 라우팅 테이블들, 명령어 데이터, 운영 체제 상태, 그외의 메타 데이터 유사 통계들 등), 어떤 데이터는 스트리밍된다(예를 들면, 패킷 바디 정보). 일부 실시예들에 따라 관리되지 않는 캐시를 이용하는 것은 재사용된 데이터 전체가 그것이 재사용되는 기회를 갖기 전에 스트리밍 데이터에 의해 축출될 수 있기 때문에, 재사용된 데이터를 무용으로 할 수 있다. 일 실시예에 있어서, GPA는, 애플리케이션들이 프로세싱 아키텍처를 인식하지도 않고 또한 동일 머신 상에서 구동하는 그외의 애플리케이션들을 인식하지 않는, 가상 머신 팜(farm) 등의 애플리케이션들에 유용할 수 있는, 아키텍처 인식 최적화 없이 기입되는 애플리케이션에서조차도, 어느 데이터를 캐시 내에 유지하고 어느 데이터를 스트리밍할지를 동적으로 결정할 수 있다.

[0007] 도 1은 본 발명의 적어도 하나의 실시예가 이용될 수 있는 마이크로프로세서를 도시한다. 특히, 도 1은 각각이 로컬 캐시(107, 113)와 각각 연관된 하나 이상의 프로세서 코어들(105, 110)을 갖는 마이크로프로세서(100)를 도시한다. 또한 도 1에는 로컬 캐시들(107, 113)의 각각에 저장된 정보의 적어도 일부의 버젼들을 저장할 수 있는 공유 캐시 메모리(115)가 도시되어 있다. 일부 실시예들에 있어서, 마이크로프로세서(100)는 또한 집적 메모리 제어기, 집적 그래픽 제어기 등의, 도 1에 도시되지 않은 그외의 로직과, 컴퓨터 시스템 내에서 I/O 제어 등의 그외의 기능들을 수행하기 위한 그외의 로직을 포함할 수 있다. 일 실시예에서는, 멀티프로세서 시스템 내의 각각의 마이크로프로세서 또는 멀티코어 프로세서 내의 각각의 프로세서 코어는 로직(119)을 포함하거나 또는 다른 경우에는 로직(119)과 연관되어 적어도 하나의 실시예에 따라 컴퓨팅 자원 할당 기술들을 가능하게 할 수 있다. 로직은 회로들, (유형의 매체(tangible medium)에 구현된) 소프트웨어 또는 양쪽을 포함하여, 복수의 코어 또는 프로세서 중에서, 일부 종래 기술의 구현에서보다도 더욱 효율적인 자원 할당을 가능하게 할 수 있다.

[0008] 도 2는, 예를 들면, 본 발명의 일 실시예가 이용될 수 있는 FSB(front-side-bus) 컴퓨터 시스템을 도시한다. 임의의 프로세서(201, 205, 210, 또는 215)는 프로세서 코어들(223, 227, 233, 237, 243, 247, 253, 257) 중 하나 내의 또는 다른 경우에는 프로세서 코어들(223, 227, 233, 237, 243, 247, 253, 257) 중 하나와 연관된

임의의 로컬 레벨 1(L1) 캐시 메모리(220, 225, 230, 235, 240, 245, 250, 255)로부터 정보를 액세스할 수 있다. 또한, 임의의 프로세서(201, 205, 210, 또는 215)는 공유된 레벨 2(L2) 캐시들(203, 207, 213, 217) 중 임의의 하나로부터 혹은 칩셋(265)을 통해 시스템 메모리(260)로부터 정보를 액세스할 수 있다. 도 2의 하나 이상의 프로세서들은 적어도 하나의 실시예에 따라 자원 할당 기술을 가능하게 하기 위해 로직(219)을 포함하거나 또는 다른 경우에는 로직(219)과 연관될 수 있다.

