(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2016-0037828
(43) 공개일자 2016년04월06일

(51) 국제특허분류(Int. Cl.)
*G06F 12/08* (2016.01) *G06F 13/16* (2006.01)
*G06F 13/362* (2006.01) *G06F 15/16* (2006.01)
*G06F 15/173* (2006.01) *G06F 9/38* (2006.01)
*G06F 9/46* (2006.01) *G06F 9/48* (2006.01)
*H04L 12/863* (2013.01)
(52) CPC특허분류
*G06F 12/0875* (2013.01)
*G06F 12/0815* (2013.01)
(21) 출원번호 10-2015-7022210
(22) 출원일자(국제) 2013년06월26일
심사청구일자 없음
(85) 번역문제출일자 2015년08월17일
(86) 국제출원번호 PCT/US2013/048013
(87) 국제공개번호 WO 2014/113063
국제공개일자 2014년07월24일
(30) 우선권주장
61/753,892 2013년01월17일 미국(US)
*(뒷면에 계속)*

(71) 출원인
**엑소케츠 인코포레이티드**
미국 캘리포니아 산호세 사우스 알메이든 블러바드 1 #805 (우: 95113)
(72) 발명자
**다랄, 파린**
미국 94105 캘리포니아 샌프란시스코 미션 스트리트 201 스위트 1204
**벨에어, 스티븐**
미국 95062 캘리포니아 산타 크루즈 브랜시포르테 애비뉴 1119
(74) 대리인
**특허법인 남앤드남**

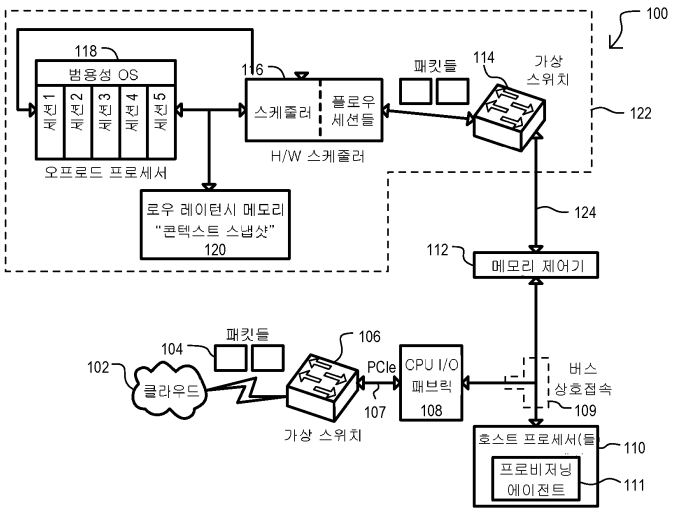
전체 청구항 수 : 총 31 항

(54) 발명의 명칭 **오프로드 프로세서들을 사용한 콘텍스트 스위치**

## (57) 요 약

메모리 버스에 연결된 복수의 오프로드 프로세서들 -각각의 오프로드 프로세서는 연관된 캐시 상태를 갖는 캐시를 포함함-, 오프로드 프로세서들에 커플링된 콘텍스트 메모리, 및 콘텍스트 메모리와 오프로드 프로세서들과 중 적어도 하나 사이에서의 캐시 상태의 전송을 지시하도록 구성된 스케줄링 회로를 포함하는, 콘텍스트 스위칭 캐시 시스템이 개시된다.

*대 표 도* - 도1a



(52) CPC특허분류
*G06F 13/16* (2013.01)
*G06F 13/362* (2013.01)
*G06F 15/161* (2013.01)
*G06F 15/17337* (2013.01)
*G06F 9/3877* (2013.01)
*G06F 9/461* (2013.01)
*G06F 9/4843* (2013.01)
*H04L 47/624* (2013.01)
*H04L 47/6295* (2013.01)

(30) 우선권주장
61/753,895 2013년01월17일 미국(US)
61/753,899 2013년01월17일 미국(US)
61/753,901 2013년01월17일 미국(US)
61/753,903 2013년01월17일 미국(US)
61/753,904 2013년01월17일 미국(US)
61/753,906 2013년01월17일 미국(US)
61/753,910 2013년01월17일 미국(US)
61/753,907 2013년01월17일 미국(US)

# 명 세 서

## 청구범위

### 청구항 1

콘텍스트 스위칭 캐시 시스템으로서,

메모리 버스에 연결된 복수의 오프로드 프로세서들 －각각의 오프로드 프로세서는 연관된 캐시 상태를 갖는 캐시를 포함함－,

상기 오프로드 프로세서들에 커플링된 콘텍스트 메모리, 및

상기 콘텍스트 메모리와 상기 오프로드 프로세서들과 중 적어도 하나 사이에서의 캐시 상태의 전송을 지시하도록 구성된 스케줄링 회로

를 포함하는, 콘텍스트 스위칭 캐시 시스템.

### 청구항 2

제 1 항에 있어서,

상기 복수의 오프로드 프로세서들은 자신의 캐시 상태들을 액세싱하기 위한 ACP(accelerator coherency port)를 갖는, 콘텍스트 스위칭 캐시 시스템.

### 청구항 3

제 1 항에 있어서,

상기 연관된 캐시 상태는, 오프로드 프로세서 레지스터들의 상태, 오프로드 프로세서에 의한 실행을 위한 명령들, 스택 포인터, 프로그램 카운터, 상기 오프로드 프로세서에 의한 실행을 위한 프리페칭된(prefetched) 명령들, 상기 오프로드 프로세서에 의한 사용을 위한 프리페칭된 데이터 및 상기 오프로드 프로세서의 상기 캐시에 기록된 데이터의 그룹으로부터 선택된 적어도 하나를 포함하는, 콘텍스트 스위칭 캐시 시스템.

### 청구항 4

제 1 항에 있어서,

적어도 하나 오프로드 프로세서는 OS(operating system)를 실행시키도록 구성되며; 상기 스케줄링 회로는, 프로세싱 세션에 대한 콘텍스트가 상기 오프로드 프로세서의 상기 캐시에 물리적으로 연속이게 설정하기 위해 상기 OS와 협력하도록 구성되는, 콘텍스트 스위칭 캐시 시스템.

### 청구항 5

제 1 항에 있어서,

적어도 하나 오프로드 프로세서는 OS(operating system)를 실행시키도록 구성되며; 상기 스케줄링 회로는, 프로세싱 세션에 대한 콘텍스트가 상기 오프로드 프로세서의 상기 캐시에 물리적으로 연속이게 설정하고, 상기 오프로드 프로세서의 상기 캐시에 프로세싱 세션 컬러, 크기 및 시작 물리적 어드레스를 설정하도록, 상기 OS와 협력하도록 구성되는, 콘텍스트 스위칭 캐시 시스템.

### 청구항 6

제 1 항에 있어서,

적어도 하나 오프로드 프로세서는 OS(operating system)를 실행시키도록 구성되며; 상기 스케줄링 회로는, 프로세싱 세션에 대한 콘텍스트가 상기 오프로드 프로세서의 상기 캐시에 물리적으로 연속이게 설정하고; 상기 오프로드 프로세서의 상기 캐시에 프로세싱 세션 컬러, 크기 및 시작 물리적 어드레스, 그리고 캐시에서 허용가능한 세션들의 수; 및 로케이션들의 수를 설정하도록, OS와 협력하도록 구성되며,

세션은 캐시에서의 주어진 컬러에 대해 발견될 수 있는, 콘텍스트 스위칭 캐시 시스템.

**청구항 7**

제 1 항에 있어서,

상기 스케줄링 회로는 하나의 오프로드 프로세서의 캐시 상태를 다른 오프로드 프로세서의 캐시로의 전송을 지시하도록 구성되는, 콘텍스트 스위칭 캐시 시스템.

**청구항 8**

제 1 항에 있어서,

상기 스케줄링 회로는, 상기 오프로드 프로세서들 중 하나와 연관된 제 1 세션을 중단하고, 상기 오프로드 프로세서의 캐시 상태를 저장하고, 그리고 제 2 큐로 홀딩된 네트워크 패킷들의 프로세싱을 개시함으로써, 제 1 큐로 네트워크 패킷들의 프로세싱을 우선순위화하도록 구성되는, 콘텍스트 스위칭 캐시 시스템.

**청구항 9**

제 1 항에 있어서,

상기 연관된 캐시 상태는, 오프로드 프로세서 레지스터들의 상태, 오프로드 프로세서에 의한 실행을 위한 명령들, 스택 포인터, 프로그램 카운터, 상기 오프로드 프로세서에 의한 실행을 위한 프리페치된 명령들, 상기 오프로드 프로세서에 의한 사용을 위한 프리페치된 데이터, 및 상기 오프로드 프로세서의 상기 캐시에 기록된 데이터의 그룹으로부터 선택된 적어도 하나를 포함하며,

적어도 하나 오프로드 프로세서는 OS(operating system)는 실행하도록 구성되며; 그리고

상기 스케줄링 회로는, 세션 콘텍스트가 상기 오프로드 프로세서의 상기 캐시에 물리적으로 연속이게 설정하기 위해, 상기 OS와 협력하도록 구성되는, 콘텍스트 스위칭 캐시 시스템.

**청구항 10**

제 1 항에 있어서,

상기 스케줄링 회로는, 메모리 버스를 통해, 콘텍스트 메모리와 상기 오프로드 프로세서들 중 적어도 하나 사이에서의 캐시 상태의 전송을 지시하도록 구성되는, 콘텍스트 스위칭 캐시 시스템.

**청구항 11**

제 1 항에 있어서,

상기 콘텍스트 메모리는 적어도 하나의 로우(low) 레이턴시 메모리 디바이스를 포함하는, 콘텍스트 스위칭 캐시 시스템.

**청구항 12**

메모리 버스를 통해 프로세싱을 위한 데이터를 수신하도록 커플링된 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법으로서,

스케줄링 회로의 동작에 의해, 복수의 오프로드 프로세서들 중 적어도 하나의 캐시로부터 콘텍스트 메모리로의 벌크 판독(bulk read)을 통해, 캐시 상태의 저장을 지시하는 단계 －상기 캐시 상태의 임의의 가상 및 물리적 메모리 로케이션들은 정렬됨－, 및

추후, 상기 스케줄링 회로의 동작에 의해, 프로세싱을 위해 상기 오프로드 프로세서들 중 적어도 하나로의 상기 캐시 상태의 전송을 지시하는 단계

를 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 13**

제 12 항에 있어서,

상기 벌크 판독은 ACP(Accelerator Coherency Port)를 통해 이루어지는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 14**

제 12 항에 있어서,

상기 연관된 캐시 상태는, 오프로드 프로세서 레지스터들의 상태, 상기 오프로드 프로세서에 의한 실행을 위한 명령들, 스택 포인터, 프로그램 카운터, 상기 오프로드 프로세서에 의한 실행을 위한 프리페칭된 명령들, 상기 오프로드 프로세서에 의한 사용을 위한 프리페칭된 데이터, 및 상기 오프로드 프로세서의 캐시에 기록된 데이터 중 적어도 하나를 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 15**

제 12 항에 있어서,

상기 캐시 상태는 세션 콘텍스트를 포함하고,

상기 방법은, 상기 스케줄링 회로 및 상기 오프로드 프로세서 상에서 실행되는 OS(operating system)의 협력에 의해, 상기 세션 콘텍스트가 상기 오프로드 프로세서의 상기 캐시에 물리적으로 연속이게 설정하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 16**

제 12 항에 있어서,

프로세싱 세션의 초기화시에, 세션 컬러, 세션 크기 및 상기 스케줄링 회로에 대한 프로세싱 세션을 위한 시작 물리적 캐시 어드레스를 통신하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 17**

제 12 항에 있어서,

복수의 프로세싱 세션들 각각에 대한 시작 어드레스, 오프로드 프로세서의 캐시에 허용가능한 세션들의 수, 및 로케이션들의 수를 결정하는 단계를 더 포함하며, 세션은 주어진 세션 컬러에 대해 발견될 수 있는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 18**

제 12 항에 있어서,

상기 오프로드 프로세서들 중 하나의 캐시 상태를 상기 오프로드 프로세서들 중 다른 하나의 캐시에 전송하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 19**

제 12 항에 있어서,

상기 오프로드 프로세서들 중 하나와 연관된 제 1 세션을 중단하고, 상기 오프로드 프로세서의 상기 캐시 상태를 저장하고, 그리고 제 2 큐로 홀딩된 네트워크 패킷들의 프로세싱을 개시함으로써, 상기 메모리 버스를 통해 수신된 제 1 큐로 네트워크 패킷들의 프로세싱을 우선순위화하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 20**

제 12 항에 있어서,

오프로드 프로세서에 의해 실행되는 프로세싱의 세션 콘텍스트를 상기 오프로드 프로세서의 상기 캐시에 물리적으로 연속이게 설정하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 21**

제 12 항에 있어서,

상기 캐시로부터 상기 콘텍스트 메모리로의 벌크 판독은 상기 메모리 버스를 통한 로우 레이턴시 메모리 디바이스로의 벌크 판독을 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 22**

복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법으로서,

소켓 연결된 메모리 버스(memory bus connected socket)를 통한 프로세싱을 위해 네트워크 패킷들을 수신하는 단계,

프로세싱을 위해 상기 네트워크 패킷들을 복수의 세션들로 구조화(organizing)하는 단계,

스케줄링 회로의 동작에 의해, 상기 오프로드 프로세서 중 적어도 하나의 캐시 상태를 콘텍스트 메모리로 판독함으로써 적어도 하나의 세션의 프로세싱을 서스펜딩(suspending)하는 단계 -가상 메모리 로케이션들 및 물리적 캐시 로케이션들은 정렬됨-, 및

추후, 상기 캐시 상태를 상기 스케줄링 회로의 동작에 의한 프로세싱을 위해 상기 오프로드 프로세서들 중 적어도 하나로 전송하게 지시하는 단계

를 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 23**

제 22 항에 있어서,

상기 벌크 판독은 ACP(accelerator coherency port)를 통해 이루어지는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 24**

제 22 항에 있어서,

상기 연관된 캐시 상태는, 상기 오프로드 프로세서 레지스터들의 상태, 상기 오프로드 프로세서에 의한 실행을 위한 명령들, 스택 포인터, 프로그램 카운터,상기  오프로드 프로세서에 의한 실행을 위한 프리페칭된 명령들, 상기 오프로드 프로세서에 의한 사용을 위한 프리페칭된 데이터, 및 상기 오프로드 프로세서의 캐시에 기록된 데이터 중 적어도 하나를 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 25**

제 22 항에 있어서,

상기 캐시 상태는 세션 콘텍스트를 포함하며; 그리고

상기 방법은, 상기 스케줄링 회로 및 상기 오프로드 프로세서 상에서 실행되는 OS(operating system)의 협력에 의해, 상기 세션 콘텍스트를 설정함으로써, 상기 세션 콘텍스트가 상기 오프로드 프로세서의 상기 캐시에 물리적으로 연속이게 설정하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 26**

제 22 항에 있어서,

프로세싱 세션의 초기화시에, 세션 컬러, 세션 크기 및 상기 스케줄링 회로에 대한 프로세싱 세션을 위한 시작 물리적 캐시 어드레스를 통신하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 27**

제 22 항에 있어서,

복수의 프로세싱 세션들 각각에 대한 시작 어드레스, 오프로드 프로세서의 캐시에 허용가능한 세션들의 수, 및 로케이션들의 수를 결정하는 단계를 더 포함하며, 세션은 주어진 세션 컬러에 대해 발견될 수 있는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 28**

제 22 항에 있어서,

상기 오프로드 프로세서들 중 하나의 캐시 상태를 상기 오프로드 프로세서들 중 다른 하나의 캐시에 전송하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 29**

제 22 항에 있어서,

상기 오프로드 프로세서들 중 하나와 연관된 제 1 세션을 중단하고, 상기 오프로드 프로세서의 상기 캐시 상태를 저장하고, 그리고 제 2 큐로 홀딩된 네트워크 패킷들의 프로세싱을 개시함으로써, 상기 메모리 버스를 통해 수신된 제 1 큐로 네트워크 패킷들의 프로세싱을 우선순위화하는 단계를 더 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 30**

제 22 항에 있어서,

상기 적어도 하나의 세션의 프로세싱을 서스펜딩하는 단계는, 세션 실행을 제어하기 위해 프리엠프션(preemption) 모드에서 동작하는 단계를 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

**청구항 31**

제 22 항에 있어서,

네트워크 패킷들을 수신하는 단계는, 상기 메모리 버스를 통해 DIMM(dual in line memory module) 호환가능 소켓을 통해 네트워크 패킷들을 수신하는 단계를 포함하는, 복수의 오프로드 프로세서들의 콘텍스트 스위칭을 위한 방법.

## *발명의 설명*

### *기 술 분 야*

[0001] 설명된 실시예들은 오프로드 프로세서들을 가진 메모리 버스 연결 모듈을 포함하는 컴퓨터 시스템들에 대한 결정론적 콘텍스트 스위칭에 관한 것이다.

### *배 경 기 술*

[0002] 콘텍스트 스위칭(때때로 프로세스 스위치 또는 태스크 스위치로서 지칭됨)은 하나의 프로세스 또는 스레드(thread)의 실행으로부터 다른 프로세스 또는 스레드의 실행으로 프로세서의 스위칭이다. 콘텍스트 스위치 동안 프로세스의 상태(콘텍스트)는, 실행이 동일한 추후 시점에서 재개될 수 있도록 메모리에 저장된다. 이것은 단일 프로세서를 공유하고 멀티태스킹 오퍼레이팅 시스템을 지원하기 위하여 다수의 프로세스들을 가능하게 한다. 보통, 프로세스는 병렬로 운용할 수 있고 어드레스 공간(즉, 메모리 위치들의 범위) 및 자신의 모 프로세스(parent process)들을 가진 다른 자원들을 공유할 수 있는 프로그램의 실행 또는 운용 예이다. 콘텍스트는 일반적으로 특정 시간에 프로세서의 레지스터들 및 프로그램 카운터의 콘텐츠를 포함한다. 오퍼레이팅 시스템은, 추후에 메모리로부터 제 2 프로세스의 콘텍스트를 리트리빙하고 이를 프로세스의 레지스터들에 복원하면서, 제 1 프로세스의 실행을 연기하고 메모리 내 해당 프로세스에 대한 콘텍스트를 저장할 수 있다. 제 2 프로세스를 종료하거나 연기한 후, 제 1 프로세스의 콘텍스트는 리로딩(reload)될 수 있고, 제 1 프로세스의 실행이 재개된다.

[0003] 그러나, 콘텍스트 스위칭은 계산적으로 집중적이다. 콘텍스트 스위치는 상당한 프로세서 시간을 요구할 수 있고, 이는 초당 몇십 또는 몇백 콘텍스트 스위치들의 각각에 대해 나노초들 정도일 수 있다. 현대 프로세서들이

몇백 또는 몇천의 별개의 프로세스들을 핸들링할 수 있기 때문에, 콘텍스트 스위칭에 전념된 시간은 프로세스 시간의 측면에서 시스템에 상당한 비용을 나타낸다. 개선된 콘텍스트 스위칭 방법들 및 시스템들은 전체 시스템 성능을 크게 개선하고 서버 또는 다른 데이터 프로세싱 시스템들에 대한 하드웨어 및 전력 요건들을 감소시킨다.

## 발명의 내용

[0004] 본 개시는 시스템에서 프로세서들의 콘텍스트 스위칭에 적당한 시스템들, 하드웨어 및 방법들의 실시예들을 설명한다. 실시예들은 다수의 오프로드 프로세서들을 포함할 수 있고, 오프로드 프로세서들 각각은 메모리 버스에 연결되고, 개별 오프로드 프로세서들은 각각 연관된 캐시 상태와 연관된 캐시를 가진다. 낮은 레이턴시 메모리는 메모리 버스를 통하여 다수의 오프로드 프로세서들에 연결될 수 있고, 스케줄 회로는 개별 오프로드 프로세서들 중 적어도 하나로부터 낮은 레이턴시 메모리로 캐시 상태의 저장을 지시하고, 메모리 버스를 통해 개별 오프로드 프로세서들 중 적어도 하나로 캐시 상태의 추후 전달을 지시하기 위하여 사용될 수 있다. ARM 아키텍처 프로세서들의 사용과 연관된 것들을 포함하는 특정 실시예들에서, 다수의 오프로드 프로세서들은 개선된 속도로 캐시 상태에 액세싱하기 위한 가속도기 코히어런시 포트를 가질 수 있다. 다른 실시예들에서, 공통 모듈은 오프로드 프로세서들, 낮은 레이턴시 메모리, 및 스케줄 회로를 지원할 수 있고, 외부 네트워크 패킷들에 대한 액세스는 듀얼 인라인 메모리 모듈(DIMM: dual in line memory module) 소켓들(이들로 제한되지 않음)을 포함하는 메모리 소켓 중재 연결을 통하여 제공된다.

