(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2022년12월30일
(11) 등록번호 10-2483506
(24) 등록일자 2022년12월28일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01) *G06F 11/34* (2006.01)
G06F 12/0815 (2016.01) *G06F 12/084* (2016.01)
G06F 12/0875 (2016.01) *G06F 12/0895* (2016.01)
(52) CPC특허분류
G06F 11/3636 (2013.01)
G06F 11/3476 (2013.01)
(21) 출원번호 10-2020-7011236
(22) 출원일자(국제) 2018년06월22일
심사청구일자 2021년05월25일
(85) 번역문제출일자 2020년04월17일
(65) 공개번호 10-2020-0056430
(43) 공개일자 2020년05월22일
(86) 국제출원번호 PCT/US2018/038875
(87) 국제공개번호 WO 2019/055094
국제공개일자 2019년03월21일
(30) 우선권주장
62/559,780 2017년09월18일 미국(US)
15/915,930 2018년03월08일 미국(US)
(56) 선행기술조사문헌
JP2000148533 A
JP2002304328 A
JP2007207223 A
KR1020110134855 A

(73) 특허권자
마이크로소프트 테크놀로지 라이센싱, 엘엘씨
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
모라 조르디
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 마이크로소프트 테크놀로지 라이센싱, 엘엘씨
(74) 대리인
김태홍, 김진희

전체 청구항 수 : 총 20 항 심사관 : 장지혜

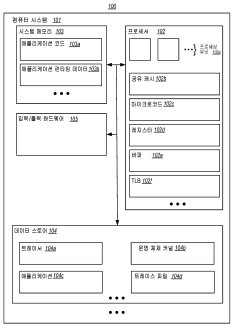
(54) 발명의 명칭 **캐시 코히어런스 프로토콜 데이터를 사용한 캐시 기반 트레이스 기록**

(57) 요 약

캐시 코히어런스 프로토콜(cache coherence protocol, CCP) 데이터를 사용하여 캐시 기반 트레이스 기록을 수행하는 방법을 개시한다. 실시예는 캐시 라인과 백킹 스토어 간의 상호 작용을 초래하는 동작이 일어났고, 그 동작을 초래한 프로세싱 유닛에 대하여 로깅이 가능하고, 캐시 라인이 로깅의 참여자이며, CCP가 트레이스에 로깅될

(뒷면에 계속)

대 표 도 - 도1



데이터가 있다고 표시하는 것을 검출한다. 그 후 실시예는 동작을 리플레이하는 데 사용될 수 있는 그 데이터가 트레이스에 로깅되게 한다.

(52) CPC특허분류
G06F 11/3632 (2013.01)
G06F 11/364 (2013.01)
G06F 12/0815 (2013.01)
G06F 12/084 (2013.01)
G06F 12/0875 (2013.01)
G06F 12/0895 (2013.01)
G06F 2212/454 (2013.01)

명 세 서

청구범위

청구항 1

컴퓨팅 디바이스에 있어서,

복수의 프로세싱 유닛;

하나 이상의 백킹 스토어(backing store)로부터의 데이터를 캐싱하는 데 사용되고 상기 복수의 프로세싱 유닛에 의해 공유되는 복수의 캐시 라인을 포함하는 캐시 메모리 - 상기 복수의 캐시 라인 및 상기 하나 이상의 백킹 스토어의 데이터 사이의 일관성은 캐시 코히어런스 프로토콜(cache coherence protocol, CCP)에 따라 관리됨 - ; 및

저장된 제어 로직

을 포함하며, 상기 저장된 제어 로직은, 적어도 다음:

적어도 다음 조건들:

(i) 동작(operation)이 상기 복수의 캐시 라인 중 특정 캐시 라인과 상기 하나 이상의 백킹 스토어 사이의 상호 작용을 야기하였다는 것;

(ii) 상기 동작을 야기한 상기 복수의 프로세싱 유닛 중 특정 프로세싱 유닛에 대하여 로깅이 인에이블되었다는 것;

(iii) 상기 특정 캐시 라인이 로깅 참여자라는 것; 및

(iv) 상기 CCP가 상기 동작에 기초하여 트레이스(trace)에 로깅될 데이터가 있다고 표시한다는 것

이 충족되었다고 결정하는 것; 및

적어도 상기 조건들이 충족되었다는 결정에 기초하여, 상기 데이터가 상기 트레이스에 로깅되게 하는 것 - 상기 데이터는 상기 동작을 리플레이하는 데 사용 가능함 -

을 수행하도록 상기 컴퓨팅 디바이스를 구성하는, 컴퓨팅 디바이스.