[0009] 도 2에 도시된 FSB 컴퓨터 시스템 외에도, 본 발명의 다양한 실시예들과 관련하여 점대점(P2P) 상호접속 시스템들 및 링 상호접속 시스템들을 포함하는 그외의 시스템 구성들이 이용될 수 있다. 도 3의 P2P 시스템은 예를 들면 여러개의 프로세서들을 포함할 수 있으며, 예로서 이들 중 2개의 프로세서들(370, 380)만이 도시된다. 프로세서들(370, 380)은 각각이 메모리(32, 34)와 접속하기 위해 로컬 메모리 제어기 허브(MCH)(372, 382)를 포함할 수 있다. 프로세서들(370, 380)은 PtP(point-to-point) 인터페이스 회로들(378, 388)을 이용하여 PtP 인터페이스(350)를 통해 데이터를 교환할 수 있다. 프로세서들(370, 380)은 각각이 PtP 인터페이스 회로들(376, 394, 386, 398)을 이용하여 개별적인 PtP 인터페이스들(352, 354)을 통해 칩셋(390)과 데이터를 교환할 수 있다. 칩셋(390)은 또한 고성능 그래픽 인터페이스(339)를 통해 고성능 그래픽 회로(338)와 데이터를 교환할 수 있다. 본 발명의 실시예들은 임의의 수의 프로세싱 코어들을 갖는 임의의 프로세서 내에, 또는 도 3의 PtP 버스 에이전트들 각각 내에 배치될 수 있다. 일 실시예에 있어서, 임의의 프로세서 코어는 로컬 캐시 메모리(도시되지 않음)를 포함하거나 또는 다른 경우에는 이와 연관될 수 있다. 또한, 공유 캐시(도시되지 않음)는 p2p 인터커넥트를 통해 프로세서들과 여전히 접속되어 있는, 양쪽 프로세서들 중 외측의 어느 하나의 프로세서에 포함되어, 프로세서가 저전력 모드에 놓이면, 어느 하나 또는 양쪽 프로세서들의 로컬 캐시 정보가 공유 캐시에 저장되도록 할 수 있다. 도 3의 프로세서들 또는 코어들 중 하나 이상은 적어도 하나의 실시예에 따라 자원 할당 기술을 가능하게 하도록 로직(319)을 포함하거나 다른 경우에는 로직(319)과 연관될 수 있다.

[0010] 도 4a는 본 발명의 적어도 하나의 실시예에 따라 자원 관리된 캐시의 개념화를 도시한다. 일 실시예에 있어서, 캐시(400a)는 2개의 파트(401a, 405a)로 논리적으로 분할되고, 이들 각각은 3개의 상이한 스레드들에 대응하는 3개의 블럭(410a-412a)으로 분할된다. 일 실시예에 있어서, 각 분할에서의 3개의 블록은 스롯틀(throttle)("T")의 사용에 의해, 특정 데이터 클래스에 대응하는 각 스레드의 변화하는 요구에 응답하여 제어된다. 일 실시예에 있어서, 스롯틀은 데이터 클래스에 공유 자원에 대한 최우선순위 액세스가 주어지는 시간 비율을 나타낸다. 일 실시예에 있어서, 스롯틀의 세트가 주어지는, 낮은 우선순위 액세스에 대해서보다 높은 우선순위 액세스에 대해 알맞은 자원들의 할당은 공유 자원의 최적 할당을 보증한다. 일 실시예에 있어서, 스롯틀(420a, 425a)은 캐시 내의 2개의 논리 파트(예를 들면, 세트들) 내의 유효 스롯틀들에 대응하고, 각 부분에서 +델타 및 -델타 만큼 증분되고 감분되어 초기 스롯틀(T_0)+델타에 대한 자원 할당이 T_0-델타에 대해서보다 높게 될 것이다. 도 4a에서, 3개의 스레드(T_0, T_1, T_2)에 대해 스롯틀의 세트가 주어진 캐시 용량을 도시한다.

[0011] 도 4b에는, 일 실시예에 따라, 자원 할당 테스트가 예시되어 있다. 일 실시예에서, 도 4b는 캐시(400a)의 2개의 파트(401a, 405a)에 대한 2개의 성능 측정을 예시하고 있는데, 스롯틀(이 경우, T_1)은 블록(411a)에 저장된 데이터 클래스를 위한 제1 파티션(401a)에 대해서는 제1 방향(이 경우, 포지티브)이고, 스롯틀(T_1)은 블록(411a)에 저장된 데이터 클래스를 위한 제2 파티션(405a)에 대해서는 제2 방향(이 경우, 네거티브)으로 이동한다. 결과적인 성능 메트릭은, (성능 카운터의 이용을 포함해서) 각종 성능 측정 기술에 따라 결정된다(그 결과는 도 4b에 예시). 이들 자원 할당 테스트는, 프로그램에서의 상 변화, 스레드 이주 / 스왑핑 등에 따라 연속적으로 실행하여, 스롯틀을 연속적으로 최적화하고 그에 따라 특정 데이터 클래스 또는 클래스들에 할당되는 캐시 용량의 크기를 연속적으로 최적화할 수 있다. 일 실시예에서, 캐시(400a)에 저장된 각 데이터 클래스(410a-412a)는 예컨대 멀티 스레드형 프로그램에서 상이한 스레드에 대응한다.