[0005] 몇몇 실시예들에서 연관된 캐시 상태는: 레지스터 저장 영역에 저장된 프로세서 레지스터들의 상태, 실행되는 파이프라인의 명령들, 스택 포인터 및 프로그램 카운터, 세션에 의해 실행되도록 대기하는 프리페치(prefetch)된 명령들 및 데이터, 및 캐시에 기록된 데이터 중 적어도 하나를 포함한다. 시스템은 다수의 오프로드 프로세서들 중 적어도 하나에서 운용하는 오퍼레이팅 시스템(OS)을 더 포함할 수 있다. OS 및 스케줄 회로는 캐시에서 물리적으로 인접한 세션 콘텍스트들을 확립하기 위하여 협력할 수 있다. 세션 컬러, 사이즈 및 시작 물리적 어드레스는 세션 초기화시 스케줄러 회로에 그리고 각각의 세션의 시작 어드레스, 캐시에 허용 가능한 세션들의 수, 및 세션이 주어진 컬러에 대해 발견될 수 있는 위치들의 수를 결정하기 위하여 사용된 메모리 할당기에 통신될 수 있다.

[0006] 실시예들에 따라, 다수의 오프로드 프로세서들 중 하나에 의해 저장된 캐시 상태는 다른 오프로드 프로세서에 전달될 수 있다. 특정 애플리케이션들에서, 이것은 스케줄 회로로 하여금 오프로드 프로세서들 중 하나와 연관된 제 1 세션을 정지하고, 연관된 캐시 상태를 저장하고, 그리고 제 2 큐에 홀딩된 네트워크 패킷들의 프로세싱을 개시함으로써 메모리 버스를 통해 수신된 제 1 큐에서 네트워크 패킷들의 프로세싱을 우선순위화하게 한다.

[0007] 실시예는 또한 연관된 캐시 상태를 가진 연관된 캐시를 각각 가진 다수의 오프로드 프로세서들의 콘텍스트 스위칭, 및 메모리 버스를 통하여 다수의 오프로드 프로세서들에 연결된 낮은 레이턴시 메모리를 사용하기 위한 방법을 포함할 수 있다. 방법은 스케줄 회로를 사용하여 낮은 레이턴시 메모리에, 개별 오프로드 프로세서들 중 적어도 하나로부터의 벌크 판독(bulk reading)을 통해 캐시 상태의 저장을 지시하는 단계를 포함하고, 임의의 가상 및 물리적 메모리 위치들이 할당된다. 추후, 프로세싱을 위한 개별 오프로드 프로세서들 중 적어도 하나에 캐시 상태의 전달이 메모리 버스를 통해 지시되고, 전달은 스케줄 회로에 의해 제어된다. 이전에 설명된 구조 실시예들과 마찬가지로, 공통 모듈은 오프로드 프로세서들, 낮은 레이턴시 메모리, 및 스케줄링 회로를 지원할 수 있고, 외부 네트워크 패킷들에 대한 액세스는 DIMM 또는 다른 메모리 소켓 연결을 통해 제공된다.

## 도면의 간단한 설명

[0008] 도 1a은 실시예에 따른 콘텍스트 스위칭을 가진 시스템을 도시한다.

도 1b은 컬러링 없이 물리적으로 인덱싱된 캐시에서 페이지 충돌들을 도시하는 다이어그램이다.

도 1c는 가상 인덱싱된 캐시를 도시한다.

도 1d은 실시예에 따른 가상/물리적 할당 캐시를 도시한다.

도 2a 내지 도 2d은 다양한 실시예들에 따른 프로세서 모듈들을 도시한다.

도 2e는 종래의 듀얼-인-라인 메모리 모듈을 도시한다.

도 2f는 다른 실시예에 따른 시스템을 도시한다.

도 3은 일 실시예에 따른, 콘텍스트 스위칭 능력들을 가진 메모리 버스 연결 오프로드 프로세서 모듈을 가진 시스템을 도시한다.

도 4는 하나의 특정 실시예에 따른 콘텍스트 스위칭 동작들을 도시하는 흐름도이다.

## *발명을 실시하기 위한 구체적인 내용*

[0009] 다양한 실시예들은 이제 다수의 도면들을 참조하여 상세히 설명될 것이다. 실시예들은 시스템 메모리 버스에 연결된 오프로드 프로세서들에서 콘텍스트들을 스위칭하기 위한 모듈들, 시스템들 및 방법들을 도시한다. 그런 오프로드 프로세서들은 게다가 시스템 메모리 버스에 연결된 임의의 호스트 프로세서들일 수 있고, 임의의 호스트 프로세서들에 무관하게 시스템 메모리 버스를 통해 전달된 데이터 상에서 동작할 수 있다. 특정 실시예들에서, 오프로드 프로세서들은 빠른 저장 및 빠른 콘텍스트 스위칭을 위한 콘텍스트 데이터의 리트리벌을 가능하게 할 수 있는 낮은 레이턴시 메모리에 액세스를 가질 수 있다. 매우 특정한 실시예들에서, 프로세싱 모듈들은 인-라인 메모리 모듈들(예를 들어, 듀얼 인 라인 메모리 모듈(DIMM)들)을 시스템 메모리 버스에 연결하기 위한 물리적 슬롯들을 거주시킬 수 있다.

[0010] 도 1a은 일 실시예에 따른 시스템(100)을 도시한다. 시스템(100)은 하나 또는 초과의 오프로드 프로세서들(118), 스케줄러(116), 및 콘텍스트 메모리(120)를 포함할 수 있다. 오프로드 프로세서(118)는, 캐시 메모리와 함께 동작하는 하나 또는 그 초과의 프로세서 코어들을 포함할 수 있다. 콘텍스트 스위치 동작시, 오프로드 프로세서(118)의 제 1 프로세싱 태스크의 콘텍스트는 콘텍스트 메모리(120) 내에 저장될 수 있으며, 이후 오프로드 프로세서(118)는 새로운 프로세싱 태스크를 착수할(undertake) 수 있다. 이후, 저장된 콘텍스트가 콘텍스트 메모리(120)로부터 오프로드 프로세서(118)로 복원될 수 있으며, 오프로드 프로세서(118)는 제 1 프로세싱 태스크를 재개할 수 있다. 특정 실시예들에서, 콘텍스트 데이터의 저장 및 복원은, 콘텍스트 메모리(120)와 오프로드 프로세서(118)의 캐시 간의 데이터의 전송을 포함할 수 있다.

[0011] 스케줄러(116)는 수신된 프로세싱 요청들에 기초하여 오프로드 프로세서들(118)의 콘텍스트 스위치들을 조정할(coordinate) 수 있다. 이에 따라, 스케줄러(116)는, 오프로드 프로세서들(118)의 상태 뿐만 아니라 오프로드 프로세서들(118)에 대한 콘텍스트 데이터의 위치를 통지받을 수 있거나, 또는 오프로드 프로세서들(118)의 상태 뿐만 아니라 오프로드 프로세서들(118)에 대한 콘텍스트 데이터의 위치를 액세스할 수 있다. 콘텍스트 데이터 위치들은, 프로세서 캐시 내에서의 위치들 뿐만 아니라, 콘텍스트 메모리(120) 내에서의 위치들을 포함할 수 있다. 스케줄러(116)는 또한, 오프로드 프로세서들(118)의 상태를 따를 수 있거나, 또는 그러한 상태로 업데이트될 수 있다.

[0012] 상기로부터 이해되는 바와 같이, 콘텍스트 메모리(120)는 이후의 검색(retrieval)을 위해 오프로드 프로세서들(118)의 콘텍스트 데이터를 저장할 수 있다. 콘텍스트 메모리(120)는 오프로드 프로세서들의 캐시 메모리들과 분리될 수 있다. 몇몇 실시예들에서, 콘텍스트 메모리(120)는, 신속한 콘텍스트 저장 및 검색을 가능하게 하기 위해, 시스템 내의 다른 메모리와 비교하여 로우 레이턴시 메모리일 수 있다. 몇몇 실시예들에서, 콘텍스트 메모리(120)는 콘텍스트 데이터 이외의 데이터를 저장할 수 있다.

[0013] 도시된 특정 실시예에서, 오프로드 프로세서들(118), 스케줄러(116) 및 콘텍스트 메모리(120)는, 메모리 버스(124)에 연결된 모듈(122)의 일부일 수 있다. 오프로드 프로세서들(118)에 의해 실행하기 위한 데이터 및 프로세싱 태스크들은 메모리 버스(124)를 통해 수신될 수 있다. 몇몇 실시예들에서, 오프로드 프로세서들(118)과 콘텍스트 메모리(120) 간의 콘텍스트 데이터의 전송들은 메모리 버스(124)를 통해 일어날 수 있다. 하지만, 다른 실시예들에서, 그러한 전송들은 모듈(122) 상의 상이한 데이터 버스를 통해 일어날 수 있다.

[0014] 도 1a을 여전히 참조하면, 도시된 매우 특정한 실시예에서, 방법은 추가적으로, 제 2 스위치(114), 메모리 제어기(112), 호스트 프로세서(110), 입/출력(I/O) 패브릭(108) 및 제 1 스위치(106)를 포함할 수 있다. 제 2 스위치(114)는 모듈(122) 상에 포함될 수 있다. 도 1a의 특정 시스템(100)은 네트워크 패킷 프로세싱 스케쥴링 및 트래픽 관리에 관한 것이지만, 다른 실시예들은, 다른 타입들의 프로세싱 태스크들에 관한, 본원에서 설명되거나 동등물들인, 콘텍스트 스위칭 동작들을 포함할 수 있음이 이해된다.

[0015] 도 1a의 특정 실시예에서, 제 1 스위치(106)는 데이터 소스(102)로부터 데이터 패킷들(104)을 수신하고 그리고/또는 데이터 소스(102)에 데이터 패킷들(104)을 송신할 수 있다. 데이터 소스(102)는, 인터넷, 네트워크 클라우드, 인터- 또는 인트라- 데이터 센터 네트워크들, 클러스터 컴퓨터들, 랙 시스템들(rack systems), 복수의 또는 개별 서버들 또는 개인용 컴퓨터들 등을 포함하는, 패킷 데이터의 임의의 적합한 소스일 수 있다. 데이터는 패킷 또는 스위치 기반일 수 있지만, 특정 실시예들에서, 넌-패킷 데이터는 일반적으로, 취급(handling)의 용이

성을 위해 패킷들로 변환 또는 캡슐화된다. 데이터 패킷들은 전형적으로, 전송 프로토콜 넘버, 소스 및 목적지 포트 넘버들, 또는 소스 및 목적지 (인터넷 프로토콜) IP 어드레스들을 포함하는 특정의 특성들을 갖는다. 데이터 패킷들은 추가적으로, 패킷 분류 및 관리를 돕는 연관된 메타데이터를 가질 수 있다.

[0016] 스위치(106)는 가상 스위치(I/O 디바이스)일 수 있다. 스위치(106)는, 비제한적으로, PCI(peripheral component interconnect) 또는 PCIe(PCI express) 버스(107)를 통해 호스트 마더보드와 연결되는 PCIe 디바이스들 및/또는 PCI와 호환가능한 디바이스들을 포함할 수 있다. 스위치(106)는 네트워크 인터페이스 제어기(NIC), 호스트 버스 어댑터, 융합 네트워크 어댑터(converged network adapter), 또는 스위치형(switched) 또는 비동기 전송 모드(ATM) 네트워크 인터페이스를 포함할 수 있다. 몇몇 실시예들에서, 스위치(106)는, 단일 네트워크 I/O 디바이스가 복수의 디바이스들로서 보이도록 하기 위해, SR-IOV(single root I/O virtualization) 인터페이스와 같은 IO 가상화 방식들(IO virtualization schemes)을 이용할 수 있다. SR-IOV는, 물리적 제어와 가상 함수들(virtual functions) 모두를 제공함으로써, 다양한 PCIe 하드웨어 기능들(functions) 사이에서의 자원들에 대한 개별적인(separate) 액세스를 허용한다. 특정 실시예들에서, 스위치(106)는, 제어 평면으로부터 추상화(abstract)하기 위한, 오픈플로우(OpenFlow) 또는 유사한 소프트웨어 정의되는(software defined) 네트워킹을 지원할 수 있다. 제 1 가상 스위치의 제어 평면은, 라우트(route) 결정, 타겟 노드 식별 등과 같은 기능들(functions)을 수행한다.

[0017] 스위치(106)는, 네트워크 패킷들을 검사하고, 그 제어 평면을 이용하여 네트워크 패킷들에 대한 적절한 출력 포트들을 생성할 수 있다. 네트워크 패킷들 또는 그러한 네트워크 패킷들과 연관된 데이터 흐름들에 대한 라우트 계산에 기초하여, 스위치(106)의 포워딩 평면(forwarding plane)은 출력 인터페이스에 패킷들을 전송할 수 있다. 스위치의 출력 인터페이스는 IO 버스와 연결될 수 있으며, 그리고 특정 실시예들에서, 스위치(106)는 메모리 판독 또는 기록 동작(다이렉트 메모리 액세스 동작)을 위해 메모리 버스 상호접속(109)에 네트워크 패킷들을 직접적으로 (또는 I/O 패브릭(108)을 통해 간접적으로) 전송할 수 있는 성능을 가질 수 있다. 기능적으로, 특정 애플리케이션들에 대해, 네트워크 패킷들은 제어 평면 기능성(control plane functionality)에 기초하여 특정 메모리 위치들로의 전송을 위해 할당될 수 있다.

[0018] IO 패브릭(108)과 메모리 버스 상호접속(109)에 연결되어 있는 스위치(106)는 호스트 프로세서(들)(110)에 또한 연결될 수 있다. 호스트 프로세서(들)(110)은, 프로비저닝 에이전트(provisioning agent)(111)를 포함하는 컴퓨테이션 서비스들(computational services)을 제공할 수 있는 하나 또는 그 초과의 호스트 프로세서들을 포함할 수 있다. 프로비저닝 에이전트(111)는, 호스트 프로세서(들)(110) 상에서 작동하는 사용자 코드 또는 운영 체제의 일부일 수 있다. 프로비저닝 에이전트(111)는 전형적으로, 시스템(100)에 의해 제공되는 가상 함수 드라이버들(virtual function drivers)을 초기화하고 이들과 상호작용한다. 가상 함수 드라이버는, 다이렉트 메모리 어드레싱(DMA)이 요구되는 메모리 공간의 가상 어드레스(virtual address)를 제공하는 것을 담당할 수 있다. 각각의 디바이스 드라버에는, 물리적 어드레스들에 대해 맵핑되는 가상 어드레스들이 할당될 수 있다. 디바이스 모델은, 생성될 수 있는 복수의 가상 함수들(VF) 각각을 인식하기 위해 호스트 프로세서(110)에 대한 물리적 디바이스의 에뮬레이션(emulation)을 생성하는 데에 이용될 수 있다. 이러한 디바이스 모델은, VF 드라이버들(가상 IO 디바이스와 상호작용하는 드라이버)에 이들이 물리적 디바이스와 상호작용하고 있다는 인상(impression)을 주기 위해 다수회 복제될 수 있다. 예를 들어, 특정 디바이스 모델을 이용하여, VF 드라이버가 연결하고자 작동할 수 있는 네트워크 어댑터를 에뮬레이팅할 수 있다. 디바이스 모델 및 VF 드라이버는, 특권(privileged) 모드 또는 비특권(non-privileged) 모드에서 작동할 수 있다. 디바이스 모델 및 VF 드라이버에 대응하는 코드를 어떤 디바이스가 호스팅/작동시키는 지에 대해서는 어떠한 제한도 없다. 하지만, 코드는, 생성될 상기 I/O 인터페이스의 복수의 카피들을 가능하게 하기 위해 디바이스 모델 및 VF 드라이버의 복수의 카피들을 생성할 수 있는 성능을 가질 수 있다. 특정 실시예들에서, 운영 체제는, VF 드라이버들에 의해 지원되는 애플리케이션들에 대해, 정의된 물리적 어드레스 공간(defined physical address space)을 또한 생성할 수 있다. 추가적으로, 호스트 운영 체제는 애플리케이션 또는 프로비저닝 에이전트에 가상 메모리 어드레스 공간을 할당할 수 있다. 프로비저닝 에이전트(111)는, 이용가능한 물리적 어드레스 공간의 서브세트와 가상 어드레스들 간에 맵핑을 생성하기 위해, 호스트 운영 체제와 중개할(broker) 수 있다. 프로비저닝 에이전트(111)는, 각각의 VF 드라이버를 생성하고 이를 정의된 가상 어드레스 공간에 할당하는 것을 담당할 수 있다.

[0019] 제 2 가상 스위치(114)는 또한, 메모리 버스(109)를 사용하여 메모리 제어기(112)에 연결될 수 있다. 제 2 가상 스위치(114)는, 오프로드 프로세서들(118)로부터의 그리고 그로의 메모리 버스(109)로부터 유래하는 트래픽을 수신하고 스위칭한다. 트래픽은, 오프로드 프로세서들(118)에 의해 지원되는 프로세싱으로, 프로비저닝 에이전트(111)에 의해 생성되고 할당된, 가상 디바이스들로의 데이터 플로우들을 포함할 수 있지만, 이에 제한되

지는 않는다. 제 2 가상 스위치의 포워딩 플레인은, 패킷들을, 메모리 버스(109)로부터 오프로드 프로세서들(118)로, 또는 다시 오프로드 프로세서들(118)로부터 메모리 버스(109) 상으로 운반한다. 특정 애플리케이션들에 대해, 설명된 시스템 아키텍처는, 호스트 프로세서(110)에 대한 최소의 인터럽션들로, 또는 호스트 프로세서(110)에 대한 인터럽션들 없이, 오프로드 프로세서들(118)로의 네트워크 패킷들의 비교적 직접적인 통신을 허용한다. 제 2 가상 스위치(114)는, 정의된 아비트레이션(arbitration) 및 스케줄링 스킴(scheme)에 기초하여, 상이한 하드웨어 스케줄러들로의 분배 전에, 패킷들을 수신하고 이들을 분류하는 것이 가능할 수 있다. 하드웨어 스케줄러(116)는, 하나 또는 그 초과의 개별적인 세션에서의 프로세싱을 위해 스케줄링된 플로우 세션들에 할당될 수 있는 패킷들을 수신한다.

[0020] 스케줄러(116)는, 컨텍스트들의 스위칭을 포함하는, 오프로드 프로세서들(118)에 의한 실행을 위한 프로세싱 태스크들을 제어할 수 있다. 몇몇 실시예들에서, 메모리 버스(124)를 통해 수신된 데이터 내에 포함된 메타데이터(또는 그러한 데이터로부터 도출된 메타데이터)는, 오프로드 프로세서들(118)의 태스크들을 스케줄링/스위칭하기 위해 스케줄러(116)에 의해 사용될 수 있다. 그러나, 메모리 버스 수신된 커맨드들 또는 플래그들을 통한 스케줄러의 커맨드 기반 제어가 또한 고려된다.

[0021] 도 1a의 특정한 실시예에서, 스케줄러(116)는 인커밍 패킷들의 트래픽 관리를 구현하기 위해 채용될 수 있다. 특정한 소켓에 대한 특정 애플리케이션 또는 플로잉에 관계된, 특정한 트래픽 클래스에 관한, 특정한 소스로부터의 패킷들은, 세션 플로우의 부분이라고 지칭되고, 세션 메타데이터를 사용하여 분류된다. 세션 메타데이터는 종종, 패킷들이 우선순위화되는 기준으로서 서빙하고, 따라서, 인커밍 패킷들은 이들의 세션 메타데이터에 기초하여 재순서화될 수 있다. 패킷들의 이러한 재순서화는 하나 또는 그 초과의 버퍼들에서 발생할 수 있고, 이들 플로우들의 트래픽 형상을 변경할 수 있다. 세션 메타데이터에 기초하여 재순서화된 세션의 패킷들은, 아비트레이션 회로(미도시)를 사용하여, 출력 포트들로 아비트레이팅된(arbitrated) 특정 트래픽 관리된 큐들로 전송될 수 있다. 아비트레이션 회로는, 직접적으로, 다운스트림 패킷 프로세싱/터미네이팅 리소스로 이들 패킷 플로우들을 공급할 수 있다. 특정한 실시예들은, 스레드 및 큐 관리의 통합을 제공하여, 위의 상기 스레드들을 통하는 네트워크 데이터의 터미네이션을 핸들링하는 다운스트림 리소스들의 스루풋을 향상시킨다.

[0022] 여전히 도 1a을 참조하면, 스케줄러(116)에 도달하는 데이터는 또한, 오프로드 프로세서들(118)에서 터미네이팅되기를 대기하는 패킷 데이터일 수 있거나, 또는 그것은 프로세싱, 변경, 또는 스위칭되기를 대기하는 패킷 데이터일 수 있다. 스케줄러(116)는, 패킷 데이터의 검토에 기초하여, 대응하는 애플리케이션 세션들로 인커밍 패킷들을 세그리게이팅하는 것에 대해 책임이 있을 수 있다. 스케줄러(116)는, 패킷 검사 및 관련된 패킷 특성들의 식별을 위한 회로들을 가질 수 있다. 몇몇 실시예들에서, 스케줄러(116)는, 네트워크 스택 프로세싱으로부터 초래되는 오버헤드로부터 오프로드 프로세서들(118)을 자유롭게 하기 위해, 네트워크 스택의 부분을 오프로딩할 수 있다. 특정한 실시예들에서, 스케줄러(116)는, TCP/운반 오프로드, 인크립션/디크립션 오프로드, 세그먼테이션, 및 리어셈블리 중 임의의 것을 수행할 수 있고, 따라서, 오프로드 프로세서들이 직접적으로 네트워크 패킷들의 페이로드들 상에서 동작하게 허용할 수 있다.