청구항 2

제1항에 있어서, 상기 저장된 제어 로직은 또한, 상기 특정 캐시 라인과 연관된 하나 이상의 어카운팅(accounting) 비트를 업데이트하여, 상기 특정 캐시 라인이 상기 동작 후에 로깅 참여자로 남는지 여부를 표시하도록 상기 컴퓨팅 디바이스를 구성하는, 컴퓨팅 디바이스.

청구항 3

제2항에 있어서, 상기 특정 캐시 라인과 연관된 상기 하나 이상의 어카운팅 비트는, (i) 단일 비트, (ii) 상기 복수의 프로세싱 유닛 중 하나에 각각 대응하는 복수의 비트, 및 (iii) 프로세서 인덱스 값을 저장하는 복수의 비트 중 하나를 포함하는, 컴퓨팅 디바이스.

청구항 4

제2항에 있어서, 상기 특정 캐시 라인과 연관된 상기 하나 이상의 어카운팅 비트는, 하나 이상의 백킹 스토어로부터의 데이터를 캐싱하는 데 사용되는 캐시 라인으로부터 분리된 하나 이상의 예약된 캐시 라인에 저장되는, 컴퓨팅 디바이스.

청구항 5

제1항에 있어서, 상기 데이터가 상기 트레이스에 로깅되게 하는 것은, 상기 데이터를 버퍼에 쓰는 것을 포함하며, 상기 버퍼로부터 트레이스 파일로 데이터를 플러싱하는 것은 메모리 버스 활동에 기초하여 지연되는, 컴퓨

팅 디바이스.

청구항 6

제1항에 있어서, 상기 저장된 제어 로직은 또한, 연관 캐시에서의 그룹(group) 및 웨이(way)를 참조하여 적어도 하나의 캐시 제거(eviction)를 로깅하도록 상기 컴퓨팅 디바이스를 구성하는, 컴퓨팅 디바이스.

청구항 7

제1항에 있어서, 상기 로깅되는 데이터는 상이한 CCP 상태 사이의 천이를 포함하는, 컴퓨팅 디바이스.

청구항 8

제1항에 있어서, 상기 로깅되는 데이터는, 쓰기 상태로부터 읽기 상태로의 천이, 쓰기 상태로부터 쓰기 상태로의 천이, 및 읽기 상태로부터 쓰기 상태로의 천이 중 적어도 하나를 포함하는, 컴퓨팅 디바이스.

청구항 9

제1항에 있어서, 트레이스에 로깅될 데이터가 있다고 식별하기 위해 상기 CCP를 사용하는 것은, 읽기 상태로부터 읽기 상태로의 천이가 상기 트레이스에 로깅될 필요가 없다고 식별하는 것을 포함하는, 컴퓨팅 디바이스.

청구항 10

제1항에 있어서, 각 프로세싱 유닛에 대한 데이터는 적어도 하나의 개별 데이터 스트림에 로깅되는, 컴퓨팅 디바이스.

청구항 11

제1항에 있어서, 둘 이상의 프로세싱 유닛에 대한 데이터는 동일한 데이터 스트림에 로깅되지만, 프로세싱 유닛 식별자로 태깅되는, 컴퓨팅 디바이스.

청구항 12

제1항에 있어서, 상기 트레이스에 로깅될 데이터는 서열화(ordering) 정보를 포함하는, 컴퓨팅 디바이스.

청구항 13

제1항에 있어서, 상기 로깅될 데이터는, 엔클레이브(enclave)에 의해 상기 특정 캐시 라인에 쓰여진 데이터를 포함하며, 상기 데이터가 상기 트레이스에 로깅되게 하는 것은:

상기 특정 캐시 라인과 상기 하나 이상의 백킹 스토어 사이의 상호 작용을 야기한 동작이, 상기 엔클레이브와 상호 작용하는 스레드에 대응할 때, 상기 스레드에 대응하는 트레이스 데이터 스트림에 상기 데이터가 로깅되게 하는 것; 또는

상기 특정 캐시 라인과 상기 하나 이상의 백킹 스토어 사이의 상호 작용을 야기한 동작이 상기 엔클레이브에 대응할 때, 상기 스레드에 대응하는 트레이스 데이터 스트림으로부터 분리되도록 상기 데이터가 로깅되게 하는 것을 포함하는, 컴퓨팅 디바이스.