[0012] 도 4c는 적어도 일 실시예에 따라 이용할 수 있는 로직을 예시한다. 도 4c의 로직(400c)은 프로세싱 코어(401c), 그 코어를 인터커넥트(예컨대, 링 인터커넥트)에 인터페이스하는 인터페이스 로직(405c), 그리고 총괄적인 최하위 레벨 캐시, L2 캐시일 수 있는 캐시(410c)를 예시한다. 일 실시예에서, GPA 모듈(415c)은 상기 캐시(410c) 상에 기술된 자원 할당 테스트들을 수행하고 연속적으로 테스트의 결과들에 따라 데이터 클래스들(및 대응 스레드들)에 캐시 용량을 할당하는 데에 이용된다. 일 실시예에서, 인터페이스 로직(405c)은 존재하지 않거나 코어 또는 다른 로직에 포함될 수도 있다.

[0013] 일 실시예에서, 캐시 상의 각 테스트의 결과는 GPA 모듈로 보내지고, GPA 모듈은 캐시 미스 레이트와 같은 글로벌 메트릭을 최적화하도록 캐시 할당을 조정하는 방법을 결정한다. 또한, 캐시는 특정 데이터 클래스용으로 캐시에서 추가의 캐시 블록 사이즈를 할당하려 할 때마다, GPA 모듈로부터 통지(신호 "통지")를 요청할 수 있고,

GPA 모듈은 우선 순위(신호 "우선 순위")로 응답할 것이다. 예컨대, GPA 모듈이 '고 우선 순위'를 지시하면 블록 사이즈를 증가시키고, GPA 모듈이 '저 우선 순위'를 지시하면 캐시가 블록 사이즈를 증가시키지 않거나 블록 사이즈를 약간 증가시키거나 그 블록 또는 파트를 다음에 대체하도록 대체 비트들을 설정할 수 있다. 적절한 자원 할당 정보를 지시 및 유지하기 위해서, GPA 모듈은 각 데이터 클래스에 대한 상태 비트들과 데이터 클래스들에 걸리는 글로벌 비트들을 포함할 수 있다. 예컨대, 데이터 클래스마다, GPA 모듈은,

[0014] 18 비트 -- 제1 성능 카운터에 대응

[0015] ("referencecounter")

[0016] 12 비트 -- 제2 성능 카운터에 대응

[0017] ("randomWalkCounter")

[0018] 8 비트 -- 스롯틀 컨트롤에 대응

[0019] 를 포함하는 총 38 상태 비트들을 저장할 수 있다. 또한, 최적 알고리즘을 파라미터화하기 위해서,

[0020] 4 비트 -- 각 데이터 클래스에 할당되는 상이한 캐시 영역들에 대응

[0021] ("numGradientRegionsLog")

[0022] 8 비트 -- 제1 스롯틀 델타에 대응

[0023] ("gradientDelta")

[0024] 8 비트 -- 제2스롯틀 델타에 대응

[0025] ("updateDelta")

[0026] 4 비트 -- GPA 모듈이 잠재적인 성능 이득에 응답할 감도에 대응

[0027] randomWalkThresholdMultiplier

[0028] 를 포함하는 4개의 글로벌 레지스터들(총 24 비트)을 이용할 수 있다.

[0029] 일 실시예에서, GPA 모듈은 이들 비트를 이용하여 그레이디언트 디센트(gradient descent)(또는 "힐 클라이밍(hill climbing)") 알고리즘의 변화를 이용한 공유 자원의 최적 할당을 찾아내는데, 여기서 데이터 클래스에 대한 해당 자원을 점진적으로 더 많이 그 다음에 점진적으로 더 적게 하는 것으로 하여 데이터 클래스마다 실험을 행한다. 그 다음에 GPA는 이들 비트를 이용하여 양 시나리오에 대한 "글로벌 굳니스(global goodness)" 메트릭을 측정하고 실험에 따라 해당 데이터 클래스에 공칭 정량의 자원을 할당한다. 일 실시예에서, GPA는 체르노프 한계(chernoff bounds)를 이용하여 자원의 분할을 조정하는 시기를 결정하고, 다른 실시예에서는, 다른 알고리즘을 이용하여 자원의 분할 조정 시기를 결정할 수 있다.