[0023] 스케줄러(116)는 추가로, 특정한 트래픽 관리 큐로 세션에 속하는 패킷들을 전달하는 능력을 가질 수 있다. 스케줄러(116)는, 범용 OS로의 다수의 그러한 세션들 각각의 스케줄링을 제어할 수 있다. 범용 OS를 포함하는, 스테이지들의 파이프라인에 걸친 세션들의 스티키니스(stickiness)는, 스테이지들 각각에서 수행되는 동작들을 최적화하는 스케줄러(116)에 의해 지원될 수 있다. 그러한 동작들의 특정한 실시예들이 아래에서 더 상세히 설명된다.

[0024] 스케줄러(116)가 임의의 적합한 형태를 갖지만, 스케줄러로서 전부 또는 부분적으로 사용될 수 있는 스케줄링 회로가, 2010년 7월 20일자로 달랄(Dalal)에 의해 발행된 미국 특허 번호 제 7,760,715 호(이하, '715 특허)에서 보여지고, 인용에 의해 본원에 포함된다. '751 특허는 다운스트림 실행 리소스들을 고려하는 스케줄링 회로를 개시한다. 이들 큐들 각각에서의 세션 플로우들은 출력 포트를 통해 다운스트림 네트워크 엘리먼트로 전송된다.

[0025] 스케줄러는, 이용가능한 출력 포트들로의 다수의 트래픽 관리 출력 큐들의 액세스를 조정(mediate)하기 위해 아비트레이션 회로를 채용할 수 있다. 출력 포트들 각각은 패킷 버퍼를 통해 오프로드 프로세서 코어들 중 하나에 연결될 수 있다. 패킷 버퍼는 헤더 풀 및 패킷 바디 풀을 더 포함할 수 있다. 헤더 풀은 오프로드 프로세서들(118)에 의해 프로세싱될 패킷들의 헤더만을 포함할 수 있다. 때때로, 프로세싱될 패킷의 크기가 충분히 작은 경우에, 헤더 풀은 전체 패킷을 포함할 수 있다. 패킷들은, 오프로드 프로세서(118)에서 수행되는 동작의 성질에 따라, 헤더 풀 또는 패킷 바디 풀로 전달될 수 있다. 패킷 프로세싱, 오버레이, 분석, 필터링, 및 그러

한 다른 애플리케이션들에 대해, 오프로드 프로세서들(118)에 패킷 헤더만을 전달하는 것은 적절할 수 있다. 이러한 경우에, 패킷 헤더의 핸들링에 따라, 패킷 바디는, 패킷 헤더와 함께 소잉될(sewn) 수 있고 이그레스 인터페이스를 통해 전달될 수 있거나 또는 드롭될(dropped) 수 있다. 패킷들의 터미네이션을 요구하는 애플리케이션들에 대해, 패킷의 전체 바디가 전달될 수 있다. 따라서, 오프로드 프로세서들은 패킷들을 수신할 수 있고, 이들에 대한 적합한 애플리케이션 세션을 실행할 수 있다.

[0026] 스케줄러(116)는 오프로드 프로세서들(118) 상의 상이한 세션들을 스케줄링할 수 있어서, 그러한 세션들을 조정하도록 작용하여, 컨텍스트 스위치들 동안에 오버헤드를 감소시킬 수 있다. 스케줄러(116)는, 라인 레이트 속도들로 아웃고잉 큐들 또는 세션 플로우들 사이 뿐만 아니라, 매우 높은 속도들로 터미네이팅된 세션들 사이를 아비트레이팅할 수 있다. 스케줄러(116)는 오프로드 프로세서들(118) 상의 세션들의 큐잉을 관리할 수 있고, OS 상의 새로운 애플리케이션 세션들을 인보킹하는 것에 대해 책임이 있을 수 있다. 스케줄러(116)는, 트래픽에 기초하여 새로운 세션에 대한 패킷들이 이용가능한 것을 OS에게 표시할 수 있다.

[0027] 스케줄러(116)에는 또한, 오프로드 프로세서들(118)의 실행 리소스들의 상태, 실행 리소스 상에서 실행되는 현재 세션, 및 그것에 할당된 메모리 공간, 뿐만 아니라, 오프로드 프로세서 캐시에서의 세션 컨텍스트의 위치가 통지될 수 있다. 따라서, 스케줄러(116)는 트래픽 관리 및 아비트레이션 판정들을 수행하기 위해 실행 리소스의 상태를 사용할 수 있다.

[0028] 도시된 실시예에서, 스케줄러(116)는 인커밍 패킷들의 트래픽 관리와 운영 시스템 상의 스레드 관리의 통합을 제공할 수 있다. 그것은, 오프로드 프로세서들(118) 상의 프로세싱 엔티티들 및 트래픽 관리 큐들을 포함하는 컴포넌트들의 스펙트럼에 걸친 세션 플로우들의 지속성(persistence)을 유발할 수 있다. 오프로드 프로세서(118) 상에서 실행하는 OS는, 그것이 현재 핸들링하고 있는 특정한 큐에 메모리 및 프로세서 사이클들과 같은 실행 리소스들을 할당할 수 있다. OS는 추가로, 그 특정한 큐에 대해 스레드들의 그룹 또는 스레드를 할당할 수 있고, 그에 따라, 그것은 개별적인 엔티티로서 범용 프로세싱 엘리먼트에 의해 분명하게 핸들링될 수 있다. 다수의 세션들이 범용(GP) 프로세싱 리소스(예컨대, 오프로드 프로세서 리소스) 상에서 실행되게 하여, 스케줄러(116) 상의 큐에 존재하는 특정한 세션 플로우로부터의 각각의 핸들링 데이터는 GP 프로세싱 리소스 및 스케줄러(116)를 단단히 통합시킬 수 있다. 이는, GP 프로세싱 리소스 및 스케줄러(116) 및 트래픽 관리에 걸친 세션 정보내에서 지속성의 엘리먼트를 야기할 수 있다.

[0029] 일부 실시예들에서, 오프로드 프로세서(118) OS는, 리소스들 간의 콘텍스트 스위치와 연관된 패널티 및 오버헤드를 감소시키기 위해 이전의 OS로부터 수정될 수 있다. 이것은, 큐들 간의 심리스 스위칭, 및 결과적으로 실행 리소스에 의한 상이한 세션들로서 자신들의 실행을 수행하기 위해 하드웨어 스케줄러에 의해 추가로 이용될 수 있다.

[0030] 특정 실시예들에 따라, 스케줄러(116)는 인커밍 패킷들의 트래픽 관리를 구현할 수 있다. 특정 트래픽 클래스에 관련되거나, 특정 애플리케이션에 속하거나, 특정 소켓으로 흐르는, 특정 소스로부터의 패킷들은 세션 플로우의 부분으로 지칭되고, 세션 메타데이터를 사용하여 분류된다. 세션 메타데이터는 패킷들을 우선순위화하는 기준으로서 기능할 수 있고, 이로써 인커밍 패킷들은 자신들의 세션 메타데이터에 기초하여 재정렬된다. 이러한 패킷들의 재정렬은 하나 또는 그 초과의 버퍼들에서 발생할 수 있고, 이러한 플로우들의 트래픽 형상을 수정할 수 있다. 세션 메타데이터에 기초하여 재정렬된 세션의 패킷들은, 중재 회로를 사용하여 출력 포트들로 중재되는 특정 트래픽 관리 큐들로 전송될 수 있다. 중재 회로는 이러한 패킷 플로우들을 다운스트림 패킷 프로세싱 및/또는 종단 리소스(예를 들면, 오프로드 프로세서)로 직접적으로 피딩할 수 있다. 특정 실시예들은, 스레드들 위를 통해 네트워크 데이터의 종료를 처리하는 다운스트림 리소스들의 스루풋을 개선하기 위해 스레드 및 큐 관리의 통합을 제공한다.

[0031] 인커밍 네트워크 패킷들(및 플로우들)의 트래픽 관리, 중재 및 스케줄링을 수행하는 이외에, 스케줄러(116)는 OS 오프로드 프로세서(118) 상의 종료된 세션들 사이의 최소 오버헤드 콘텍스트 스위칭을 인에이블하는 것을 담당할 수 있다. 오프로드 프로세서(118)의 세션들 상의 다수의 세션들 사이의 스위칭은 매우 높은 속도들에서 다수의 세션들을 종료하는 것이 가능하게 될 수 있다. 도시된 실시예에서, 빠른 콘텍스트 스위칭은 콘텍스트 모듈(120)의 동작에 의해 발생할 수 있다. 특정 실시예들에서, 콘텍스트 메모리(120)는 시스템(100)에서 효율적이고, 로우 레이턴시 콘텍스트 서비스들을 제공할 수 있다.

[0032] 도시된 특정 실시예에서, 패킷들은 제 2 스위치(114)의 동작에 의해 스케줄러(116)로 전송될 수 있다. 스케줄러(116)는 오프로드 프로세서들(118) 상의 세션과 새로운 세션 사이의 스위칭뿐만 아니라 콘텍스트 메모리(120) 내의 콘텍스트의 저장을 개시하는 것 둘 모두를 담당할 수 있다. 세션의 콘텍스트는 레지스터 저장 영역에 저

장된 프로세서 레지스터들의 상태, 실행되는 파이프라인 내의 명령들, 스택 포인터 및 프로그램 카운터, 세션에 의해 실행되기를 대기하는 프리페치된 명령들 및 데이터, 최근에 캐시에 기록된 데이터 및 오프로드 프로세서(118) 상에서 실행되는 세션을 식별할 수 있는 임의의 다른 관련 정보를 포함할 수 있지만, 이에 제한되지 않는다. 특정 실시예들에서, 세션 콘텍스트는 세션 id, 캐시 내의 세션 인덱스 및 시작하는 물리적 어드레스의 조합을 사용하여 식별될 수 있다.

[0033] 도 1d에 관련하여 더 상세히 설명될 바와 같이, 가상 메모리 내의 세션의 연속적인 페이지들이 오프로드 프로세서(118)의 캐시에서 물리적으로 연속적이도록, 변환 방식이 사용될 수 있다. 캐시 내의 세션의 이러한 연속적인 특성은 콘텍스트 메모리(120) 내의 저장을 위한 '콘텍스트 스냅샷'으로의 세션 콘텍스트의 벌크 판독 출력을 허용할 수 있고, 운영 시스템(OS)이 프로세서 리소스들을 다시 세션으로 스위칭할 때, 세션 콘텍스트가 콘텍스트 메모리로부터 리트리브될 수 있다. 콘텍스트 메모리(120)(로우 레이턴시 메모리일 수 있고, 따라서 시스템의 메인 메모리보다 수십배 더 빠를 수 있음)로부터 무결절로 세션 콘텍스트를 페칭하는 능력은 오프로드 프로세서들(118)의 L2 캐시의 크기를 효율적으로 확장시키도록 기능할 수 있다.

[0034] 일부 실시예들에서, 시스템(100)의 OS는, TLB(translation lookaside buffer)(또는 등가의 룩업 구조)가 각각의 세션의 콘텐츠를 뚜렷하게 식별하도록 허용하기 위해 자신의 IOMMU(input output memory management unit)(미도시)에서 최적화들을 구현할 수 있다. 그러한 어레인지먼트는 어드레스 변환들이 세션 스위치 아웃 동안에 뚜렷하게 식별되고 TLB 외부에 있는 페이지 테이블 캐시로 전송되도록 허용할 수 있다. 페이지 테이블 캐시의 사용은 TLB의 크기에서의 확장을 허용할 수 있다. 또한, 가상 메모리 내의 연속적인 위치들이 물리적 메모리에서 및 물리적으로 인덱싱된 캐시 내의 연속적인 위치들에 있다는 사실이 주어지면, 세션을 식별하기 위해 요구되는 어드레스 변환들의 수가 상당히 감소된다.

[0035] 도 1a의 특정 실시예에서, 시스템(100)은 세션 및 패킷 종료 서비스들을 제공하기에 매우 적합할 수 있다. 일부 실시예들에서, 네트워크 스택 프로세싱의 제어는 스케줄러(116)에 의해 수행될 수 있다. 따라서, 스케줄러(116)는 트래픽 관리 큐, 중재 회로 및 네트워크 스택 오프로드 디바이스로서 작동할 수 있다. 스케줄러(116)는 오프로드 프로세서들(118) 대신에 전체 세션 및 플로우 관리를 처리하는 것을 담당할 수 있다. 그러한 어레인지먼트에서, 오프로드 프로세서들(118)에는 버퍼로의 세션에 직접적으로 속하는 패킷들이 피딩될 수 있고, 오프로드 프로세서는 버퍼로부터 사용을 위한 패킷 데이터를 추출할 수 있다. 네트워크 스택의 프로세싱은 네트워크 생성된 인터럽트들을 처리(그리고 인터럽트 서비스 루틴을 실행)하기 위해 커널 모드로의 스위치들을 회피하기 위해 최적화될 수 있다. 이러한 방식으로, 시스템(100)은 세션들의 콘텍스트 스위칭을 무결절로 그리고 가능한 적은 오버헤드로 수행하도록 최적화될 수 있다.

[0036] 여전히 도 1a을 참조하면, 이해될 바와 같이, PCI, 파이버 채널과 같은 다수의 타입들의 종래의 입력/출력 버스들이 설명된 시스템(100)에 사용될 수 있다. 버스 아키텍처는 또한 관련 JEDEC 표준들, DIMM 데이터 전송 프로토콜들, HyperTransport 또는 임의의 다른 고속, 로우 레이턴시 상호접속 시스템에 기초할 수 있다. 오프로드 프로세서들(118)은 DDR, DRAM, RLDRAM, 임베디드 DRAM, HMC(Hybrid Memory Cube)와 같은 차세대 적층 메모리, 플래시, 또는 다른 적절한 메모리, 별개의 로직 또는 버스 관리 칩들, FPGA들(field programmable gate arrays)과 같은 프로그래머블 유닛들, 고객 설계 ASIC들(application specific integrated circuits) 및 ARM, ARC, 텐실리카, MIPS, Strong/ARM 또는 RISC 아키텍처들에 기반한 것들과 같은 에너지 효율적인 범용 프로세서를 포함할 수 있다. 호스트 프로세서들(110)은 인텔 또는 AMD x86 아키텍처, 인텔 아이테니엄 아키텍처, MIPS 아키텍처, SPARC 아키텍처 등에 기초한 것들을 비롯해서 범용 프로세서일 수 있다.

[0037] 또한 이해될 바와 같이, 도 1a의 시스템에 의해 수행되는 것과 같은 프로세싱을 실행하는 종래의 시스템들은 다수의 프로세싱 코어들 상에서 실행되는 다수의 스레드들 상에서 구현될 수 있다. 다수의 스레드 콘텍스트들로의 태스크들의 그러한 병렬화는 증가된 스루풋을 제공할 수 있다. MIPS와 같은 프로세서 아키텍처들은 사이클당 명령들의 수를 개선하기 위해 심도있는 명령 파이프라인들을 포함할 수 있다. 또한, 다중-스레디드 프로그래밍 환경을 실행하는 능력은 기존의 프로세서 리소스들의 개선된 사용을 초래한다. 하드웨어 상의 병렬 실행을 추가로 증가시키기 위해, 프로세서 아키텍처들은 다수의 프로세서 코어들을 포함할 수 있다. 동질적인 코어 아키텍처들로 지칭되는 동일한 타입의 코어들을 포함하는 다중-코어 아키텍처들은 다수의 코어들에 걸쳐 스레드들 또는 프로세스들을 병렬화함으로써 더 높은 명령 스루풋을 제공한다. 그러나, 그러한 동질적인 코어 아키텍처들에서, 메모리와 같은 공유된 리소스들은 작은 수의 프로세서들에 걸쳐 분할된다. 또 다른 실시예들에서, 다수의 오프로드 또는 호스트 프로세서들은, 결국 랙들 또는 별개의 서버들 상에 상주하는 별개의 랙 유닛들 또는 블레이드들에 접속된 모듈들 상에 상주할 수 있다. 이들은 추가로 클러스터들 및 데이터센터들로 그룹화될 수 있고, 이것은 공간적으로 동일한 빌딩, 동일한 도시 또는 심지어 상이한 국가들에 위치될 수 있다. 임의의

그룹 레벨은 서로 접속될 수 있고 및/또는 공개 또는 비밀 클라우드 인터넷들에 접속될 수 있다.

[0038] 그러한 종래의 접근법들에서, 메모리 및 I/O 액세스들은 많은 양의 프로세서 오버헤드를 발생시킬 수 있다. 추가적으로, 본 명세서에서 언급된 바와 같이, 종래의 범용 프로세싱 유닛들 내의 콘텍스트 스위치들은 계산 집중적일 수 있다. 따라서, 프로세서 스루풋을 증가시키기 위해 복수의 네트워킹된 애플리케이션들을 핸들링하여, 네트워킹된 컴퓨팅 리소스에서 콘텍스트 스위치 오버헤드를 감소시키는 것이 바람직하다. 종래의 서버 로드들은, 복잡한 전송, 높은 메모리 대역폭, 심각한 양들의 데이터 대역폭을 요구할 수 있지만(랜덤하게 액세스되고, 병렬화되고, 매우 이용가능함), 종종 라이트 터치 프로세싱, 즉 HTML, 비디오, 패킷-레벨 서비스들, 보안, 및 분석을 종종 이용한다. 추가적으로, 유휴 프로세서들은 여전히, 그들의 피크 전력 소비스의 50% 초과를 소비한다.

[0039] 대조적으로, 도 1a 또는 등가물에 도시된 것과 같은 일 실시예에서, 복잡한 전송, 데이터 대역폭 집중, 빈번한 랜덤 액세스 지향된 "라이트 터치" 프로세싱은 다수의 오프로드 프로세서(118) 코어들 상에서 생성된 소켓 추상화(abstraction) 뒤에서 핸들링될 수 있다. 동시에, "헤비 터치", 즉 컴퓨팅 집약적인 로드들은, 호스트 프로세서(110) 코어(예를 들어, x86 프로세서 코어들) 상에서 소켓 추상화에 의해 핸들링될 수 있다. 그러한 소프트 소켓들은, 라이트 터치(예를 들어, ARM)와 헤비 터치(예를 들어, x86) 프로세서 코어들 사이에서 이들 로드들의 자연적인 분할을 허용할 수 있다. 실시예들에 따라 새로운 애플리케이션 레벨 소켓들의 사용에 의해, 서버 로드들은 오프로드 프로세서들(118) 및 호스트 프로세서(들)(110)에 걸쳐 분할될 수 있다.

[0040] 본 명세서에 기재된 실시예들의 동작들을 더 양호하게 이해하기 위해, 종래의 캐시 방식들이 도 1b 및 1c를 참조하여 설명된다. 가상 메모리를 구현하는 현대의 운영 시스템들은, 프로세스들에 대한 가상 및 물리 메모리 둘 모두의 할당을 담당하며, 프로세스가 실행하고 가상적으로 어드레싱된 메모리에 액세스하는 경우 발생하는 가상 투 물리 변환들을 초래한다. 프로세스에 대한 메모리의 관리에서, 가상 어드레스 범위와 가상 어드레스들에 의해 매핑될 대응하는 물리 어드레스들의 할당 사이에 어떠한 조정도 통상적으로 존재하지 않는다. 조정의 이러한 부재는, 프로세스가 실행하는 경우 프로세서 캐시 오버헤드 및 효율 둘 모두에 영향을 줄 수 있다.

[0041] 종래의 시스템들에서, 프로세서는, 실행하고 있는 각각의 프로세스에 대해 가상 메모리에서 인접한 메모리 페이지들을 할당한다. 프로세서는 또한, 반드시 인접할 필요는 없는 물리 메모리에서 페이지들을 할당한다. 변환 방식은, 가상 메모리의 추상화가 물리 메모리 페이지들에 의해 정확히 지원된다는 것을 보장하기 위해 2개의 방식들의 어드레싱 사이에서 설정된다. 프로세서는, 즉시의 데이터 프로세싱 필요성들을 충족시키기 위해 프로세서에 인접하여 상주하는 캐시 블록들을 이용할 수 있다. 종래의 캐시들은 계층적으로 배열될 수 있다. 레벨 1(L1) 캐시들은 프로세서에 가장 인접하고, 후속하여 L2, L3 등이 근접한다. L2는 L1에 대한 백업을 동작시키는 등의 식이다. 프로세스의 물리 어드레스들의 일부에 의해 인덱싱된 캐시들에 대해, 메모리 관리 유닛(MMU) 페이지의 사이즈를 넘어선 어드레스들의 범위에 대한 가상 및 물리 메모리의 할당 사이의 상관에서의 부족은, 프로세서 캐시들에서 무작정(haphazard)이고 비효율적인 효과들을 초래한다. 이것은 캐시 오버헤드들을 증가시키고, 지연이 콘텍스트 스위치 동작 동안 도입된다.