청구항 14

복수의 프로세싱 유닛, 및 하나 이상의 백킹 스토어로부터의 데이터를 캐싱하는 데 사용되고 상기 복수의 프로세싱 유닛에 의해 공유되는 복수의 캐시 라인을 포함하는 캐시 메모리 - 상기 복수의 캐시 라인 및 상기 하나 이상의 백킹 스토어의 데이터 사이의 일관성은 캐시 코히어런스 프로토콜에 따라 관리됨 - 를 포함하는 컴퓨팅 환경에서 구현되는, 캐시 코히어런스 프로토콜(CCP) 데이터를 사용하여 캐시 기반 트레이스 기록(recording)을 수행하기 위한 방법에 있어서,

적어도 다음 조건들 -

(i) 동작이 상기 복수의 캐시 라인 중 특정 캐시 라인과 상기 하나 이상의 백킹 스토어 사이의 상호 작용을 야기하였다는 것;

(ii) 상기 동작을 야기한 상기 복수의 프로세싱 유닛 중 특정 프로세싱 유닛에 대하여 로깅이 인에이블

되었다는 것;

(iii) 상기 특정 캐시 라인이 로깅 참여자라는 것; 및

(iv) 상기 CCP가 상기 동작에 기초하여 트레이스에 로깅될 데이터가 있다고 표시한다는 것

- 이 충족되었다고 결정하는 단계; 및

적어도 상기 조건들이 충족되었다는 결정에 기초하여, 상기 데이터가 상기 트레이스에 로깅되게 하는 단계 - 상기 데이터는 상기 동작을 리플레이하는 데 사용 가능함 -

를 포함하는, 캐시 기반 트레이스 기록을 수행하기 위한 방법.

청구항 15

제14항에 있어서, 상기 특정 캐시 라인과 연관된 하나 이상의 어카운팅 비트를 업데이트하여, 상기 특정 캐시 라인이 상기 동작 후에 로깅 참여자로 남는지 여부를 표시하는 단계를 더 포함하는, 캐시 기반 트레이스 기록을 수행하기 위한 방법.

청구항 16

제14항에 있어서, 상기 데이터가 상기 트레이스에 로깅되게 하는 단계는, 상기 데이터를 버퍼에 쓰는 단계를 포함하며, 상기 버퍼로부터 트레이스 파일로 데이터를 플러싱하는 것은 메모리 버스 활동에 기초하여 지연되는, 캐시 기반 트레이스 기록을 수행하기 위한 방법.

청구항 17

제14항에 있어서, 상기 로깅되는 데이터는 상이한 CCP 상태 사이의 천이를 포함하는, 캐시 기반 트레이스 기록을 수행하기 위한 방법.

청구항 18

제14항에 있어서, 상기 로깅되는 데이터는, 쓰기 상태로부터 읽기 상태로의 천이, 쓰기 상태로부터 쓰기 상태로의 천이, 및 읽기 상태로부터 쓰기 상태로의 천이 중 적어도 하나를 포함하는, 캐시 기반 트레이스 기록을 수행하기 위한 방법.

청구항 19

제14항에 있어서, 데이터가 트레이스에 로깅될지의 여부를 식별하기 위해 상기 CCP를 사용하는 것은, 읽기 상태로부터 읽기 상태로의 천이가 상기 트레이스에 로깅될 필요가 없다고 식별하는 것을 포함하는, 캐시 기반 트레이스 기록을 수행하기 위한 방법.

청구항 20

복수의 프로세싱 유닛, 및 하나 이상의 백킹 스토어로부터의 데이터를 캐싱하는 데 사용되고 상기 복수의 프로세싱 유닛에 의해 공유되는 복수의 캐시 라인을 포함하는 캐시 메모리 - 상기 복수의 캐시 라인 및 상기 하나 이상의 백킹 스토어의 데이터 사이의 일관성은 캐시 코히어런스 프로토콜(CCP)에 따라 관리됨 - 를 포함하는 컴퓨팅 디바이스에서 사용하기 위한, 매체에 저장된 컴퓨터 프로그램에 있어서, 상기 매체에 저장된 컴퓨터 프로그램은, 상기 컴퓨팅 디바이스로 하여금 적어도 다음 동작들을 수행하게 하도록 하나 이상의 프로세싱 유닛에 의해 실행 가능한 컴퓨터 실행 가능 명령어를 포함하며, 상기 동작들은:

적어도 조건들이 충족되었다고 결정하는 것 - 상기 조건들은,

(i) 동작이 상기 복수의 캐시 라인 중 특정 캐시 라인과 상기 하나 이상의 백킹 스토어 사이의 상호 작용을 야기하였다는 것;

(ii) 상기 동작을 야기한 상기 복수의 프로세싱 유닛 중 특정 프로세싱 유닛에 대하여 로깅이 인에이블되었다는 것;

(iii) 상기 특정 캐시 라인이 로깅 참여자라는 것; 및

(iv) 상기 CCP가 상기 동작에 기초하여 트레이스에 로깅될 데이터가 있다고 표시한다는 것

임 - ; 및

적어도 상기 조건들이 충족되었다는 결정에 기초하여, 상기 데이터가 상기 트레이스에 로깅되게 하는 것 - 상기 데이터는 상기 동작을 리플레이하는 데 사용 가능함 -

을 포함하는, 매체에 저장된 컴퓨터 프로그램.