[0030] 일 실시예에서, 상기 GPA 모듈 비트들을 이용하여 상태 머신을 구현하고, 그 상태 천이는 다음의 코드 예에 따라 기술될 수 있다.

```
void GRADIENT_PARTITION_ALGORITHM_CLASS::Reference(
   bool _hitEqualsTrue,
   UINT32 _setIndex )
{
   UINT32 setIndex = SetIndexHash( _setIndex );
   UINT32 gradientRegionIndex = setIndex & ( ( 1 << numGradientRegionsLog ) - 1 );
   setIndex = setIndex >> numGradientRegionsLog;
   bool setMap;
   for( UINT32 i = gradientRegionIndex; i < numDataClasses; i = i + ( 1 <<
numGradientRegionsLog ) )
   {
      setMap = XOR( setIndex & ( ( i >> numGradientRegionsLog ) + 1 ) ); // +delta or -
delta
      data[ i ].referenceCounter++;
      data[ i ].randomWalk += ( _hitEqualsTrue == setMap ) ? 1 : -1;
      if( ( data[ i ].referenceCounter * randomWalkThresholdMultiplier ) <
         data[ i ].randomWalk * data[ i ].randomWalk )
      {  // did we cross the dynamic threshold?  If so, move the throttle in the winning
direction.
         data[ i ].throttle += ( data[ i ].randomWalk > 0 ) ? updateDelta : -updateDelta;
         data[ i ].throttle = ( data[ i ].throttle > throttleMask ) ? throttleMask :
data[ i ].throttle;
         data[ i ].throttle = ( data[ i ].throttle < 0 ) ? 0 : data[ i ].throttle;
         Reset( i ); // reset referenceCounter and randomWalk for the ith data class
      }
      else if( data[ i ].referenceCounter >= maxReferenceCount )
```

[0031]

```
        {  // telescoping, which is resetting while preserving some tendency from the
    previous experiment
            data[ i ].referenceCounter = data[ i ].referenceCounter >> 2;
            data[ i ].randomWalk = data[ i ].randomWalk >> 1;
        }
    }
}

bool GRADIENT_PARTITION_ALGORITHM_CLASS::InstallAdvice(
    UINT32 _dataClass,
    UINT32 _setIndex )
{
    UINT32 setIndex = SetIndexHash( _setIndex );
    UINT32 gradientRegionIndex = setIndex & ( ( 1 << numGradientRegionsLog ) - 1 );
    setIndex = setIndex >> numGradientRegionsLog;
    bool setMap = XOR( setIndex & ( ( _dataClass >> numGradientRegionsLog ) + 1 ) );
    INT32 randomThrottle = ( INT32 ) ( rand( ) & throttleMask );
    if( gradientRegionIndex == ( _dataClass & ( ( 1 << numGradientRegionsLog ) - 1 ) ) )
    {  // we are in a gradient region
        return randomThrottle <= ( data[ _dataClass ].throttle +
                        ( ( setMap == true ) ? gradientDelta : -gradientDelta ) );
    }
    else
    {
        return clock <= data[ _dataClass ].throttle;
    }
}
```

[0032]

[0033] 도 5는 실시예에서 이용하는 프로세서 또는 시스템 구성에 상관없이 본 발명의 적어도 일 실시예에 따라 이용 가능한 동작의 흐름도를 예시한다. 동작 501에서, 일 실시예에 따라 자원 할당 실험을 행하는 적어도 2개의 파트로 캐시를 분할한다. 동작 505에서, 최적화하는 특정 데이터 클래스들에 대응하는 블록들로 각 파트를 분할한다(예컨대 스트리밍 데이터 및 재사용 데이터). 일 실시예에서, 각 캐시 파티션 내의 각 데이터 클래스 파티션은 상이한 스레드에 대응한다. 동작 510에서, 전술한 실시예들에 따라 캐시 파티션마다 자원 할당 테스트를 행한다. 동작 515에서, 프로그램의 성능에서의 여러 지점에서, 캐시는 GPA 모듈에 정보를 전달하고, GPA 모듈은 연속적으로 성능을 측정하고, 성능 향상을 위해 캐시에 최적화 파라미터들을 제공한다.