[0042] 물리적으로 어드레싱된 캐시들에서, 가상 메모리 내의 다음의 페이지에 대한 캐시 엔트리는, 캐시 내의 다음의 인접한 페이지에 대응하지 않을 수도 있으며, 따라서, 달성될 수 있는 전체 성능을 열화시킨다. 예를 들어, 도 1b에서, 가상 메모리(130) 내의 인접한 페이지들(프로세스 1의 페이지 1 및 2)은, 물리 메모리(132) 내의 그들의 물리 어드레스들이 (프로세서의) 물리적으로 인덱싱된 캐시(134)의 동일한 위치에 인덱싱하는 경우 충돌한다. 즉, 프로세서 캐시(즉, 134)가 물리적으로 인덱싱되며, 물리 메모리(132) 내의 페이지들의 어드레스들은 프로세서 캐시 내의 동일한 페이지에 인덱싱한다. 또한, 공유 캐시에 액세스하는 다수의 프로세스들의 효과들이 고려되는 경우, OS가 물리 메모리를 프로세스들에 할당하는 때에 전체 캐시 성능에 대한 고려 부족이 통상적으로 존재한다. 이러한 고려 부족은, 서로의 라인들을 불필요하게 대체할 수 있는 콘텍스트 스위치들에 걸쳐 캐시에서 스레싱(thrash)하는 상이한 프로세스들(예를 들어, 도 1b의 프로세스 1 및 프로세스 2)을 초래하며, 이는, 프로세스를 재개할 시에 불명확한 수의 캐시 미스/충진, 또는 콘텍스트 스위치에 걸친 증가된 수의 라인 라이트백(writeback)들을 초래할 수 있다.

[0043] 도 1c를 참조하여 설명된 바와 같이, 다른 종래의 어레인지먼트(arrangement)들에서, 프로세서 캐시들은 프로세스의 가상 어드레스들의 일부에 의해 대안적으로 인덱싱될 수 있다. 가상적으로 인덱싱된 캐시들은, 프로세서의 가상 어드레스의 비트들의 섹션을 사용함으로써 액세스된다. 가상 메모리(130)에서 인접한 페이지들은 도 1c에서 관측되는 바와 같이, 가상적으로 인덱싱된 캐시들(136)에서 인접할 것이다. 프로세서 캐시들이 가상적으로 인덱싱되는 한, 가상 어드레스들의 할당과 물리 메모리(132)의 할당을 조정하는 것에 대해 어떠한 주의도

지불될 필요가 없다. 프로그램들이 가상 어드레스 범위들에 걸쳐 스웝하는 경우, 그 프로그램들은 프로세서 캐시에서 공간 로컬러티의 이점들을 향유할 것이다. 그러한 세트-연관 캐시들은 인덱스에 대응하는 수 개의 엔트리들을 가질 수 있다. 주어진 캐시 인덱스 상으로 매핑하는 주어진 페이지는 그 특정한 세트 내의 임의의 장소에 있을 수 있다. 캐시 엔트리에 대해 이용가능한 수 개의 포지션들이 존재한다고 가정하면, 프로세서가 가능한 가장 긴 정도로 캐시 내의 사용된 엔트리들을 유지할 여유가 있을 수 있으므로, (즉, 도 1b에 도시된 바와 같이) 콘텍스트 스위치들에 걸쳐 캐시에서 스레싱하는 것을 초래했던 문제점들은 세트-연관 캐시들에 대해 특정한 정도로 완화된다. 이를 위해, 캐시들은 적어도 최근에 사용된 알고리즘을 이용한다. 이것은, 가상 어드레싱 방식, 후속하여 운영 시스템과 연관된 문제점들 중 몇몇의 완화를 초래하지만, 캐시의 사이즈에 제약들을 배치한다. 따라서, 더 큰 멀티-방식 연관 캐시들은, 최근에 사용된 엔트리들이 유효하지 않게 되고/플러시 아웃(flush out)되지 않는다는 것을 보장하기 위해 요구될 수 있다. 멀티-방식 세트 연관 캐시에 대한 비교기 회로는 병렬 비교를 수용하도록 복잡할 수 있으며, 이는 캐시와 연관된 회로 레벨 복잡도를 증가시킨다.

[0044] "페이지 컬러링"으로 알려진 캐시 제어 방식은, 가상 어드레싱 방식으로 인한 캐시-미스들의 문제점을 처리하기 위하여 몇몇 종래의 운영 시스템들에 의해 사용된다. 프로세서 캐시가 물리적으로 인덱싱되었다면, 운영 시스템은, 동일한 컬러의 캐시 내의 위치들로 인덱싱하지 않을 물리 메모리 위치들을 찾기 위해 제한되었다. 그러한 캐시 제어 방식 하에서, 운영 시스템은, 모든 각각의 가상 어드레스에 대해, 그들 페이지들이 물리적으로 인덱싱된 캐시에 해시하는 인덱스에 기초하여 허용가능한 물리 메모리 내의 그들 페이지들에 액세스해야 할 것이다. 도출된 인덱스들이 동일한 컬러를 가질 수도 있으므로, 수 개의 물리 어드레스들이 허용되지 않는다. 그러므로, 물리적으로 인덱싱된 캐시들에 대해, 가상 메모리 내의 모든 각각의 페이지는, 자신의 대응하는 캐시 위치를 식별하며, 다음의 페이지가 물리 메모리 및 그에 따른 동일한 컬러의 캐시 위치에 할당되는지 또는 할당되지 않는지를 결정하기 위해 컬러링될 것이다. 이러한 프로세스는 모든 각각의 페이지에 대해 반복될 것이며, 이는 성가신 동작일 수 있다. 그것이 캐시 효율을 개선시키는 동안, 최근에 사용된 페이지들이 오버라이팅되는 것을 방지하기 위해 모든 각각의 페이지의 컬러들이 식별되어야 할 것이므로, 페이지 컬러링은 메모리 관리 및 변환 유닛에 대한 오버헤드를 증가시킨다. 운영 시스템의 복잡도 레벨은, 그것이 캐시 내의 이전의 가상 메모리 페이지의 컬러의 표시자를 유지할 필요가 있으므로 대응적으로 증가한다.

[0045] 가상으로 인덱싱된 캐시에 대한 문제점은, 캐시 액세스 레이턴시들이 더 높다는 사실에도 불구하고, 에일리어싱의 만연한 문제점이 존재한다는 것이다. 에일리어싱의 경우에서, 물리 메모리 내의 동일한 페이지에 매핑하는 (상이한 인덱스들을 갖는) 다수의 가상 어드레스들은 (상이한 인덱스들로 인해) 캐시 내의 상이한 위치들에 있다. 페이지 컬러링은, 가상 페이지들 및 물리 페이지들이 동일한 컬러를 갖게 하고 따라서 캐시에서 동일한 세트를 점유하게 한다. 페이지 컬러링은, 에일리어싱이 동일한 슈퍼세트 비트들을 공유하고 캐시 내의 동일한 라인들에 인덱싱하도록 한다. 이것은 에일리어싱의 문제점을 제거한다. 페이지 컬러링은 또한, 메모리 할당에 제약들을 부과한다. 새로운 물리 페이지가 페이지 실수로 할당되는 경우, 메모리 관리 알고리즘은 자유 리스트로부터 가상 컬러와 동일한 컬러를 갖는 페이지를 선정해야 한다. 시스템들이 가상 공간을 시스템적으로 할당하기 때문에, 상이한 프로그램들의 페이지들은 동일한 컬러들을 가지는 경향이 있으며, 따라서, 몇몇 물리 컬러들은 다른 것들보다 더 빈번할 수도 있다. 따라서, 페이지 컬러링은 페이지 실패 레이트에 영향을 줄 수도 있다. 또한, 몇몇 물리 컬러들의 우위는, 물리 어드레스들을 이용하여 액세스된 제 2 레벨 캐시 내의 프로그램들 사이에서 매핑 충돌들을 생성할 수도 있다. 따라서, 프로세서는 직전에 설명된 종래의 페이지 컬러링 방식에 대해 매우 큰 문제점에 직면된다. 가상 페이지들 각각은, 그들이 상이한 캐시 컬러들을 점유하도록 물리 메모리에서 상이한 페이지들을 점유하고 있을 수 있지만, 프로세서는 각각의 및 모든 각각의 페이지의 어드레스 변환을 저장할 필요가 있을 것이다. 프로세스가 충분히 클 수 있고, 각각의 프로세스가 수 개의 가상 페이지들을 포함할 것이라고 가정하면, 페이지 컬러링 알고리즘은 매우 복잡하게 될 수 있다. 이것은 또한, 그것이 프로세서의 가상 메모리, 동등하게는 물리 어드레스의 각각의 페이지를 식별할 필요가 있으므로, TLB 단에서 그것을 복잡하게 할 것이다. 콘텍스트 스위치들이 TLB 엔트리들을 유효하게 하지 않는 경향이 있으므로, 프로세서는 페이지 워크(walk)들을 수행하고 TLB 엔트리들을 충진시킬 필요가 있을 것이며, 이것은, 루틴 콘텍스트 스위치인 것에 비결정성 및 레이턴시를 추가적으로 부가할 것이다.

[0046] 이러한 방식으로, 일반적으로 사용가능한 종래의 운영 체제들에서, 컨텍스트 스위치들은, 프로세스/스레드가 재개될 때 캐시 내 충돌들뿐만 아니라 TLB 손실들을 초래한다. 프로세스/스레드가 재개할 때, 스레드의 작업(working) 세트가 캐시 내부로 다시 재로딩됨에 따라, 불확정적 수의 명령 및 데이터 캐시 손실들이 존재한다(즉, 스레드가 사용자 공간에서 재개하고 명령들을 실행함에 따라, 명령들은 통상적으로 애플리케이션 데이터와 함께 캐시 내부로 로딩되어야 할 필요가 있을 것이다). 스위치-인(즉, 프로세스/스레드의 재개) 시에, TLB 맵핑들은 완전하게 또는 부분적으로 무효화될 수 있으며, 여기서 레지스터에 기록된 새로운 스레드의 페이지 테이

블들의 베이스가 그 목적을 위해 예비된다. 스레드를 실행함에 따라, TLB 손실들은 (하드웨어 또는 소프트웨어에 의한) 페이지 테이블 워크(page table walks)를 초래할 것이며, 이는 TLB 충진(fill)들을 초래한다. 이러한 TLB 손실들 각각은 예외(예를 들어, 페이지 테이블들이 캐시 내에 있지 않다면, 관련 캐시 손실들/메모리 로드들과 함께, 페이지 테이블 워크를 수행할 때 메모리 액세스들에 의해 생성되는 오버헤드)로 인해 파이프라인 스톨을 포함하는 자신 소유의 하드웨어 비용을 갖는다. 이러한 비용들은, 프로세스의 연속 실행들 사이에 프로세서에서 발생했던 것에 의존하며, 이에 따라 고정된 비용들은 아니다. 게다가, 이러한 여분의 레이턴시들은, 콘텍스트 스위치의 비용에 부가하며, 프로세스의 효율적인 실행을 손상한다. 인식되는 바와 같이, 이렇게 앞서 설명된 캐시 제어 방법들은, 프로세싱 시간, 메모리 요건들, 또는 다른 운영 체제 제어 리소스들에 관련하여 비-결정적이며, 이는 시스템 운영의 전반적인 효율을 저하시킨다.

[0047] 도 1d은, 일 실시예에 따른 캐시 제어 시스템을 나타낸다. 캐시 제어 시스템에서, 세션 컨텐츠들은 물리적으로 인덱싱된 캐시(134') 내에서 인접해 있을 수 있다. 설명된 실시예는 전환 방식(translation scheme)을 이용할 수 있으며, 이에 따라 가상 메모리(130) 내 세션의 연속 페이지들이 물리적으로 인덱싱된 캐시(134) 내에서 물리적으로 인접하게 된다. 앞서 설명된 비-결정적 캐시 제어 방식들과 대조적으로, 적어도 콘텍스트 스위치 동작의 지속주기는 결정적일 수 있다. 설명된 실시예에서, 새로운 프로세스의 콘텍스트에 의해 이전 프로세스의 콘텍스트를 대체하는 것은, 도 1a의 콘텍스트 메모리(120)에 의해 제공되는 것과 같은 외부의 로우 레이턴시 메모리로부터 새로운 프로세스 콘텍스트를 전송하는 것(transferring)을 수반한다. 콘텍스트 스위칭의 프로세스 시에, 시스템의 메인 메모리의 액세스는 회피될 수 있다(여기서, 이러한 액세스들은 딜레이 집중적일 수 있다). 프로세스 콘텍스트는 (로우 레이턴시 메모리일 수 있는) 콘텍스트 메모리(120)로부터 프리페치된다. 다른 콘텍스트 스위치에 대해 필요로 되는 경우, 프로세스 콘텍스트는 콘텍스트 메모리(120)에 다시 한 번 저장될 수 있다. 이러한 방식으로, 콘텍스트 스위치 동작이 수행되기 위해 필요한 동작들 및 사이클들의 수와 관련하여 정의될 수 있기 때문에, 결정적 콘텍스트 스위칭이 달성된다. 또한, 콘텍스트 데이터를 저장하기 위한 로우 레이턴시 메모리의 이용은 신속한(rapid) 콘텍스트 스위칭을 제공할 수 있다.

[0048] 도 2a 내지 도 2f는, 본원에 설명된 것과 같은 콘텍스트 스위칭을 포함할 수 있는 모듈의 하드웨어 실시예들의 양상들을 설명한다. 특정 실시예들에서, 이러한 프로세싱 모듈들은 DIMM 탑재가능 모듈들을 포함할 수 있다.

[0049] 도 2a은 일 실시예에 따른 프로세싱 모듈(200)의 블록도이다. 프로세싱 모듈(200)은, 물리적 커넥터(202), 메모리 인터페이스(204), 아비터 로직(206), 오프로드 프로세서(들)(208), 로컬 메모리(210), 및 제어 로직(212)을 포함할 수 있다. 커넥터(202)는 시스템 메모리 버스에 대한 물리적 접속을 제공할 수 있다. 이는, 메모리 제어기 등을 통해 시스템 메모리 버스에 액세스할 수 있는 호스트 프로세서와는 대조적이다. 매우 특정한 실시예들에서, 커넥터(202)는 컴퓨팅 시스템의 듀얼-인-라인 메모리 모듈(DIMM) 슬롯과 호환가능할 수 있다. 이에 따라, 다수의 DIMM 슬롯들을 포함하는 시스템은 하나 또는 그 초과의 프로세싱 모듈들(200), 또는 프로세싱 모듈들과 DIMM 모듈들의 혼합(mix)으로 파퓰레이팅될 수 있다(populated).

[0050] 메모리 인터페이스(204)는, 시스템 메모리 버스 상에서의 데이터 전송들을 검출할 수 있고, 그리고 적당한 경우들에서, 기록 데이터가 프로세싱 모듈(200) 내에 저장되는 것을 가능하게 하고 그리고/또는 판독 데이터가 프로세싱 모듈(200)로부터 판독되는 것을 가능하게 할 수 있다. 이러한 데이터 전송들은 특정 네트워크 식별자를 갖는 패킷 데이터의 수신을 포함할 수 있다. 일부 실시예들에서, 메모리 인터페이스(204)는 슬레이브 인터페이스일 수 있고, 이에 따라 데이터 전송들은 프로세싱 모듈(200)과는 별도로 마스터 디바이스에 의해 제어된다. 매우 특정한 실시예들에서, 메모리 인터페이스(204)는, 직접 메모리 액세스(DMA) 마스터에 의해 개시된 시스템 메모리 버스를 통해 DMA 전송들을 수용하기 위한 직접 메모리 액세스(DMA) 슬레이브일 수 있다. 일부 실시예들에서, DMA 마스터는 호스트 프로세서와는 상이한 디바이스일 수 있다. 이러한 구성들에서, 프로세싱 모듈(200)은 프로세싱을 위한 데이터를 수신할 수 있고(예를 들어, DMA 기록), 호스트 프로세서 리소스들을 소모하지 않고 프로세싱된 데이터를 전송할 수 있다(예를 들어, DMA 판독).

[0051] 아비터 로직(206)은 프로세싱 모듈(200) 내에서 데이터의 충돌 액세스들 사이를 중재할 수 있다. 일부 실시예들에서, 아비터 로직(206)은 오프로드 프로세서(208)에 의한 액세스들과 프로세서 모듈(200) 외부로의 액세스들 사이를 중재할 수 있다. 프로세싱 모듈(200)이 동시에 동작되는 다수의 위치들을 포함할 수 있다는 점이 이해된다. 아비터 로직(206)에 의해 중재되는 액세스들이 프로세서 모듈(200)에 의해 점유되는 물리적 시스템 메모리 공간에 대한 액세스들 뿐만 아니라 다른 리소스들(예를 들어, 오프로드 또는 호스트 프로세서의 캐시 메모리)로의 액세스들을 포함할 수 있다는 점이 이해된다. 이에 따라, 아비터 로직(206)에 대한 중재 규칙들은 애플리케이션에 따라 변할 수 있다. 일부 실시예들에서, 주어진 프로세서 모듈(200)에 대한 이러한 중재 규칙들은 고정된다. 이러한 경우들에서, 상이한 프로세싱 모듈들을 스위칭함으로써 상이한 애플리케이션들이 수용

될 수 있다. 그러나, 대안적인 실시예들에서, 이러한 중재 규칙들은 변경가능(configurable)할 수 있다.

[0052] 오프로드 프로세서(208)는 시스템 메모리 버스를 통해 전송되는 데이터에 대해 동작할 수 있는 하나 또는 그 초과의 프로세서들을 포함할 수 있다. 일부 실시예들에서, 오프로드 프로세서들은 프로세서 컨텍스트들이 절약되고 리트리브되는 것을 가능하게 하는 (그러나, 하나의 매우 특정한 예시로서) Apache와 같은 범용 운영 체제 또는 서버 애플리케이션들을 실행할 수 있다. 오프로드 프로세서(208)에 의해 실행되는 컴퓨팅 태스크들은 하드웨어 스케줄러에 의해 처리될 수 있다. 오프로드 프로세서들(208)은 프로세서 모듈(200)에서 버퍼링된 데이터에 대해 동작할 수 있다. 이에 더해 또는 대안적으로, 오프로드 프로세서들(208)은 시스템 메모리 공간 내의 어딘가에 저장된 데이터에 액세스할 수 있다. 일부 실시예들에서, 오프로드 프로세서들(208)은 콘텍스트 정보를 저장하도록 구성된 캐시 메모리를 포함할 수 있다. 오프로드 프로세서(208)는 다수의 코어들 또는 하나의 코어를 포함할 수 있다.

[0053] 프로세서 모듈(200)은 호스트 프로세서(미도시)를 갖는 시스템 내에 포함된다. 일부 실시예들에서, 오프로드 프로세서들(208)은 호스트 프로세서들과 비교하여 상이한 유형의 프로세서일 수 있다. 특정 실시예들에서, 오프로드 프로세서들(208)은 호스트 프로세서보다 적은 전력을 소모할 수 있고 그리고/또는 호스트 프로세서보다 적은 컴퓨팅 전력을 가질 수 있다. 매우 특정한 실시예들에서, 오프로드 프로세서들(208)은 "윔피(wimpy)" 코어 프로세서들일 수 있는 반면, 호스트 프로세서는 "브로니(brawny)" 코어 프로세서일 수 있다. 그러나, 대안적인 실시예들에서, 오프로드 프로세서(208)는 임의의 호스트 프로세서에 동일한 컴퓨팅 전력을 가질 수 있다. 매우 특정한 실시예들에서, 호스트 프로세서는 x86 유형의 프로세서일 수 있는 반면, 오프로드 프로세서(208)는, 몇몇 예시들로서, ARM, ARC, Tensilica, MIPS, Strong/ARM, 또는 RISC 유형의 프로세서를 포함할 수 있다.

[0054] 로컬 메모리(210)는 콘텍스트 정보의 저장을 인에이블하기 위해 오프로드 프로세서(208)에 접속될 수 있다. 이에 따라, 오프로드 프로세서(208)는, 현재 콘텍스트 정보를 저장할 수 있고, 그후 새로운 컴퓨팅 태스크로 스위칭한 후, 후속하여 이전 태스크를 재개하기 위해 콘텍스트 정보를 리트리브할 수 있다. 매우 특정한 실시예들에서, 로컬 메모리(210)는 시스템 내의 다른 메모리들와 관련된 로우 레이턴시 메모리일 수 있다. 일부 실시예들에서, 콘텍스트 정보의 저장은, 오프로드 프로세서(208) 캐시를 카피하는 것을 포함할 수 있다.

[0055] 일부 실시예들에서, 로컬 메모리(210) 내의 동일한 공간이 동일한 유형의 다수의 오프로드 프로세서들(208)에 의해 액세스가능하다. 이러한 방식으로, 하나의 오프로드 프로세서에 의해 저장된 콘텍스트는 상이한 오프로드 프로세서에 의해 재개될 수 있다.