발명의 설명

기 술 분 야

배 경 기 술

[0001] 소프트웨어 애플리케이션을 개발하는 동안 코드를 작성할 때 개발자는 일반적으로 런타임 오류 및 기타 소스 코드 오류를 찾기 위하여 코드를 " 디버깅"하는 데 많은 시간을 소비한다. 그렇게 하는 데 있어서, 개발자는 상이한 입력에 기반한 프로그램의 동작 관찰, (예를 들어, 변수 값 프린팅, 실행 분기 추적 등을 위한) 디버깅 코드 삽입, 코드 부분의 일시적 제거 등과 같은 소스 코드 버그를 재현하고 로컬라이즈하는 여러 가지 접근법을 사용할 수 있다. 런타임 오류를 추적하여 코드 버그를 정확히 찾아내는(pinpoint) 것은 애플리케이션 개발 시간의 상당 부분을 차지할 수 있다.

[0002] 개발자의 코드 디버깅 프로세스를 지원하기 위하여 많은 유형의 디버깅 애플리케이션("디버거(debugger)")이 개발되었다. 이들 툴은 개발자에게 컴퓨터 코드 실행을 트레이스, 시각화 및 변경할 수 있는 능력을 제공한다. 예를 들어, 디버거는, 다른 것들 중에서도, 코드 명령어의 실행을 시각화할 수 있고, 코드 실행 중 다양한 시간에 코드 변수 값을 제시할 수 있고, 개발자가 코드 실행 경로를 변경할 수 있게 하고, 그리고/또는 개발자가 관심있는 코드 요소에 (실행 중 도달할 때, 코드의 실행이 중지(suspend)되게 할 수 있는) "중단점(breakpoint)" 및/또는 "감시점(watchpoint)"을 설정할 수 있도록 할 수 있다.

[0003] 새로운 형태의 디버깅 애플리케이션은 "시간 여행(time travel)", "리버스(reverse)" 또는 "과거의(historic)" 디버깅을 가능하게 한다. "시간 여행" 디버깅을 사용하면, 프로그램(예를 들어, 스레드와 같은 실행 가능 엔티티)의 실행이 트레이스 애플리케이션에 의해 하나 이상의 트레이스 파일로 기록(record)/트레이스(trace)된다. 이후 이들 트레이스 파일을 사용하여, 전방향(forward) 및 역방향(backward) 분석 둘 다를 위하여 나중에 프로그램의 실행을 리플레이(replay)할 수 있다. 예를 들어, "시간 여행" 디버거는 개발자가 리버스 중단점/감시점뿐만 아니라, (종래의 디버거와 마찬가지로) 전방향 중단점/감시점을 설정하게 할 수 있다.

발명의 내용

[0004] 본 명세서의 실시예는 트레이스 파일에 어떤 데이터가 로깅되어야 하는지를 결정하기 위하여 프로세서의 공유 캐시를 그 캐시 코히어런스 프로토콜(cache coherence protocol, CCP)과 함께 이용함으로써 "시간 여행" 디버깅 기록을 향상시킨다. 이렇게 하면 이전의 접근법과 비교할 때 트레이스 파일 크기를 몇 배 정도 줄일 수 있으므로 트레이스 기록의 오버헤드가 크게 줄어든다.

[0005] 일부 실시예에서, (i) 복수의 프로세싱 유닛, 및 (ii) 하나 이상의 백킹 스토어(backing store)로부터의 데이터를 캐싱하는 데 사용되고, 복수의 프로세싱 유닛에 의해 공유되는 복수의 캐시 라인을 포함하는 캐시 메모리를 포함하는 컴퓨팅 환경에서 구현된다. 복수의 캐시 라인과 하나 이상의 백킹 스토어의 데이터 사이의 일관성(consistency)은 캐시 코히어런스 프로토콜에 따라 관리된다.