[0034] 머신에 의해 판독될 때, 머신으로 하여금 본원에 기재된 기술을 실행하는 로직을 제조하게 하는, 프로세서 내의 각종 로직을 나타내는 머신 판독가능한 매체 상에 저장된 표현적인 데이터에 의해 적어도 일 실시예의 하나 이상의 다른 형태가 구현될 수 있다. "IP 코어들"로 알려진 그러한 표현들은 유형의 머신 판독가능한 매체("테이프") 상에 저장되고 여러 커스터머 또는 제조 설비에 공급되어 사실상 로직 또는 프로세서를 이루는 제조 머신

에 로드된다.

[0035] 마이크로 아키텍처 메모리 영역 액세스를 다이렉트하는 방법 및 장치가 개시되었다. 상기한 개시는 예시적인 것이지 제한적인 것이 아니다. 상기한 개시를 읽고 이해하다면 다른 많은 실시예들도 당업자에게 명백할 것이다. 따라서, 본 발명의 범위는 첨부한 청구 범위와 그것이 부여하는 균등물의 전체 범위에 따라 결정되어야 한다.

도면의 간단한 설명

[0036] 본 발명의 실시예들은 첨부하는 도면들 내의 도면에서 예시로서 도시되며, 한정하는 것은 아니며, 유사한 참조 번호들은 유사한 구성요소를 나타낸다.

[0037] 도 1은 본 발명의 적어도 하나의 실시예가 이용될 수 있는 마이크로프로세서의 블록도를 도시한다,

[0038] 도 2는 본 발명의 적어도 하나의 실시예가 이용될 수 있는 공유 버스 컴퓨터 시스템의 블록도를 도시한다,

[0039] 도 3은 본 발명의 적어도 하나의 실시예가 이용될 수 있는 점-대-점 상호접속 컴퓨터 시스템의 블록도를 도시한다,

[0040] 도 4는 본 발명의 적어도 하나의 실시예가 구현될 수 있는 로직의 블록도를 도시한다,

[0041] 도 5는 본 발명의 적어도 하나의 실시예를 수행하기 위해 이용될 수 있는 동작의 흐름도이다.