[0056] 제어 로직(212)은 오프로드 프로세서(들)에 의해 실행된 프로세싱 태스크들을 제어할 수 있다. 일부 실시예들에서, 제어 로직(212)은, 데이터 평가자(214), 스케줄러(216) 및 스위치 제어기(218)를 포함하는 것으로서 개념화될 수 있는 하드웨어 스케줄러로 고려될 수 있다. 데이터 평가자(214)는, 시스템 메모리 버스를 통해 전송되는 기록 데이터로부터 "메타데이터"를 추출할 수 있다. 본원에 이용되는 것과 같은 "메타데이터"는, 기록 데이터의 블록의 전부 또는 일부에 대해 수행될 프로세싱을 나타내고 그리고/또는 데이터가 속하는 특정 태스크/프로세스(예를 들어, 분류 데이터)를 나타내는 기록 데이터의 블록의 하나 또는 그 초과의 미리결정된 위치들에 임베딩된 임의의 정보일 수 있다. 일부 실시예들에서, 메타데이터는 기록 데이터의 블록에 대한 더 상위 레벨의 조직을 나타내는 데이터일 수 있다. 그러나 하나의 매우 특정한 실시예로서, 메타데이터는 (더 상위 계층 패킷 구조 내에 캡슐화될 수 있거나 또는 캡슐화되지 않을 수 있는) 하나 또는 그 초과의 네트워크 패킷들의 헤더 정보일 수 있다.

[0057] 스케줄러(216)(예를 들어, 하드웨어 스케줄러)는 오프로드 프로세서(들)(208)에 대한 컴퓨팅 태스크들을 순서화(order)할 수 있다. 일부 실시예들에서, 스케줄러(216)는 프로세싱에 대한 기록 데이터가 수신되는 것으로서 계속적으로 업데이트되는 스케줄을 생성할 수 있다. 매우 특정한 실시예들에서, 스케줄러(216)는 오프로드 프로세서(들)(208)의 컨텍스트들을 스위치하기 위한 능력에 기초하여 이러한 스케줄을 생성할 수 있다. 이 방식으로, 온-모듈 컴퓨팅 우선순위들이 온 더 플라이(on the fly)로 조정될 수 있다. 매우 특정한 실시예들에서, 스케줄러(216)는 컴퓨팅 태스크들에 따라, 물리 어드레스 공간의 일부분(예를 들어, 로컬 메모리(210) 내의 메모리 위치들)을 오프로드 프로세서(208)에 할당할 수 있다. 그 다음, 오프로드 프로세서(208)는 이러한 상이한 공간들 사이에서 스위치할 수 있고, 각각의 스위치에 앞서 컨텍스트 정보를 저장하며, 후속적으로, 메모리 공간으로 리턴할 때 컨텍스트 정보를 복원한다.

[0058] 스위치 제어기(218)는 오프로드 프로세서(들)(208)의 컴퓨팅 동작들을 제어할 수 있다. 특정 실시예들에서, 스케줄러(216)에 따라, 스위치 제어기(218)는 컨텍스트들을 스위치하기 위해 오프로드 프로세서(들)(208)를 순서

화할 수 있다. 컨텍스트 스위치 동작은 스위치 제어기(218)로부터의 단일 커맨드에 응답하여 실행되는 "원자" 동작일 수 있다는 것이 이해된다. 추가적으로 또는 대안적으로, 스위치 제어기(218)는 현재 컨텍스트 정보를 저장하고, 컨텍스트 정보를 리콜하는 등의 명령 세트를 발행할 수 있다.

[0059] 일부 실시예들에서, 프로세서 모듈(200)은 버퍼 메모리(미도시)를 포함할 수 있다. 버퍼 메모리는 프로세서 모듈 내에(on board) 수신된 기록 데이터를 저장할 수 있다. 버퍼 메모리는 메모리 디바이스들의 전체적으로 상이한 세트 상에서 구현될 수 있거나, 로직 및/또는 오프로드 프로세서에 임베딩된 메모리일 수 있다. 후자의 경우, 아비터 로직(206)은 버퍼 메모리로의 액세스를 중재(arbitrate)할 수 있다. 일부 실시예들에서, 버퍼 메모리는 시스템 물리 메모리 공간의 일부분에 대응할 수 있다. 시스템 메모리 공간의 나머지 부분은 다른 유사 프로세서 모듈들 및/또는 동일한 시스템 메모리 버스에 연결된 메모리 모듈들에 대응할 수 있다. 일부 실시예들에서, 버퍼 메모리는 로컬 메모리(210)와 상이할 수 있다. 예를 들어, 버퍼 메모리는 로컬 메모리(210)보다 더 느린 액세스 시간을 가질 수 있다. 그러나, 다른 실시예들에서, 버퍼 메모리 및 로컬 메모리는 유사 메모리 디바이스들로 구현될 수 있다.

[0060] 매우 특정한 실시예들에서, 프로세싱하기 위한 기록 데이터는 예상된 최대 흐름 레이트를 가질 수 있다. 프로세서 모듈(200)은 이러한 흐름 레이트에서 또는 이러한 흐름 레이트보다 더 빠르게 이러한 데이터에 대해 동작하도록 구성될 수 있다. 이 방식으로, 마스터 디바이스(미도시)는 "진행 중인" 데이터를 겹쳐쓸 위험 없이 프로세서 모듈에 데이터를 기록할 수 있다.

[0061] 프로세서 모듈(200)의 다양한 컴퓨팅 엘리먼트들은 하나 또는 둘 이상의 집적 회로 디바이스들(IC들)로서 구현될 수 있다. 도 2a에 도시된 다양한 컴포넌트들이 동일하거나 상이한 IC들에서 형성될 수 있다는 것이 이해된다. 예를 들어, 제어 로직(212), 메모리 인터페이스(214) 및/또는 아비터 로직(206)은 하나 또는 둘 이상의 로직 IC들로 구현될 수 있는 반면, 오프로드 프로세서(들)(208) 및 로컬 메모리(210)는 개별 IC들이다. 로직 IC들은 고정된 로직(예를 들어, 주문형 IC들), 프로그래머블 로직(예를 들어, 필드 프로그래머블 게이트 어레이들, FPGA들) 또는 이들의 결합들일 수 있다.

[0062] 유리하게, 상기 하드웨어 및 시스템들은 전통적 컴퓨팅 시스템들에 비해 향상된 컴퓨테이션 성능을 제공할 수 있다. x86 프로세서들에 기초한 것들을 포함하는 종래 시스템들은 종종 이러한 높은 볼륨 애플리케이션들을 핸들링하기 위해 장착되어 있지 않다. 심지어 유휴 상태일 때도, x86 프로세서들은 상당한 양의 전력을 이용하고, 높은 대역폭 패킷 분석 또는 다른 높은 볼륨 프로세싱 태스크들에 대한 거의 연속적 동작은 프로세서 에너지 비용들에 대한 지배적 가격 인자들 중 하나가 되게 한다.

[0063] 또한, 종래의 시스템들은 컨텍스트 스위칭의 높은 비용을 갖는 문제들을 가질 수 있고, 여기서, 호스트 프로세서는 하나의 스레드로부터 또 다른 스레드로 스위치하는 것을 포함할 수 있는 명령들을 실행하도록 요구된다. 이러한 스위치는 스레드에 대한 컨텍스트를 저장 및 리콜하는 것을 요구할 수 있다. 이러한 컨텍스트 데이터가 호스트 캐시 메모리에 상주하면, 이러한 컨텍스트 스위치는 비교적 빨리 발생할 수 있다. 그러나, 이러한 컨텍스트 데이터가 캐시 메모리(즉, 캐시 미스) 내에 더이상 있지 않다면, 데이터는 시스템 메모리로부터 리콜되어야 하고, 이는 멀티-사이클 레이턴시를 발생시킬 수 있다. 컨텍스트 스위칭 동안의 연속적 캐시 미스들은 시스템 성능에 악영향을 미칠 수 있다.

[0064] 도 2b은 많은 종래의 서버 시스템들과 연관된 컨텍스트 스위칭 또는 높은 볼륨 프로세싱과 연관된 문제들을 감소시킬 수 있는 하나의 매우 특정한 실시예에 따른 프로세서 모듈(200-1)을 도시한다. 프로세서 모듈(200-1)은 PCB(printed circuit board) 타입 기판(222)에 장착된 IC들(220-0/1)을 포함할 수 있다. PCB 타입 기판(222)은 인-라인 모듈 커넥터(202)를 포함할 수 있으며, 인-라인 모듈 커넥터(202)는 하나의 매우 특정한 실시예에서, DIMM 호환가능한 커넥터일 수 있다. IC(220-0)는 다수의 기능들을 통합하는 SoC(system-on-chip) 타입 디바이스일 수 있다. 나타낸 매우 특정한 실시예에서, IC(220-0)는 임베딩된 프로세서(들), 로직 및 메모리를 포함할 수 있다. 이러한 임베딩된 프로세서(들)는 본원에 설명된 바와 같은 오프로드 프로세서(들)(208) 또는 그 등가물들일 수 있다. 이러한 로직은 본원에 설명된 바와 같은 제어기 로직(212), 메모리 인터페이스(204) 및/또는 아비터 로직(206) 또는 그 등가물들 중 임의의 것일 수 있다. 이러한 메모리는 본원에 설명된 바와 같은 로컬 메모리(210), 오프로드 프로세서(들)(208)에 대한 캐시 메모리 또는 버퍼 메모리, 또는 그 등가물들 중 임의의 것일 수 있다. 로직 IC(220-1)는 IC(220-0)에 포함되지 않은 로직 기능들을 제공할 수 있다.

[0065] 도 2c는 또 다른 매우 특정한 실시예에 따른 프로세서 모듈(200-2)을 도시한다. 프로세서 모듈(200-2)은 도 2b의 프로세서 모듈과 같이, PCB 타입 기판(222)에 탑재된 IC들(220-2, -3, -4, -5)을 포함할 수 있다. 그러나, 도 2b과는 달리, 프로세서 모듈 기능들은 단일 목적 타입 IC들에 분배된다. IC(220-2)는, 오프로드 프로세서

(208)일 수 있는 프로세서 IC일 수 있다. IC(220-3)는 로컬 메모리(210), 버퍼 메모리 또는 이들의 결합들을 포함할 수 있는 메모리 IC일 수 있다. IC(220-4)는, 제어 로직(212)을 포함할 수 있고, 하나의 매우 특정한 실시예에서는, FPGA일 수 있는 로직 IC일 수 있다. IC(220-5)는 메모리 인터페이스(204) 및 아비터 로직(206)을 포함할 수 있고, 하나의 매우 특정한 실시예에서는, 또한 FPGA일 수 있는 또 다른 로직 IC일 수 있다.

[0066] 도 2b/2는 다양한 구현들 중 2개를 표현한다는 것이 이해된다. 프로세서 모듈의 다양한 기능들은 단일 SoC 타입 IC를 포함하는 임의의 적합한 수의 IC들 상에서 분배될 수 있다.

[0067] 도 2d은 매우 특정한 실시예에 따른 프로세서 모듈(200-1 또는 200-2)의 대향 면을 도시한다. 프로세서 모듈(200-3)은 도 2b의 프로세서 모듈과 같이, PCB 타입 기판(222)에 탑재된 다수의 메모리 IC들(하나가 220-6으로서 도시됨)을 포함할 수 있다. 다양한 프로세싱 및 로직 컴포넌트들은 도시된 프로세서 모듈의 대향 면 상에 탑재될 수 있다는 것이 이해된다. 메모리 IC(220-6)는 시스템의 물리 메모리 공간의 일부분을 표현하도록 구성될 수 있다. 메모리 IC들(220-6)은 다음의 기능들: 종래의 방식으로 액세스되는 시스템 메모리를 제공하는 다른 프로세서 모듈 컴포넌트들과 독립적으로 동작하는 것, 다른 프로세서 모듈 컴포넌트들에 의해 프로세싱될 수 있는 기록 데이터를 저장하는 버퍼 메모리로서 서빙하는 것, 또는 프로세서 컨텍스트 정보를 저장하기 위한 로컬 메모리로서 역할을 하는 것 중 임의의 것 또는 그 모두를 수행할 수 있다.

[0068] 도 2e는 본원에 설명된 바와 같은 프로세서 모듈들 또는 그 등가물들과 함께 메모리 버스를 파퓰레이팅(populate)할 수 있는 종래의 DIMM 모듈(즉, 그것은 단지 메모리 기능을 서빙함)을 도시한다.

[0069] 도 2f는 일 실시예에 따른 시스템(230)을 도시한다. 시스템(230)은 다수의 인-라인 모듈 슬롯들(하나가 226로서 도시됨)을 통해 액세스가능한 시스템 메모리 버스(228)를 포함할 수 있다. 실시예들에 따라, 슬롯들(226) 중 임의의 슬롯 또는 그 모두는 본원에 설명된 바와 같은 프로세서 모듈(200) 또는 그 등가물에 의해 점유될 수 있다. 모든 슬롯들(226)이 프로세서 모듈(200)에 의해 점유되지 않는 경우, 이용가능한 슬롯들은 종래의 인-라인 메모리 모듈들(224)에 의해 점유될 수 있다. 매우 특정한 실시예에서, 슬롯들(226)은 DIMM 슬롯들일 수 있다.

[0070] 일부 실시예들에서, 프로세서 모듈(200)은 하나의 슬롯을 점유할 수 있다. 그러나, 다른 실시예들에서, 프로세서 모듈은 다수의 슬롯들을 점유할 수 있다.

[0071] 일부 실시예들에서, 시스템 메모리 버스(228)는 하나 또는 둘 이상의 호스트 프로세서들 및/또는 입력/출력 디바이스(미도시)와 추가로 인터페이싱될 수 있다.

[0072] 다양한 실시예들에 따른 프로세서 모듈들이 설명되었고, 특정 실시예에 따른 그리고 메모리 버스를 통해 서버 또는 유사한 시스템과 인터페이싱할 수 있는 오프로드 프로세서 모듈의 동작들이 이제 설명될 것이다.

[0073] 도 3은 실시예에 따라 오프로드 프로세서들에서 콘텍스트 스위치들을 실행시킬 수 있는 시스템(301)을 도시한다. 도시된 예에서, 시스템(301)은 모듈 상에 로케이팅된 하나 또는 둘 이상의 계산 유닛들(하나의 계산 유닛이 300으로 도시됨)에 패킷 데이터를 전송할 수 있고, 이는 특정 실시예들에서, 기존의 메모리 모듈과 호환가능한 커넥터를 포함할 수 있다. 일부 실시예들에서, 계산 유닛(300)은 본 명세서의 실시예들에서 설명된 바와 같은 프로세서 모듈들 또는 등가물을 포함할 수 있다. 계산 유닛(300)은, 메모리 버스(316)를 통해 전송된 패킷들을 인터셉트하거나 다른 방식으로 액세스할 수 있고, 이에 제한되는 것은 아니지만 종결 또는 메타데이터 프로세싱을 포함한, 이러한 패킷들에 대한 프로세싱을 수행할 수 있다. 시스템 메모리 버스(316)는 본 명세서에서 설명된 것들 같은 시스템 메모리 버스 또는 등가물(예를 들어, 228)일 수 있다.

[0074] 도 3을 계속 참조하면, 시스템(301)은 외부 소스로부터 패킷 또는 다른 I/O 데이터를 수신할 수 있는 I/O 디바이스(302)를 포함할 수 있다. 몇몇 실시예들에서, I/O 디바이스(302)는, 네트워크 또는 다른 컴퓨터 또는 가상 머신으로부터 패킷 또는 다른 I/O 데이터를 수신하기 위해 물리적 디바이스에 의해 발생된 물리적 또는 가상 기능들을 포함할 수 있다. 도시된 매우 특정한 실시예에서, I/O 디바이스(302)는 입력 버퍼(302a)(예를 들어, DMA 링 버퍼) 및 I/O 가상화 기능(302b)을 갖는 네트워크 인터페이스 카드(NIC)를 포함할 수 있다.

[0075] 실시예들에 따르면, I/O 디바이스(302)는 패킷을 위해 필요한 메모리 동작의 상세들(즉, 판독/기록, 소스/목적지)을 포함하는 디스크립터를 기록할 수 있다. 이러한 디스크립터는 (예를 들어, 시스템(301)의 동작 시스템에 의해) 가상 메모리 위치에 할당될 수 있다. 그 다음으로, I/O 디바이스(302)는 입력 출력 메모리 관리 유닛(IOMMU)(304)과 통신하고, 입력 출력 메모리 관리 유닛(IOMMU)(304)은 IOMMU 기능(304b)을 이용하여 가상 어드레스들을 대응하는 물리적 어드레스들로 트랜슬레이팅할 수 있다. 도시된 특정 실시예에서, TLB(translation look-aside buffer)(304a)가 이러한 트랜슬레이션을 위해 이용될 수 있다. 그 다음으로, I/O 디바이스와 시스

템 메모리 위치들 사이에서 데이터를 판독 또는 기록하는 가상 기능은, 시스템(301)의 메모리 제어기(306b)를 통해 다이렉트 메모리 전달(예를 들어, DMA)을 이용하여 실행될 수 있다. I/O 디바이스(302)는 호스트 버스(312)에 의해 IOMMU(304)에 연결될 수 있다. 하나의 매우 특정한 실시예에서, 호스트 버스(312)는 PCI(peripheral interconnect) 타입 버스일 수 있다. IOMMU(304)는 중앙 프로세싱 유닛 I/O(CPUIO)(306a)에서 호스트 프로세싱 섹션(306)에 연결될 수 있다. 도시된 실시예에서, 이러한 연결(314)은 HT(HyperTransport) 프로토콜을 지원할 수 있다.

[0076] 도시된 실시예에서, 호스트 프로세싱 섹션(306)은 CPUIO(306a), 메모리 제어기(306b), 프로세싱 코어(306c) 및 대응하는 프로비저닝 에이전트(306d)를 포함할 수 있다.

[0077] 특정 실시예들에서, 계산 유닛(300)은 표준 인-라인 모듈 연결을 통해 시스템 버스(316)와 인터페이싱할 수 있고, 이는 매우 특정한 실시예들에서, DIMM 타입 슬롯을 포함할 수 있다. 도시된 실시예에서, 메모리 버스(316)는 DDR3 타입 메모리 버스일 수 있다. 대안적인 실시예들은 임의의 적절한 시스템 메모리 버스를 포함할 수 있다. 패킷 데이터는 메모리 제어기(306b)에 의해 메모리 버스(316)를 통해 DMA 슬레이브 인터페이스(310a)에 전송될 수 있다. DMA 슬레이브 인터페이스(310a)는 메모리 버스(316)를 통해 DMA 기록으로부터 캡슐화된 판독/기록 명령들을 수신하도록 적응될 수 있다.

[0078] 하드웨어 스케줄러((308b/c/d/e/h)는, 세션 메타데이터를 이용하여 플로우에 따라 착신 패킷들을 카테고리화함으로써, 착신 패킷들에 대한 트래픽 관리를 수행할 수 있다. 패킷들은 세션 우선순위에 기초하여 온보드 메모리(310b/308a/308m)에서 출력을 위해 큐잉될 수 있다. 특정 세션에 대한 패킷이 오프로드 프로세서(308i)에 의해 프로세싱될 준비가 되었다고 하드웨어 스케줄러가 결정할 때, 온보드 메모리는 그 세션으로의 콘텍스트 스위치를 위해 시그널링된다. 이러한 우선순위화 방법을 활용시, 종래의 접근방식들과 비교하여, 콘텍스트 스위칭 오버헤드가 감소될 수 있다. 즉, 하드웨어 스케줄러가 콘텍스트 스위칭 결정들을 핸들링하고 따라서 다운스트림 자원(예를 들어, 오프로드 프로세서(308i))의 성능을 최적화할 수 있다.

[0079] 앞서 유의된 바와 같이, 매우 특정한 실시예들에서, 오프로드 프로세서(308i)는 "윔피 코어(wimpy core)" 타입 프로세서일 수 있다. 일부 실시예들에 따르면, 호스트 프로세서(306c)는 "브라우니 코어(brawny core)" 타입 프로세서(예를 들어, "헤비 터치(heavy touch)" 계산 동작들을 핸들링할 수 있는 x86 또는 임의의 다른 프로세서)일 수 있다. 실시예들에 따르면, I/O 디바이스(302)가 착신 패킷들에 응답하여 호스트 프로세서 인터럽트들을 트리거하도록 구성될 수 있지만, 이러한 인터럽트들은 디스에이블될 수 있고, 이에 의해 호스트 프로세서(306c)에 대한 프로세싱 오버헤드를 감소시킨다. 일부 매우 특정한 실시예들에서, 오프로드 프로세서(308i)는 ARM, ARC, 텐실리카, MIPS, 스트롱/ARM 또는 "라이트 터치(light touch)" 동작들을 핸들링할 수 있는 임의의 다른 프로세서를 포함할 수 있다. 바람직하게, 오프로드 프로세서는, 콘텍스트 스위칭 오버헤드를 감소시키기 위해 하드웨어 스케줄러와 함께 동작하도록 최적화될 수 있는 복수의 세션들을 실행하기 위한 범용성 오퍼레이팅 시스템을 실행시킬 수 있다.

[0080] 도 3을 계속 참조하면, 동작에서, 시스템(301)은 네트워크 인터페이스를 통해 외부 네트워크로부터 패킷들을 수신할 수 있다. 패킷들은, I/O 디바이스(302)에 의해 이용되는 분류 로직 및 개략도들(schematics)에 기초하여 호스트 프로세서(306c) 또는 오프로드 프로세서(308i)를 목적지로 한다. 특정 실시예들에서, I/O 디바이스(302)는 가상화된 NIC로서 동작할 수 있고, 특정 논리적 네트워크에 대한 또는 특정 가상 MAC(VMAC) 어드레스에 대한 패킷들은 개별 큐들로 지향되고 목적지 논리적 엔티티로 전송될 수 있다. 이러한 어레인지먼트는 패킷들을 상이한 엔티티들에 전달할 수 있다. 몇몇 실시예들에서, 각각의 이러한 엔티티는, 연결된 가상 네트워크와 통신하기 위해 이용하는 가상 디바이스 모델, 가상 드라이버를 가질 수 있다.