[0042] <도면의 주요 부분에 대한 부호의 설명>

[0043] 32, 34 : 메모리

[0044] 370, 380 : 프로세서

[0045] 374, 384 : 프로세싱 코어

[0046] 338 : 고성능 그래픽 회로

[0047] 390 : 칩셋

[0048] 318 : 버스 브리지

[0049] 314 : I/O 디바이스들

[0050] 324 : 오디오 I/O

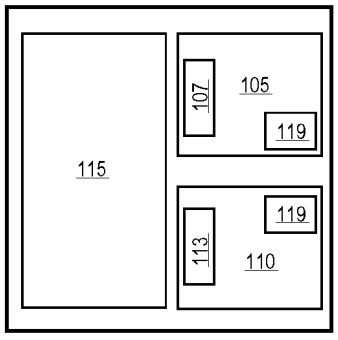
[0051] 322 : 키보드/마우스

[0052] 326 : 통신 디바이스들

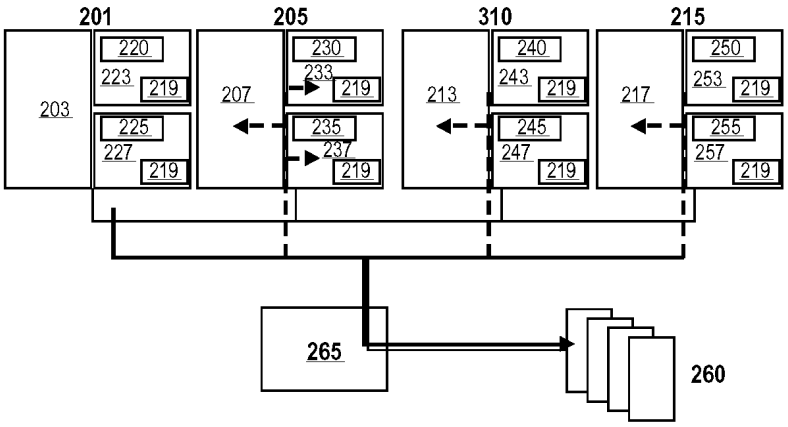
[0053] 328 : 데이터 스토리지

도면

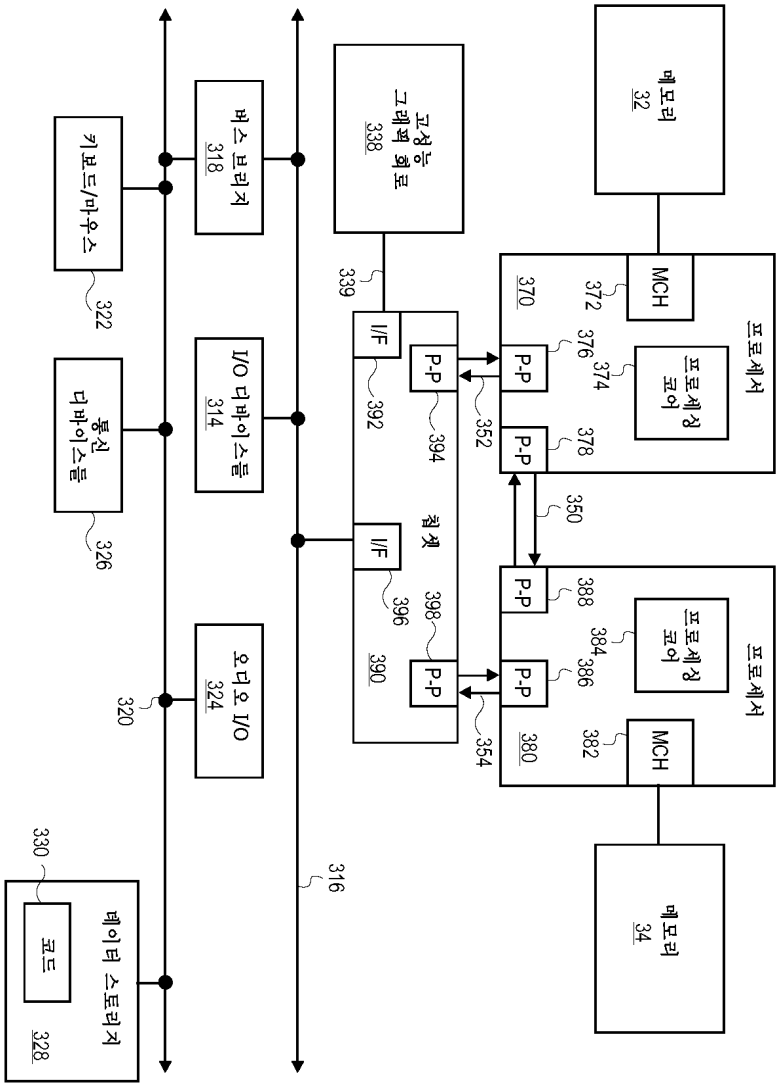
도면1



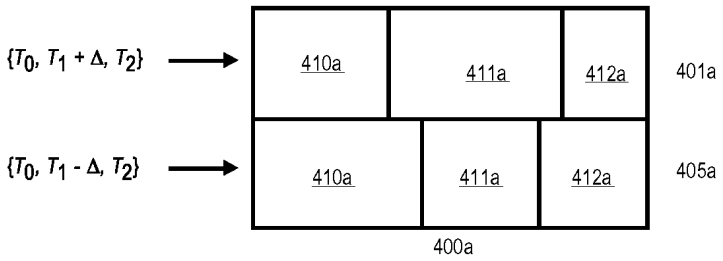
도면2


201
205
310
215
203
220
223
219
225
227
207
230
233
235
237
213
240
243
245
247
217
250
253
255
257
265
260