[0081] 실시예들에 따르면, 트래픽을 특정 메모리 어드레스들로 재지향시키기 위해 다수의 디바이스들이 이용될 수 있다. 그러므로, 네트워크 디바이스들 각각은, 패킷들을 논리적 엔티티의 메모리 위치에 전달하는 것처럼 동작한다. 그러나, 실제로는, 이러한 패킷들은, 이들이 하나 또는 둘 이상의 오프로드 프로세서들(예를 들어, 308i)에 의해 핸들링될 수 있는 메모리 어드레스들로 전달된다. 특정 실시예들에서, 이러한 전달들은 물리적 메모리 어드레스들에 대한 것이고, 따라서, 논리적 엔티티들은 프로세싱으로부터 제거될 수 있고, 호스트 프로세서는 이러한 패킷 핸들링으로부터 자유로울 수 있다.

[0082] 따라서, 실시예들은 특정 네트워크 데이터가 공급될 수 있는 메모리 "블랙 박스(black box)"를 제공하는 것으로서 개념화될 수 있다. 이러한 메모리 블랙 박스는 데이터를 핸들링(예를 들어, 데이터를 프로세싱)하고 이러한 데이터가 요청될 때 다시(back) 응답할 수 있다.

[0083] 도 3을 계속 참조하면, 일부 실시예들에 따라, I/O 디바이스(302)는 네트워크로부터 또는 컴퓨팅 디바이스로부터 데이터 패킷들을 수신할 수 있다. 데이터 패킷들은, 예를 들어, 전송 프로토콜 넘버, 소스 및 목적지 포트 넘버들, 소스 및 목적지 IP 어드레스들을 포함한 특정한 특징들을 가질 수 있다. 데이터 패킷들은, 데이터 패킷들의 분류 및 관리를 돕는, 프로세싱되는 메타데이터(308d)를 더 가질 수 있다.

[0084] I/O 디바이스(302)는 이에 제한되는 것은 아니지만, PCI(peripheral component interconnect) 또는 PCIe(PCI express) 버스(예를 들어, 312)를 통해 호스트 마더보드와 연결되는 PCI 및/또는 PCIe 디바이스들을 포함할 수 있다. I/O 디바이스들의 예들은 네트워크 인터페이스 제어기(NIC), 호스트 버스 어댑터, 융합된 네트워크 어댑터, ATM 네트워크 인터페이스 등을 포함한다.

[0085] 다수의 논리적 엔티티들이 동일한 I/O 디바이스(302)에 액세스하도록 허용하는 추상적 개념의 방식을 제공하기 위해, I/O 디바이스는 다수의 가상 디바이스들을 제공하도록 가상화될 수 있고, 다수의 가상 디바이스들 각각은 물리적 I/O 디바이스의 기능들 중 일부를 수행할 수 있다. 실시예에 따른 IO 가상화 프로그램(예를 들어, 302b)은 트래픽을 상이한 메모리 위치들로 (그리고 따라서, 메모리 버스 상의 모듈들에 부착된 상이한 오프로드 프로세서들로) 재지향시킬 수 있다. 이를 달성하기 위해, I/O 디바이스(302)(예를 들어, 네트워크 카드)는: 다수의 가상 기능(VF) 인터페이스들 및 입력/출력 가상화(IOV) 아키텍처들(예를 들어, 싱글-루트 IOV)을 지원하는 제어 기능(CF)을 포함하는 여러 기능 부분들로 파티셔닝될 수 있다. 각각의 가상 기능 인터페이스에는 전용의 사용을 위한 실행시간 동안 자원들이 제공될 수 있다. CF 및 VF의 예들은 단일 루트 I/O 가상화 또는 멀티-루트 I/O 가상화 아키텍처와 같은 방식들 하에서 물리적 기능 및 가상 기능들을 포함할 수 있다. CF는 가상 자원들을 셋업 및 관리하는 물리적 자원들로서 동작한다. CF는 또한, 제대로 된(full-fledged) IO 디바이스로서 동작할 수 있다. VF는, 다수의 논리적 엔티티들/다수의 메모리 구역들과의 통신을 위해 가상 디바이스의 추상적 개념을 제공하는 것을 담당한다.

[0086] 호스트 프로세서(306c) 상에서 구동하는 동작 시스템/하이퍼바이저/가상 머신들 중 임의의 가상 머신들/사용자 코드는, 디바이스 모델, VF 드라이버, 및 CF를 위한 드라이버를 이용하여 로딩될 수도 있다. 디바이스 모델은, 생성된 다수의 VF들 각각을 호스트 프로세서(306c)가 인지하기 위한 물리적 디바이스의 에뮬레이션을 생성하기 위해 사용될 수도 있다. 디바이스 모델은, VF 드라이버(가상 IO 디바이스와 상호작용하는 드라이버)에 특정한 타입의 물리적 디바이스와 상호작용한다는 인상을 주기 위해 수차례 복제될 수도 있다.

[0087] 예를 들어, 특정한 디바이스 모듈이 Intel® Ethernet Converged Network Adapter(CNA) X540-T2와 같은 네트워크 어댑터를 에뮬레이팅하는데 사용됨으로써 I/O 디바이스(302)가 자신이 그러한 어댑터와 상호작용한다고 믿을 수도 있다. 그러한 경우에서, 가상 기능들 각각은, 상기 CNA의 기능들을 지원할 능력을 가질 수도 있는데, 즉, 물리적 기능들 각각이 그러한 기능을 지원할 수 있어야 한다. 디바이스 모델 및 VF 드라이버는 특권 또는 비-특권 모드 중 어느 하나에서 구동될 수 있다. 몇몇 실시예들에서, 디바이스 모델 및 VF 드라이버에 대응하는 코드를 누가 호스팅/구동하는지에 관한 어떠한 제한도 존재하지 않는다. 그러나, 코드는, VF 드라이버 및 디바이스 모델의 다수의 카피들을 생성할 능력을 가짐으로써 상기 I/O 인터페이스의 다수의 카피들이 생성될 수 있게 한다.

[0088] 애플리케이션 또는 프로비저닝 에이전트(306d)는, 커널에서 구동하는 애플리케이션/사용자 레벨 코드의 일부로서, 실행시간 동안 각각의 VF에 대해 가상의 I/O 어드레스 공간을 생성하고, 그 공간에 물리적 어드레스 공간의 일부를 할당할 수도 있다. 예를 들어, VF 드라이버를 핸들링하는 애플리케이션이 VF 드라이버가 메모리 어드레스들 0xaaaa 내지 0xffff로부터 또는 그 메모리 어드레스들에 패킷들을 판독 또는 기입하도록 명령하면, 디바이스 드라이버는, 큐 엔트리들이 채워짐에 따라 동적으로 변하는 헤드 및 테일 포인터들을 갖는 디스크립터 큐 내에 I/O 디스크립터들을 기입할 수도 있다. 데이터 구조는, 링 구조(302a) 또는 해쉬 테이블을 포함하지만 이에 제한되지 않는 다른 타입들을 또한 가질 수도 있다.

[0089] VF는, 드라이버에 의해 포인팅된 어드레스 위치로부터 데이터를 판독하거나 그 어드레스에 데이터를 기입할 수 있다. 또한, 드라이버에 할당된 어드레스 공간으로의 데이터 전달의 완료 시, 상기 네트워크 패킷들을 핸들링하기 위해 호스트 프로세서에 일반적으로 트리거링되는 인터럽트들이 디스에이블링될 수 있다. 특정한 I/O 공간을 디바이스에 할당하는 것은, 상기 IO 공간을 점유된 특정한 물리적 메모리 공간에 할당하는 것을 포함할 수 있다.

[0090] 다른 실시예에서, 디스크립터가 인커밍 패킷들을 핸들링하기 위한 특정 데이터 구조와 연관되는 경우, 디스크립터는 기입 동작만을 포함할 수도 있다. 또한, 인커밍 데이터 구조의 엔트리들 각각에 대한 디스크립터는, 모든 데이터 기입을 특정한 메모리 위치로 재지향시키기 위해 일정할 수도 있다. 대안적인 실시예에서, 연속적인 엔

트리들에 대한 디스크립터는, 인커밍 패킷들을 연속적인 메모리 위치들로 지향시키기 위해 메모리 내의 연속적인 엔트리들을 포인팅할 수도 있다.

[0091] 대안적으로, 상기 운영 시스템은, VF 드라이버들을 지원하는 애플리케이션에 대한 정의된 물리적 어드레스 공간을 생성하고, 애플리케이션 또는 프로비저닝 에이전트(306d)에 가상 메모리 어드레스 공간을 할당할 수도 있으며, 그에 의해, 상기 가상 어드레스와 물리적 어드레스 공간 사이에 각각의 가상 기능에 대한 맵핑을 생성한다. 가상 메모리 어드레스 공간과 물리적 메모리 공간 사이의 상기 맵핑은, IOMMU 테이블들(예를 들어, TLB(304a))에 저장될 수도 있다. 메모리 판독들 또는 기입들을 수행하는 애플리케이션은, 상기 가상 기능에 가상 어드레스들을 공급할 수도 있으며, 호스트 프로세서 OS는 물리적 메모리 위치의 특정 부분을 그러한 애플리케이션에 할당할 수도 있다.

[0092] 대안적으로, VF는, 예를 들어, 직접 메모리 액세스(DMA) 판독 또는 기입 동작의 일부일 수도 있는 판독 및 기입과 같은 요청들을 생성하도록 구성될 수도 있다. 가상의 어드레스들은, IOMMU(304)에 의해 그들의 대응하는 물리적 어드레스들로 변환되며, 물리적 어드레스들은 액세스를 위해 메모리 제어기에 제공될 수도 있다. 즉, IOMMU(304)는, I/O 디바이스들에 의해 소싱된 메모리 요청들을 변형하여, 요청된 가상 어드레스를 물리적 어드레스로 변경할 수도 있으며, 메모리 요청은 메모리 액세스를 위해 메모리 제어기로 포워딩될 수도 있다. 메모리 요청은, HyperTransport(314)와 같은 프로토콜을 지원하는 버스(314)를 통해 포워딩될 수도 있다. 그러한 경우들에서, VF는, 가상 메모리 어드레스를 IOMMU(304)에 공급함으로써 직접 메모리 액세스를 수행할 수도 있다.

[0093] 대안적으로, 상기 애플리케이션은, VF가 허용하는 경우, 물리적 어드레스를 VF 디스크립터들로 직접 코딩할 수도 있다. VF가 호스트 프로세서(306c)에 의해 사용되는 형태의 물리적 어드레스들을 지원할 수 없으면, VF 디바이스에 의해 지원되는 하드웨어 사이즈를 갖는 어퍼처가 디스크립터로 코딩되어 VF가 디바이스의 타겟 하드웨어 어드레스를 통지받을 수도 있다. 어퍼처에 전달되는 데이터는, 변환 테이블에 의해 시스템 메모리 내의 정의된 물리적 어드레스 공간으로 맵핑될 수도 있다. DMA 동작들은, DMA 동작들을 수행하기 위해 I/O 디바이스들을 직접 또는 간접적으로 프로그래밍하는 프로세서들에 의해 실행되는 소프트웨어에 의해 개시될 수도 있다.

[0094] 계속 도 3을 참조하면, 특정한 실시예들에서, 계산 유닛(300)의 부분들은 하나 이상의 FPGA들로 구현될 수도 있다. 도 3의 시스템에서, 계산 유닛(300)은, 그 내에 DMA 슬레이브 디바이스 모듈(310a) 및 아비터(310f)가 형성될 수 있는 FPGA(310)를 포함할 수 있다. DMA 슬레이브 모듈(310a)은, DMA 판독/기입 요청들에 응답할 수 있는 메모리 버스(316)에 부착하기에 적절한 임의의 디바이스일 수 있다. 대안적인 실시예들에서, DMA 슬레이브 모듈(310a)은, 메모리 버스(316)를 통한 블록 데이터 전달들이 가능한 다른 인터페이스일 수 있다. DMA 슬레이브 모듈(310a)은, ('메모리'로부터 또는 주변기기로부터 판독을 수행하는 경우) DMA 제어기로부터 데이터를 수신하거나, 또는 (DMA 슬레이브 모듈(310a)을 통해 기입 명령을 수행하는 경우) DMA 제어기에 데이터를 전달하는 것이 가능할 수 있다. DMA 슬레이브 모듈(310a)은, (예를 들어, 패킷 또는 데이터 버스트와 같은 DDR 데이터 송신의 형태로) 메모리 버스를 통해 캡슐화된 DMA 판독 및 기입 명령들을, 또는 대응하는 메모리 버스를 통해 전송될 수 있는 임의의 다른 포맷을 수신하도록 적응될 수도 있다.

[0095] DMA 슬레이브 모듈(310a)은, 메모리 R/W 패킷으로부터의 DMA 판독/기입 명령을 재구성할 수 있다. DMA 슬레이브 모듈(310a)은, 데이터 판독들/데이터 기입들의 형태의 이들 명령에 대해 DMA 마스터에 응답하도록 적응될 수도 있으며, DMA마스터는, PCIe 버스의 경우 주변기기 디바이스 내에 하우징될 수 있거나, 또는 ISA버스의 경우 시스템 DMA 제어기일 수 있다.

[0096] DMA 디바이스(310a)에 의해 수신되는 I/O 데이터는, 그 후, 중재를 위해 큐잉될 수 있다. 중재는, 상이한 플로우들의 패킷들을 스케줄링하는 프로세스를 포함함으로써, 파라미터들의 넘버에 기초하여 이용가능한 대역폭에 대한 액세스를 제공받을 수 있다. 일반적으로, 아비터(310f)는 하나 이상의 요청자들에 대해 리소스 액세스를 제공한다. 다수의 요청자들이 액세스를 요청하면, 아비터(310f)는 어느 요청자가 액세서가 되는지를 결정하고, 그 후, 액세서로부터의 데이터를 리소스 인터페이스에 전달할 수 있으며, 다운스트림 리소스가 데이터 상에서 실행을 시작할 수 있다. 데이터가 리소스에 완전히 전달되고, 리소스가 실행을 완료한 이후, 아비터(310f)는 상이한 요청자에게 제어를 전달할 수 있으며, 이러한 사이클이 모든 이용가능한 요청자들에 대해 반복된다. 도 3의 실시예에서, 아비터(310f)는 계산 유닛(300)의 다른 부분들(예를 들어, 308)에 데이터의 인커밍을 통지할 수 있다.

[0097] 대안적으로, 계산 유닛(300)은, 2010년 10월 12일자로 Dalal에 발행된 미국 특허 7,813,283에 나타낸 중재 방식을 이용할 수 있으며, 상기 특허의 내용들은 본 명세서에 인용에 의해 포함된다. 당업계에 알려져 있는 다른

적절한 중재 방식들이 본 명세서의 실시예들에서 구현될 수 있다. 대안적으로, 본 발명의 중재 방식은 OpenFlow 스위치 및 OpenFlow 제어기를 사용하여 구현될 수도 있다.

[0098] 도 3의 매우 특정한 실시예에서, 계산 유닛(300)은, 통지/프리페치 회로들(310c)을 더 포함할 수 있으며, 통지/프리페치 회로들(310c)은, DMA 슬레이브 모듈(310a)에 응답하여 버퍼 메모리(310b) 내에 저장되고 그리고 아비터(310f)에 의해 중재된 바와 같은 데이터를 프리페칭할 수 있다. 추가적으로, 아비터(310f)는, 메모리 맵핑된 I/O 진입 경로(301e) 및 출구 경로(310g)를 통하여 계산 유닛(300)의 다른 부분들에 액세스할 수 있다.

[0099] 도 3을 참조하면, 하드웨어 스케줄러는 인커밍 패킷들의 트래픽 관리를 구현하기 위한 스케줄링 회로(308b/n)를 포함할 수 있다. 특정 트래픽 클래스에 관련되거나, 특정 애플리케이션에 속하거나, 또는 특정 소켓으로 흐르는, 특정 소스로부터의 패킷들은, 세션 플로우의 부분으로 지칭되고, 세션 메타데이터를 사용하여 분류된다. 그러한 분류는 분류기(308e)에 의해 수행될 수 있다.

[0100] 몇몇 실시예들에서, 세션 메타데이터(308d)는, 패킷들이 우선순위화 및 스케줄링되게 하고 그래서 인입 패킷들이 그들의 세션 메타데이터에 기초하여 리오더링될 수 있게 하는 기준으로서의 역할을 할 수 있다. 패킷들의 이러한 리오더링은 하나 또는 그 초과의 버퍼들에서 발생할 수 있고, 이러한 플로우들의 트래픽 형상을 변경할 수 있다. 이러한 우선순위화를 위해 선택된 스케줄링 규율, 또는 트래픽 관리(TM)는, 지연(버퍼링), 트래픽의 버스팅(버퍼링 및 버스팅), 트래픽의 스무딩(버퍼링, 및 플로우들을 레이트 제한하는 것), 트래픽을 드롭핑하는 것(버퍼를 고갈시키는 것을 방지하기 위하여 폐기할 데이터를 선택하는 것), 지연 지터(플로우의 셀들을 상이한 양들만큼 시간적으로 시프팅하는 것)를 통해 그리고 접속을 허용하지 않음으로써(예컨대, 부가적인 플로우들의 서비스 레벨 합의에 대해서는 기존의 SLA들을 동시에 보증할 수 없음), 플로우들 및 마이크로플로우들의 트래픽 형상에 영향을 끼칠 수 있다.

[0101] 실시예들에 따라, 계산 유닛(300)은 스위치 패브릭의 일부로서의 역할을 할 수 있고 깊이-제한된 출력 큐들을 트래픽 관리에 제공할 수 있으며, 이 깊이-제한된 출력 큐들로의 액세스는 스케줄링 회로(308b/n)에 의해 조정된다. 인입 플로우들에 대한 트래픽 관리를 제공하기 위한 스케줄링 규율을 사용하여 이러한 출력 큐들이 관리된다. 이러한 큐들 각각에 큐잉된 세션 플로우들은 출력 포트를 통해 다운스트림 네트워크 엘리먼트에 전송될 수 있다.

[0102] 통상적인 트래픽 관리는, 자신이 다운스트림 엘리먼트들에 대해 이미 갖고 있는 SLA 합의들을 충족시키기 위해서를 제외하고는, 상기 다운스트림 엘리먼트들에 의한 데이터의 핸들링 및 관리를 고려하지 않음이 주목된다.

[0103] 그에 반해서, 실시예들에 따라, 스케줄러 회로(308b/n)는 출력 큐들 각각에 우선순위를 할당할 수 있고, 이러한 큐들에서 세션 플로우들의 지속을 유지시키기 위해 인입 패킷들의 리오더링을 수행할 수 있다. 스케줄러 회로(308b/n)는 오프로드 프로세서(308i) 상에서 실행되는 범용성 운영체제(OS)(308j)로의 이러한 지속적 세션들 각각의 스케줄링을 제어하는데 사용될 수 있다. 위에서 정의된 바와 같이, 특정 세션 플로우의 패킷들은 특정 큐에 속할 수 있다. 스케줄러 회로(308b/n)는 이러한 큐들의 우선순위화를 제어할 수 있고, 따라서 이러한 큐들은 다운스트림에 위치된 범용성(GP) 프로세싱 자원(예컨대, 오프로드 프로세서(308i))에 의한 핸들링을 위해 조정된다. 다운스트림 프로세서(308i) 상에서 실행되는 OS(308j)는 프로세서 주기들 및 메모리와 같은 실행 자원들을, 자신이 현재 핸들링하고 있는 특정 큐에 할당할 수 있다. OS(308j)는 그 특정 큐에 대해 스레드 또는 스레드들의 그룹을 추가로 할당할 수 있고, 따라서 그 특정 큐는 범용성 프로세싱 엘리먼트(308i)에 의해 별개의 엔티티로서 별도로 핸들링된다. 다수의 세션들이 GP 프로세싱 자원 상에서 실행될 수 있다는 사실(각각이 스케줄러 회로에 의해 설정된 큐에 있는 특정 세션 플로우로부터의 데이터를 핸들링함)은 스케줄러 및 다운스트림 자원(예컨대, 308i)을 단단히 통합시킨다. 이는 트래픽 관리 및 스케줄링 회로 및 범용성 프로세싱 자원(308i)에 걸쳐 세션 정보의 지속을 초래할 수 있다.

[0104] 세션들 각각에 대한 전용 컴퓨팅 자원들(예컨대, 308i), 메모리 공간 및 세션 콘텍스트 정보는 범용성 프로세서(308i)에서 세션 플로우들 각각을 핸들링, 프로세싱 및/또는 종료하는 방식을 제공할 수 있다. 스케줄러 회로(308b/n)는 다운스트림을 스케줄링하기 위해 세션 플로우들을 큐잉하기 위해서 실행 자원의 이러한 기능을 활용할 수 있다. 스케줄러 회로(308b/n)는 실행 자원(들)(예컨대, 308i)의 상태, 실행 자원 상에서 실행되는 현재 세션; 실행 자원에 할당된 메모리 공간, 프로세서 캐시에서의 세션 콘텍스트의 위치에 관해 통보받을 수 있다.

[0105] 실시예들에 따라, 스케줄러 회로(308b/n)는 실행 자원들을 하나의 상태로부터 다른 상태로 변경시키기 위한 스위칭 회로들을 더 포함할 수 있다. 스케줄러 회로(308b/n)는, 다운스트림 실행 자원으로 스위칭되도록 준비되어 있는 큐들 사이를 조정하기 위해 이러한 능력을 사용할 수 있다. 추가로, 다운스트림 실행 자원은, 자원들

사이의 콘텍스트 스위치와 연관된 오버헤드 및 페널티를 감소시키도록 최적화될 수 있다. 이는, 스케줄러 회로(308b/n)에 의해, 큐들 사이의 심리스 스위칭, 및 결과적으로 상이한 세션들로서 실행 자원에 의한 그들의 실행을 수행하기 위해 추가로 활용될 수 있다.

[0106] 실시예들에 따라, 스케줄러 회로(308b/n)는 다운스트림 프로세싱 자원 상에서 상이한 세션들을 스케줄링할 수 있는데, 콘텍스트 스위치들 동안의 오버헤드를 감소시키기 위해 두 개가 조정되어 동작된다. 서비스들의 레이턴시 및 엔지니어링 계산 이용가능성을 감소시킬 때 중요한 인자는, 네트워크 큐잉과 동기화된 하드웨어 콘텍스트 스위칭일 수 있다. 실시예들에서, 큐가 트래픽 관리자에 의해 선택될 때, 파이프라인은 대응하는 자원(예컨대, 308i)의 캐시(예컨대, L2 캐시)의 스와핑 인(swapping in)을 조정하고, 리어셈블링된 I/O 데이터를 실행 프로세스의 메모리 공간으로 전달한다. 특정 경우들에서, 어떠한 패킷들도 큐에서 계류중이지 않지만, 이전 패킷들에 서비스하기 위해 계산이 여전히 계류중이다. 일단 이러한 프로세스가 스와핑된 데이터의 바깥에 메모리 레퍼런스를 만들면, 스케줄러 회로(308b/n)는 스레드를 스케줄링하는 것을 계속하기 위해 I/O 디바이스(302)로부터 큐잉된 데이터를 인에이블링할 수 있다.

[0107] 몇몇 실시예들에서, 데이터를 갖지 않는 프로세스에 공평한 큐잉을 제공하기 위해, 최대 콘텍스트 크기가 프로세싱되는 데이터로서 가정될 수 있다. 이러한 방식으로, 큐가 계산 자원과 네트워크 대역폭 자원 중 더 큰 것으로서 프로비저닝될 수 있다. 그러나 하나의 매우 특정한 예로서, 계산 자원이 800㎒에서 실행되는 ARM A9 프로세서일 수 있는 반면에, 네트워크 대역폭은 3Gbps의 대역폭일 수 있다. 한쪽으로 치우친 성질의 이러한 비율이 주어지면, 실시예들은, 많은 병렬 세션들(따라서, 세션 특정 데이터에 대한 하드웨어의 프리페칭이 호스트 프로세서 로드의 많은 부분을 오프로딩함)을 갖고 데이터의 최소 범용성 프로세싱을 갖는 계산을 활용할 수 있다.

[0108] 따라서, 몇몇 실시예들에서, 스케줄러 회로(308b/n)는, 라인 레이트 속도들의 아웃고잉 큐들 사이를 조정하는 것으로서가 아니라 매우 고속의 종단 세션들 사이를 조정하는 것으로서 개념화될 수 있다. 범용성 OS를 비롯한 단계들의 파이프라인에 걸친 세션들의 밀착도(stickiness)는, 스케줄러 회로가 이러한 파이프라인의 이러한 단계들 중 임의의 단계 또는 단계들 전부를 최적화하는 것일 수 있다.

[0109] 대안적으로, 스케줄링 방식은 2010년 7월 20일자로 Dalal에 발행된 미국 특허 번호 7,760,715에 나타난 바와 같이 사용될 수 있고, 이 특허는 본원에 인용에 의해 통합된다. 이 방식은, 과잉 선택된 플로우에 특정한 다른 자원의 다운스트림 혼잡을 방지하기 위해 또는 특정 플로우들에 대해 서비스 계약들을 강제하기 위해 플로우들을 레이트 제한하는 것이 바람직할 때 유용할 수 있다. 실시예들은 다운스트림 자원들, 예컨대 심리스하게 강제될 수 있는 범용성 OS의 서비스 계약들을 허용하는 조정 방식을 포함할 수 있다.

[0110] 여전히 도 3을 참조하면, 본원의 실시예들 또는 균등물들에 따른 하드웨어 스케줄러는, 세션 메타데이터에 기초하여 인입 패킷 데이터의 세션 플로우들로의 분류를 제공할 수 있다. 하드웨어 스케줄러는, 플로우들이 오프로드 프로세서들 상에서 별개의 프로세싱 엔티티들로서 조정 및 큐잉되기 이전에, 이러한 플로우들의 트래픽 관리를 추가로 제공할 수 있다.

[0111] 몇몇 실시예들에서, 오프로드 프로세서들(예컨대, 308i)은 상이한 애플리케이션 또는 전송 세션들의 패킷들을 핸들링할 수 있는 범용성 프로세싱 유닛들일 수 있다. 이러한 오프로드 프로세서들은 범용성 명령들을 실행할 수 있는 저전력 프로세서들일 수 있다. 오프로드 프로세서들은 ARM, ARC, Tensilica, MIPS, StrongARM 또는 본원에 설명되는 기능들을 서빙하는 임의의 다른 프로세서를 비롯한 임의의 적절한 프로세서일 수 있지만, 이들로 제한되지는 않는다. 이러한 오프로드 프로세서들은 그들 상에서 실행되는 범용성 OS를 갖고, 범용성 OS는 상이한 스레드들 또는 스레드들의 그룹 사이의 콘텍스트 스위칭과 연관된 페널티를 감소시키도록 최적화된다.

[0112] 그에 반해서, 호스트 프로세서들 상에서의 콘텍스트 스위치들은, 레지스터 저장 영역, 캐시의 프로세스 콘텍스트, 및 TLB 엔트리들이 그들이 무효화되거나 또는 덮어쓰기되는 경우 복구될 것을 요구하는 계산 집약적 프로세스들일 수 있다. 호스트 프로세싱 시스템들에서의 명령 캐시 미스(miss)들은 파이프라인 스톨들을 유도할 수 있고, 데이터 캐시 미스들은 동작 스톨을 유도하며, 이러한 캐시 미스들은 프로세서 효율을 감소시키고 프로세서 오버헤드를 증가시킨다.

[0113] 또한, 그에 반해서, 스케줄러 회로(308b/n)와 연관되어 오프로드 프로세서들(308i) 상에서 실행되는 OS(308j)는, 그것 상에서 실행되는 상이한 프로세싱 엔티티들 사이에서 발생하는 콘텍스트 스위치 오버헤드를 감소시키기 위해 함께 동작할 수 있다. 실시예들은 오프로드 프로세서(308i) 상에서 스케줄러 회로와 OS 사이의 협력 메커니즘을 포함할 수 있고, OS는 캐시에서 물리적으로 연속적이 되도록(세션 히프 및 스택에 대한 물

리적으로 컬러링된 할당기) 세션 콘텍스트를 셋업하고; 그 다음, 세션 초기화 시 세션 컬러, 크기, 및 시작 물리적 어드레스를 스케줄러 회로에 통신한다. 실제 콘텍스트 스위치 동안, 스케줄러 회로는, 이러한 파라미터들을 사용함으로써 캐시에서 세션 콘텍스트를 식별할 수 있고, 외부 저 레이턴시 메모리(예컨대, 308g)로의 이러한 콘텐츠의 벌크 전송을 개시할 수 있다. 부가하여, 스케줄러 회로는 구 세션의 콘텍스트가 로컬 메모리(308g)에 저장되었던 경우 구 세션의 프리페치를 관리할 수 있다. 특정 실시예들에서, 로컬 메모리(308g)는 저 레이턴시 메모리, 예컨대 감소된 레이턴시 동적 랜덤 액세스 메모리(RLDRAM)일 수 있지만, 이는 그러나 하나의 매우 특정한 실시예로서이다. 따라서, 실시예들에서, 세션 콘텍스트는 캐시에서 별도로 식별될 수 있다.

[0114] 몇몇 실시예들에서, 콘택스트 크기는 빠른 스위칭 속도를 보장하도록 제한될 수 있다. 부가적으로 또는 대안적으로, 실시예들은 로컬 메모리(308g)에 세션 콘택스트를 전달하기 위한 벌크 전달 매커니즘을 포함할 수 있다. 그 내부에 저장된 캐시콘텐츠들은 이어서, 다시 이전의 세션으로의 콘택스트 스위칭 동안 리트리브되고 프리패치될 수 있다. 상이한 콘택스트 세션 데이터는 빠른 리트리브를 위해 로컬 메모리(308g) 내에서 식별되고 및/또는 태깅될 수 있다. 위에서 언급된 바와 같이, 하나의 오프로드 프로세서에 의해 저장되는 콘택스트는 상이한 오프로드 프로세서에 의해 리콜될 수 있다.

[0115] 도 3의 매우 특정한 실시예에서, 다수의 오프로드 프로세싱 코어들이 계산 FPGA(308)에 통합될 수 있다. 다수의 계산용 FPGA들은 다른 FPGA(310)의 중재기 회로들에 의해 중재될 수 있다. 계산용 FPGA들(예를 들어, 308) 및 아비터 FPGA들(예를 들어, 310)의 결합은 "XIMM" 모듈들 또는 "Xockets DIMM 모듈들"(예를 들어, 계산 유닛(300))으로서 지칭된다. 특정한 애플리케이션들에서, 이들 XIMM 모듈들은 오프로드 프로세서들 상의 다수의 세션들의 실행을 중개하는 통합된 트래픽 및 스레드 관리 회로들을 제공할 수 있다.

[0116] 도 3은 오프로드 프로세서 터널 연결(308k)은 물론 메모리 인터페이스(308m) 및 액세스 유닛(3081)(ACP(accelerator coherency port)일 수 있음)를 또한 도시한다. 메모리 인터페이스(308m)는 버퍼 메모리(308a)에 액세스할 수 있다. 실시예들에 따라, 시스템(301)은 오프로드 프로세서(308i)의 캐시 콘텐츠들에 액세스하도록 액세스 유닛(또는 "스누핑" 유닛)(3081)의 이용을 포함할 수 있다. 특정한 실시예들에서, 액세스되는 캐시는 L2 캐시일 수 있다. 액세스 유닛(3081)은 외부의 비-캐시 메모리(308g)로부터 오프로드 프로세서 캐시로 데이터를 로딩시키는 것은 물론, 오프로드 프로세서(308i)의 캐시 콘텐츠들을 비-캐시 메모리(308g)에 전달할 수 있는 포트 또는 다른 액세스 능력을 제공할 수 있다. 계산용 엘리먼트(300)의 부분으로서, 메모리(308g)를 형성하는 몇 개의 메모리 디바이스들(예를 들어, RAM들)이 있을 수 있다. 따라서, 메모리(308g)는 세션들의 캐시 콘텐츠들을 저장하는데 이용될 수 있다. 메모리(308g)는 하나 또는 그 초과의 낮은 레이턴시 메모리들을 포함할 수 있고 이용 가능한 L2 캐시를 보충하고 및/또는 증가시키고 세션들의 코히어런시 도메인을 연장하는 것으로서 개념화될 수 있다. 부가적인 메모리(308g) 및 액세스 유닛(3081)은, 스레드가 재개될 때 그의 이전의 워킹 세트 대부분이 캐시에 이미 존재하도록 세션의 콘택스트가 오프로드 프로세서 캐시 내로 패치 또는 프리-패치될 수 있다는 점에서, 스위칭 인된 세션들에 대해 캐시 미스들의 악영향을 감소시킬 수 있다.

[0117] 일 특정한 실시예에 따라, 세션 스위칭-아웃 시에, 오프로드 프로세서(308i) 캐시 콘텐츠들은 터널(308k)을 통해 메모리(308g)에 전달될 수 있다. 그러나 몇몇 실시예들에서, 스레드의 레지스터 세트는 스위칭-아웃의 부분으로서 메모리에 저장될 수 있고, 이에 따라 이들 레지스터 콘텐츠들은 캐시에 상주하게 될 수 있다. 그러므로, 스위칭-인의 부분으로서, 세션의 콘텐츠들이 오프로드 프로세서(308i)의 캐시 내로 프리패치되고 전달될 때, 레지스터 콘텐츠들은 스레드의 재개 시에 커널에 의해 로딩될 수 있고, 이들 로드들은 메모리(308g)로부터가 아닌, 캐시로부터 발생해야 한다. 따라서, 세션의 캐시 콘텐츠들의 주의깊은 관리를 통해, 스위치-인 시에 레지스터 세트 저장 및 복원 및 캐시 미스들로 인한 콘택스트 스위칭의 비용은 크게 감소되고, 심지어는 몇몇 최적의 경우들에서 제거되어서, 콘택스트 스위칭 오버헤드의 2개의 소스들을 제거하고 유용한 프로세싱을 재개하도록 스위칭-인된 세션에 대한 레이턴시를 감소시킨다.

[0118] 몇몇 실시예들에 따라, 액세스(또는 스누핑) 유닛(예를 들어, 3081)은 적절한 세션 콘택스트가 상주하는 캐시 내의 모든 라인들의 인덱스들을 가질 수 있다. 세션이 물리적으로 인덱싱된 캐시 내의 위치들에 걸쳐서 산재되는 경우, 다수의 어드레스 치환들은 동일한 세션에서 다수의 페이지들에 액세스하도록 요구될 것이므로 모든 세션 콘텐츠들에 액세스하는 것은 매우 부담스럽게 될 수 있다. 이에 따라, 실시예들은 세션 콘텐츠들이 물리적으로 인덱싱된 캐시 내의 인접한 위치들에서 설정되는 페이지 컬러링 방식을 포함할 수 있다. 세션 데이터에 대한 메모리 할당기는 물리적으로 인접한 페이지들로부터 할당되어서, 세션에 대한 물리적 어드레스 범위들에 걸친 제어가 있게 될 수 있다. 몇몇 실시예들에서, 이는 캐시(예를 들어, 도 1d) 내의 동일한 위치로 인덱싱하도록 가상 메모리 페이지 및 물리적 메모리 페이지를 정렬함으로써 행해진다. 대안적인 실시예들에서, 가상 및 물리적 메모리 페이지들은 물리적으로 인덱싱된 캐시 내의 동일한 위치를 인덱싱해야 하는 것이 아니라, 세션의 상

이한 페이지들이 물리적 메모리에서 인접할 수 있어서, 캐시 내의 엔트리의 크기 및 시작 인덱스의 지식은 모든 세션 데이터에 액세스하는데 충분하게 될 수 있다. 또한, 세트 크기는 세션의 크기와 동일하여서, 캐시 내의 세션 엔트리의 인덱스가 일단 알려지면, 인덱스, 크기 및 세트 컬러는 캐시로부터 외부 메모리(예를 들어, 308g)로 세션 콘텐츠들을 완전히 전달하는데 이용될 수 있다.

[0119] 실시예들에 따라, 세션의 모든 페이지들에는 오프로드 프로세서의 캐시에서 동일한 컬러가 할당될 수 있다. 특정한 실시예에서, 세션의 모든 페이지들은 정의된 컬러의 페이지 바운더리에서 시작할 수 있다. 컬러에 할당된 페이지들의 번호는 캐시 내의 세션의 크기에 기초하여 고정될 수 있다. 오프로드 프로세서(예를 들어, 308i)는 특정한 타입의 세션들을 실행하기 위해 이용될 수 있고, 각각의 세션의 크기가 미리 통지된다. 이것에 기초하여, 오프로드 프로세서는 세션 바운더리의 새로운 엔트리에서 시작할 수 있다. 오프로드 프로세서는 캐시의 세션 바운더리를 인덱싱하는 물리적 메모리 내의 페이지들을 유사하게 할당할 수 있다. 세션 바운더리에서 시작하여 전체 캐시 콘택스트가 저장될 수 있다. 현재 설명된 실시예에서, 세션의 다수의 페이지들은 물리적으로 인덱싱된 캐시에서 인접할 수 있다. 세션의 다수의 페이지들은 동일한 컬러를 가질 수 있고(즉 이들은 동일한 세트의 부분임) 인접하게 로케이팅된다. 세션의 페이지들은 세션의 베이스 인덱스로부터의 오프셋을 이용함으로써 액세스 가능하다. 캐시는 배열되고 페이지들로서가 아니라, 세션들로서 특유의 세트들로 분해될 수 있다. 하나의 세션으로부터 다른 세션으로의 이동을 위해, 메모리 할당 방식은 이들 세션들에 액세스하는데 이용된 인덱스들의 최저 비트에 대한 오프셋을 이용한다. 예를 들어, 물리적으로 인덱싱되는 캐시는 512kb의 크기를 갖는 L2 캐시일 수 있다. 캐시는 8-웨이 결합식일 수 있고, L2 캐시에서 세트 당 8개의 웨이들이 가능하다. 그러므로, L2에서 임의의 컬러 당 8개의 라인들, 또는 L2에서 각각의 컬러의 8개의 별개의 인스턴스들이 있다. 8kb의 세션 콘택스트에 있어서, 512Kb L2 캐시 내에 8개의 상이한 세션 영역들, 또는 이들 선택된 크기들을 갖는 8개의 세션 컬러들이 결국 있을 것이다.

[0120] 실시예들에 따르면, 물리적 메모리 할당기는 시간적으로 이전 세션의 캐시 엔트리/메인 메모리 엔트리에 기초하여 세션에 대응하는 컬러를 식별할 수 있다. 특정 실시예에서, 물리적 메모리 할당기는, 캐시 엔트리를 이전 세션으로 할당하기 위해 사용되는 어드레스의 3비트에 기초하여 이전 세션의 세션을 식별할 수 있다. 물리적 메모리 할당기는 새로운 세션을 메인 메모리 위치(메인 메모리 위치의 컬러는 가장 최근에 사용된 엔트리에 대한 몇개의 비교들을 통해 결정될 수 있음)로 할당할 수 있고 가장 최근에 사용된 정책에 기초하여 축출(evicted)될 상이한 컬러의 세션에 대응하는 캐시 엔트리를 발생시킬 것이다. 다른 실시예에서, 오프로드 프로세서는 다수의 코어들을 포함할 수 있다. 이러한 실시예에서, 캐시 엔트리들은 각각의 프로세서 코어에 의한 사용을 위해 잠금(locked out)될 수 있다. 예를 들어, 오프로드 프로세서가 2개의 코어들을 갖는 경우, 캐시(즉, L2 캐시) 내 캐시 라인들의 주어진 세트는 프로세서들 중에서 분리될 수 있어서, 컬러들의 수를 반으로 줄인다. 새로운 세션이 생성될 때 외부 스케줄러에 대해 세션의 컬러, 세션의 인덱스 및 세션 크기가 통신될 수 있다. 이 정보는 인입하는 세션 흐름들의 큐 관리를 위해 사용될 수 있다.

[0121] 실시예들은 또한, 세션 데이터로부터 분리된 상태로, 이러한 라인들을 캐시 내부로 잠금으로써 공유된 텍스트와 임의의 공유된 데이터를 분리시킬 수 있다. 또한, 물리적 메모리 할당기 및 물리적 컬러링 기술들이 사용될 수 있다. 별개의 공유된 데이터가 캐시 내에 위치되는 경우, 액세스 유닛(예를 들어, ACP)에 의한 이송들이 이러한 라인들을 카피하지 않는 한, 이 공유된 데이터를 캐시 내부로 잠그는 것이 가능하다. 세션 데이터에 대해 메모리를 할당하는 경우, 메모리 할당기는, 캐시 내에 존재하는 세션 데이터가 맵핑됨에 따라 물리적 컬러를 인식할 수 있다.

[0122] 캐시 및 콘텍스트 스위칭 관리 동작들에 적합한 다양한 실시예들을 설명하였으며, 특정 양상들을 예시하는 실시예를 이제 설명할 것이다.

[0123] 도 4는 일 실시예에 따른 시스템에 대한 감소된 오버헤드 콘텍스트 스위칭의 방법(400)을 도시한다. 초기화시, 세션 컬러링이 요구되는지에 대한 결정이 이루어질 수 있다(402). 이러한 결정은 OS에 의해 이루어질 수 있다. 세션 컬러링이 요구되지 않는 경우(402에서 아니오), 페이지 컬러링이 OS의 디폴트 선택에 의존하여 존재할 수도 또는 존재하지 않을 수도 있다(424).