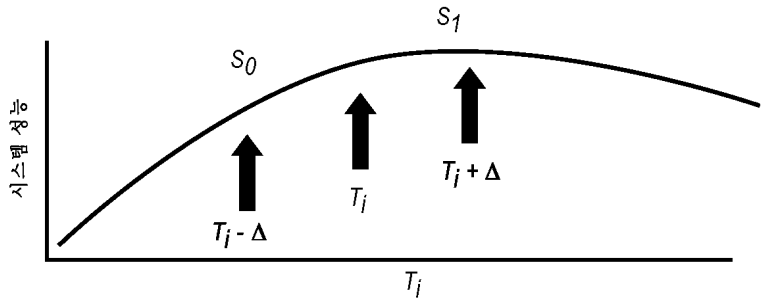
도면3


메모리
32
MCH
372
370
프로세서
프로세싱 코어
374
P-P
376
378
350
352
354
388
384
386
380
382
고성능 그래픽 회로
338
339
I/F
392
394
칩셋
396
398
390
버스 브리지
318
I/O 디바이스들
314
오디오 I/O
324
320
316
키보드/마우스
322
통신 디바이스들
326
데이터 스토리지
330
코드
328
메모리
34

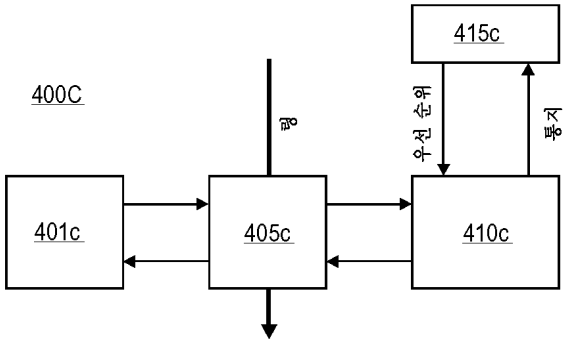
도면4a



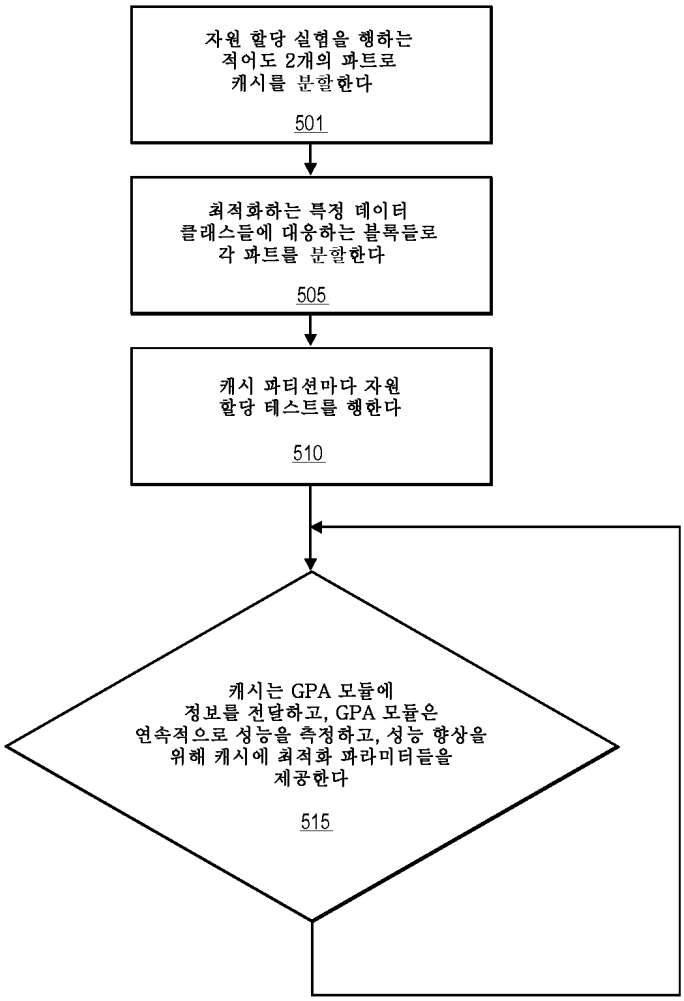
도면4b



도면4c



도면5


자원 할당 실험을 행하는
적어도 2개의 파트로
캐시를 분할한다
501
최적화하는 특정 데이터
클래스들에 대응하는 블록들로
각 파트를 분할한다
505
캐시 파티션마다 자원
할당 테스트를 행한다
510
캐시는 GPA 모듈에
정보를 전달하고, GPA 모듈은
연속적으로 성능을 측정하고, 성능 향상을
위해 캐시에 최적화 파라미터들을
제공한다
515