[0124] 세션 컬러링이 요구되는 경우(402에서 예), OS는 메모리 할당기를 초기화할 수 있다(404). 메모리 할당기는 각각의 세션 엔트리를 "세션" 경계로 할당할 수 있는 캐시 최적화 기술들을 사용할 수 있다. 메모리 할당기는 각각의 세션의 시작 어드레스, 캐시에서 허용가능한 세션들의 수, 및 세션이 주어진 컬러에 대해 발견될 수 있는 컬러들의 수를 결정할 수 있다. 이러한 동작들은, 캐시 크기, 컬러들의 수 및 세션의 크기에 기초하여 이용가능한 세트들의 수를 결정하는 것을 포함할 수 있다(단계 406).

[0125] 세션에 대한 패킷이 도달하는 경우, 패킷이 현재 또는 상이한 세션에 대한 것이지 여부에 대한 결정이 이루어질 수 있다(408). 이러한 액션은 OS에 의해 수행될 수 있다. 패킷이 상이한 세션에 대한 것이라면(408에서 예), 패킷이 조기 세션으로부터 비롯된 것인지 여부에 대한 결정이 이루어질 수 있다(410). 패킷이 조기 세션으로부터 비롯된 것이 아니라면(즉, 새로운 세션임), 새로운 세션을 위한 충분한 메모리가 존재하는지를 알기 위한 결정이 이루어질 수 있다(418). 충분한 공간이 있다면(418에서 예), 새로운 세션으로의 스위치가 이루어질 수 있다(422). 이러한 액션은 세션 경계에서 새로운 세션을 할당하는 것, 그리고 (외부의 저 레이턴시 메모리일 수 있는) 콘텍스트 메모리에 대해 현재 실행 중인 프로세스의 콘텍스트를 저장하는 것을 포함할 수 있다.

[0126] 캐시 메모리가 새로운 세션에 대해 이용가능하지 않고(418에서 아니오) 그리고/또는 패킷이 조기 세션에 대한 것이 경우(410으로부터 예), 구(old)/새로운 세션의 패킷이 동일한 컬러인지에 대해 결정하기 위해서 검사가 이루어질 수 있다(412). 상이한 컬러에 대한 것이라면(412에서 아니오), 그 세션으로의 스위치가 이루어질 수 있다(단계 414), 이러한 액션은, 태스크에 대한 캐시 엔트리들을 (조기 세션 동안) 리트리빙하는 것 또는 (새로운 세션 동안) 생성하는 것을 포함할 수 있다. 이외에도, 이러한 동작은, 필요하다면, LRU 방식에 따라 캐시 엔트리들을 플러싱하는 것을 포함할 수 있다.

[0127] 구/새로운 세션의 패킷이 동일한 컬러라면(412에서 예), 컬러 압력이 초과될 수 있는지 여부에 관한 결정이 이루어질 수 있다(416). 컬러 압력이 초과될 수 있거나, 몇몇 다른 컬러의 세션이 이용가능하지 않은 경우(416에서 예, 또는 ...), 새로운 세션으로의 스위치가 이루어질 수 있다(420). 이러한 동작은 캐시 엔트리들을 생성하는 것 및 새로운 세션 컬러를 기억하는 것을 포함할 수 있다. 컬러 압력이 초과될 수 없지만 일부 다른 컬러의 세션이 이용가능한 경우(416에서 아니오 그러나 ...), 방법은 414로 진행할 수 있다.

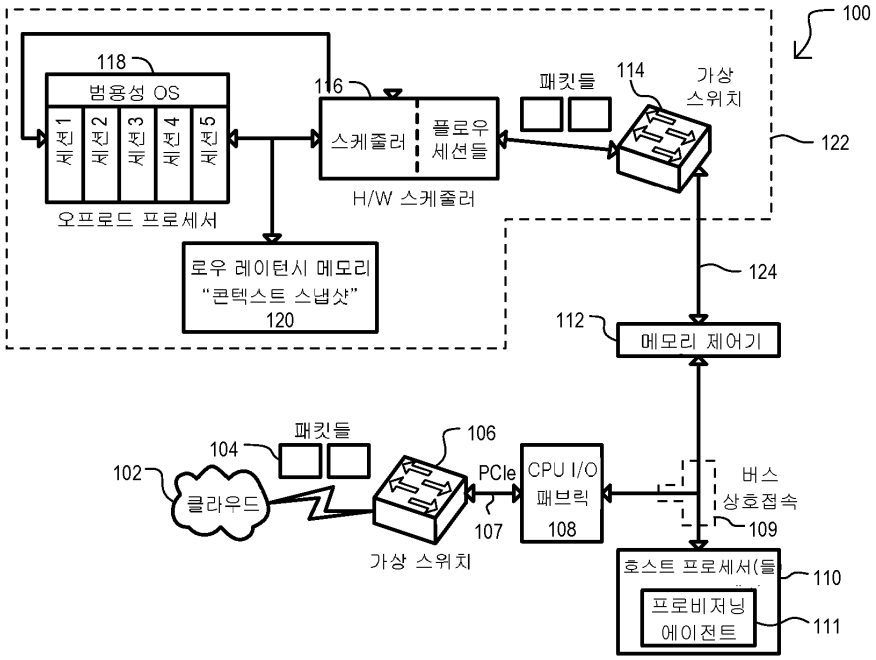
[0128] 본 발명의 예시적인 실시예들의 앞의 설명에서, 다양한 신규한 양상들 중 하나 또는 그보다 많은 것의 이해를 돕기 위해 본 개시물을 간소화하기 위해서 본 발명의 다양한 특징들은 때때로 본 개시물의 하나의 실시예, 도면, 또는 설명에서 함께 그룹화될 수 있다는 것을 인식해야 한다. 그러나, 본 개시물의 이러한 방법은, 청구된 발명이 각각의 청구범위에서 명시적으로 인용되는 것보다 더 많은 특징들을 요구한다는 의도를 반영하는 것으로서 해석되지 않아야 한다. 오히려, 다음 청구범위가 반영하는 바와 같이, 본 발명의 양상들은 하나의 상술된 실시예의 모든 특징들보다 더 적게 존재한다. 이와 같이, 다음의 상세한 설명 다음에 오는 청구범위는 이로써 명시적으로 본 상세한 설명에 포함되며, 각각의 청구항은 그 자체로 본 발명의 별개의 실시예가 된다.

[0129] 또한, 본 발명의 실시예들은 구체적으로 개시되지 않은 엘리먼트 및/또는 단계의 부재 시에 실시될 수 있다는 것을 이해한다. 즉, 본 발명의 신규한 특징들은 엘리먼트을 삭제할 수 있다.

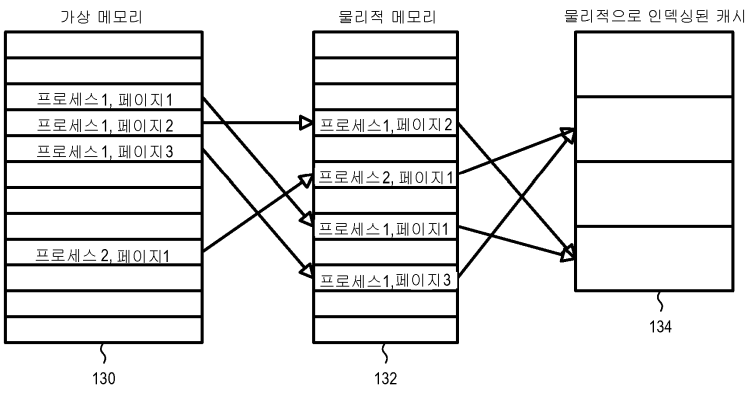
[0130] 그에 따라, 본원에 제시된 특정 실시예들의 다양한 양상들이 상세하게 설명되었지만, 본 발명은 본 발명의 정신 및 범위로부터 벗어나지 않고 다양한 변경들, 치환들, 대안들이 가능할 수 있다.

## 도면

### 도면1a

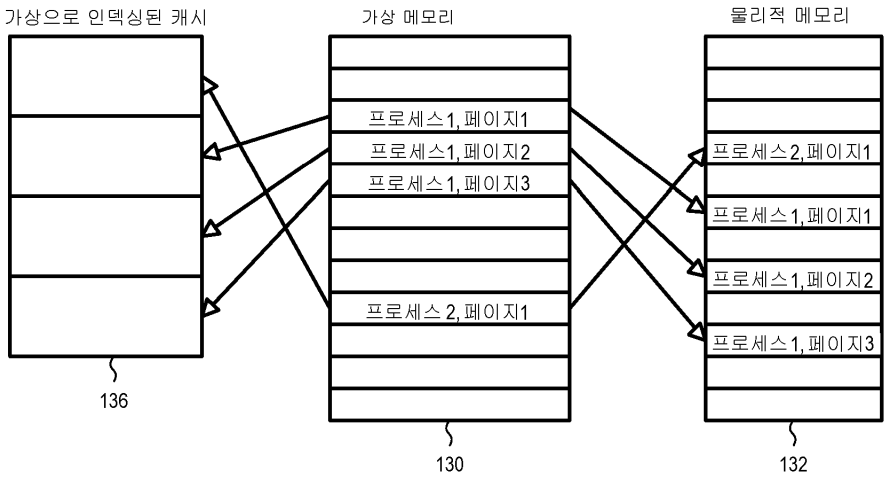



### 도면1b



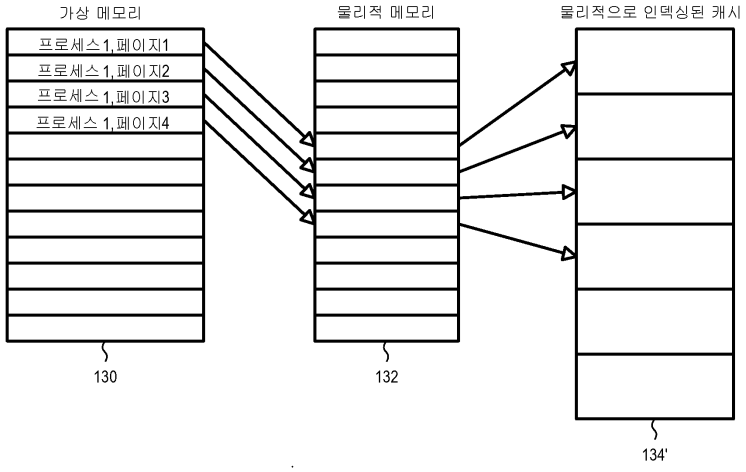

(관련 기술)

도면1c

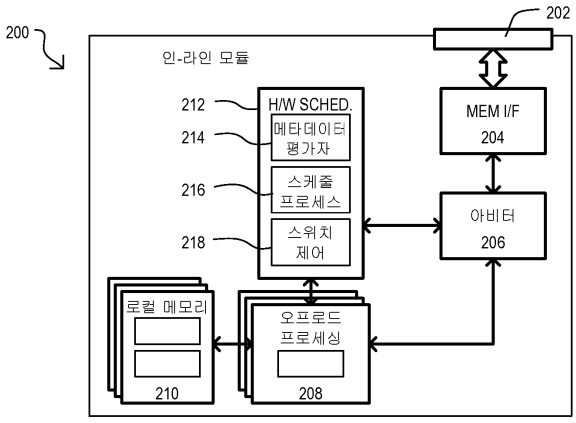



(관련 기술)

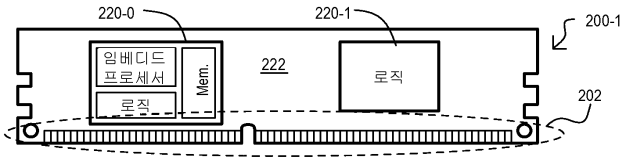
도면1d



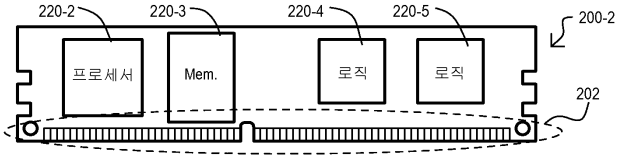

도면2a



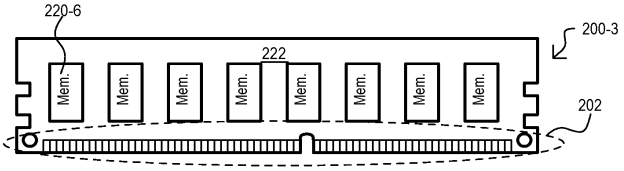
도면2b



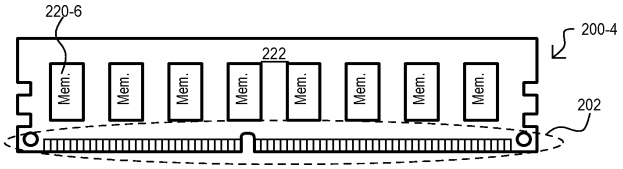

도면2c



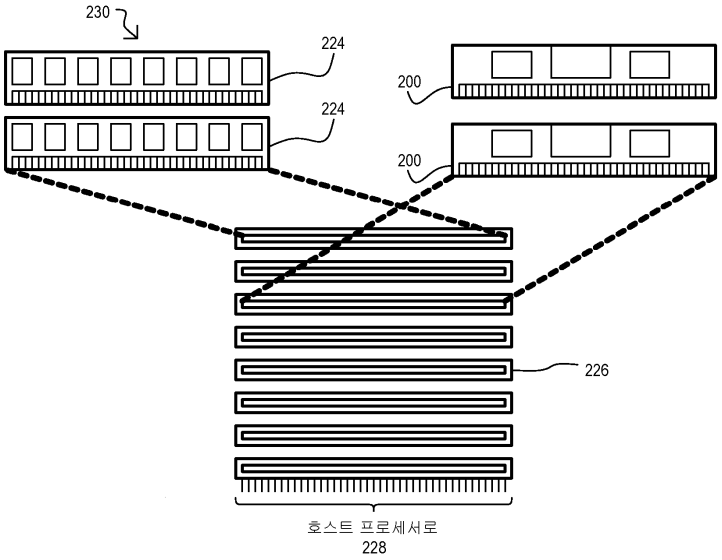

도면2d



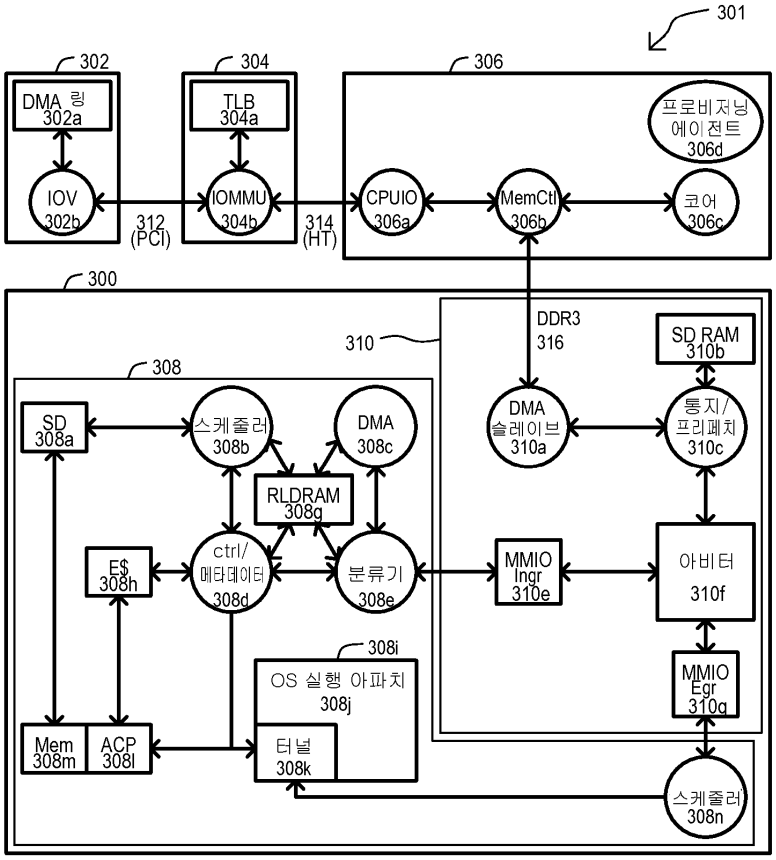

도면2e



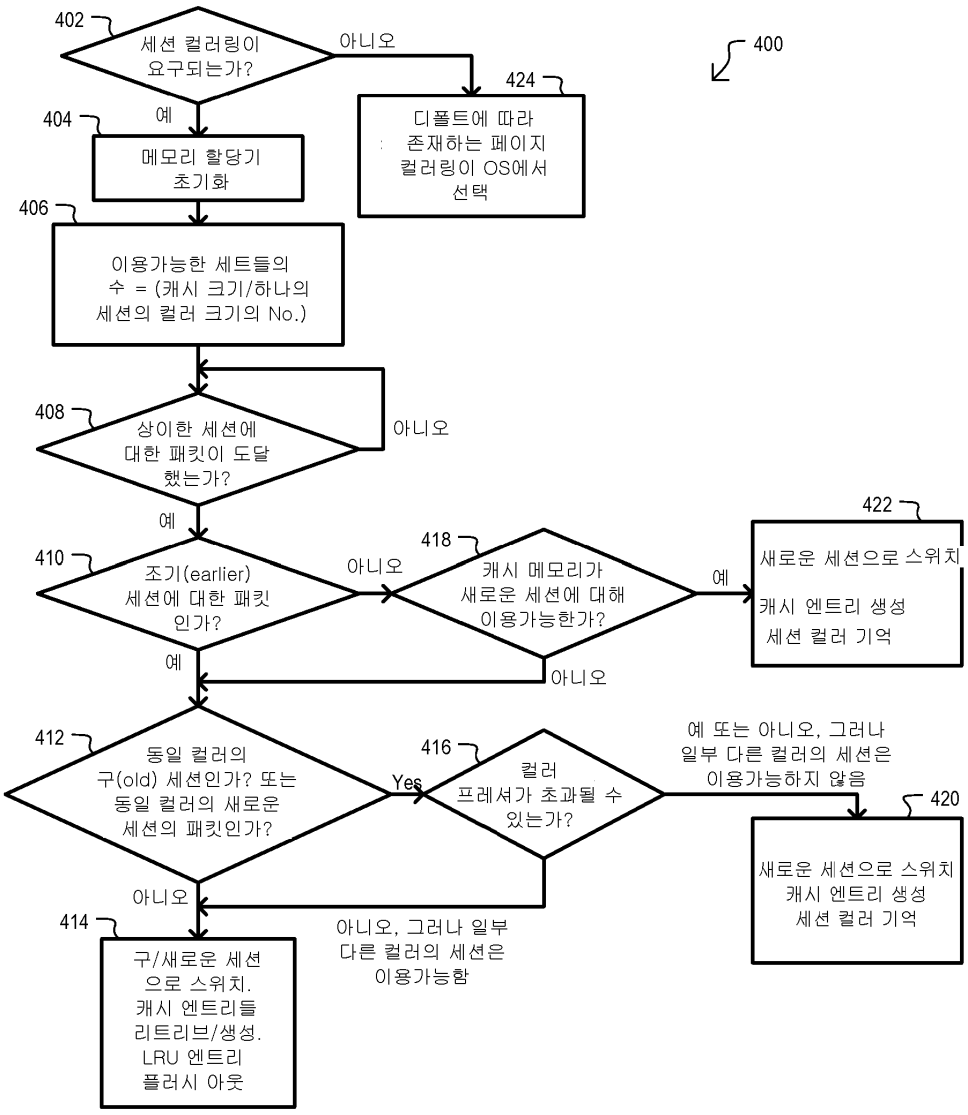

도면2f



도면3


301
302
DMA 링
302a
IOV
302b
304
TLB
304a
IOMMU
304b
306
프로비저닝
에이전트
306d
CPUIO
306a
MemCtl
306b
코어
306c
312
(PCI)
314
(HT)
300
310
DDR3
316
SD RAM
310b
308
SD
308a
스케줄러
308b
DMA
308c
DMA
슬레이브
310a
통지/
프리페치
310c
RLDRAM
308g
E$
308h
ctrl/
메타데이터
308d
분류기
308e
MMIO
Ingr
310e
아비터
310f
308i
OS 실행 아파치
308j
MMIO
Egr
310g
Mem
308m
ACP
308l
터널
308k
스케줄러
308n

도면4


400
402
세션 컬러링이 요구되는가?
아니오
424
디폴트에 따라 존재하는 페이지 컬러링이 OS에서 선택
예
404
메모리 할당기 초기화
406
이용가능한 세트들의 수 = (캐시 크기/하나의 세션의 컬러 크기의 No.)
408
상이한 세션에 대한 패킷이 도달했는가?
아니오
예
410
조기(earlier) 세션에 대한 패킷인가?
아니오
418
캐시 메모리가 새로운 세션에 대해 이용가능한가?
예
422
새로운 세션으로 스위치
캐시 엔트리 생성
세션 컬러 기억
아니오
예
412
동일 컬러의 구(old) 세션인가? 또는 동일 컬러의 새로운 세션의 패킷인가?
Yes
416
컬러 프레셔가 초과될 수 있는가?
예 또는 아니오, 그러나 일부 다른 컬러의 세션은 이용가능하지 않음
420
새로운 세션으로 스위치
캐시 엔트리 생성
세션 컬러 기억
아니오
414
구/새로운 세션으로 스위치. 캐시 엔트리들 리트리브/생성. LRU 엔트리 플러시 아웃
아니오, 그러나 일부 다른 컬러의 세션은 이용가능함