[0006] 이들 실시예는 CCP 데이터를 사용하여 캐시 기반 트레이스 기록을 수행하는 단계를 포함한다. 이들 실시예는, 어떤 동작이 복수의 캐시 라인 중 특정 캐시 라인과 하나 이상의 백킹 스토어 간의 상호 작용을 초래하였고, 복수의 프로세싱 유닛 중 그 동작을 초래한 특정 프로세싱 유닛에 대하여 로깅이 인에이블되었고, 그 특정 캐시 라인이 로깅의 참여자이며, CCP가 트레이스에 로깅될 데이터가 있다고 표시한다고 결정하는 단계를 포함한다. 적어도 이들 결정에 기초하여, 실시예는 이 데이터가 트레이스에 로깅되게 한다. 데이터는 동작을 리플레이하는 데 사용 가능하다.

[0007] 본 발명의 내용은 이하의 발명을 실시하기 위한 구체적인 내용에서 추가로 설명되는 개념의 선택을 단순화된 형태로 소개하기 위하여 제공된다. 본 발명의 내용은 특허청구된 주제(claimed subject matter)의 주요 특징 또는

필수 특징을 식별하도록 의도되지 않으며 특허청구된 주제의 범위를 결정하는 데 도움을 주기 위하여 사용되려는 의도되지도 않는다.

도면의 간단한 설명

[0008] 본 발명의 전술한 장점과 특징 및 다른 장점과 특징이 획득될 수 있는 방식을 설명하기 위하여, 위에서 간략히 설명된 본 발명의 보다 구체적인 설명이, 첨부된 도면에 도시된 특정 실시예를 참조하여 제공될 것이다. 이들 도면은 본 발명의 전형적인 실시예만을 도시하고 따라서 그 범위를 제한하는 것으로 간주되지 않아야 한다는 것을 이해하면서, 본 발명은 첨부 도면을 사용하여 추가적 특이성(specificity) 및 세부 사항과 함께 기술되고 설명될 것이다:

도 1은 캐시 코히어런스 프로토콜(CCP) 데이터를 사용하여 공유 캐시를 통해 코드 실행의 "비트까지-정확한(bit-accurate)" 트레이스의 기록을 용이하게 하는 예시적인 컴퓨팅 환경을 도시한다;

도 2는 공유 캐시의 예를 도시한다;

도 3은 CCP 데이터를 사용하여 캐시 기반 트레이스 기록을 수행하기 위한 예시적인 방법의 흐름도를 도시한다;

도 4a는 하나 이상의 추가 어카운팅(accounting) 비트로 캐시 라인 각각을 확장하는 예시적인 공유 캐시를 도시한다;

도 4b는 종래의 캐시 라인에 적용되는 어카운팅 비트를 저장하기 위한 하나 이상의 예약된 캐시 라인을 포함하는 공유 캐시의 예를 도시한다;

도 5는 연관 캐시 매핑(associative cache mapping)의 예를 도시한다;

도 6a는 공유 캐시에서 단일 라인 상의 4 개의 프로세싱 유닛에 의한 예시적인 읽기(read) 및 쓰기(write) 활동을 나타내는 표를 도시한다;

도 6b는 도 6a에 도시된 읽기 및 쓰기 활동에 기초한 예시적인 추적된(tracked) 캐시 코히어런스 상태를 나타내는 표를 도시한다;

도 6c는 도 6a에 도시된 읽기 및 쓰기 활동에 기초한 공유 캐시의 어카운팅 비트(즉, 유닛 비트, 인덱스 비트 및/또는 플래그 비트)에 저장된 예시적인 데이터를 나타내는 표를 도시한다;

도 6d는 도 6a에 도시된 읽기 및 쓰기 활동과 관련하여 트레이스 파일에 쓰여질 수 있는 예시적인 로그 데이터를 나타내는 표를 도시한다;

도 7a는 프로세서가 어떻게 추적되는지에 따라 일부 읽기->읽기 천이가 트레이스로부터 생략될 수 있는 예를 도시한다;

도 7b는 도 7a에서 강조된 읽기->읽기 천이를 생략하는 로깅 데이터의 예를 도시한다;

도 7c는 "인덱스 비트"가 사용되고 인덱스가 읽기 시에 업데이트된다면 기록될 수 있는 예시적인 로깅 데이터를 나타내는 표를 도시한다;

도 8a는 각각 4 개의 프로세싱 유닛을 포함하는 2 개의 프로세서 및 L1-L3 캐시를 포함하는 예시적인 컴퓨팅 환경을 도시한다;

도 8b는 도 8a의 프로세싱 유닛 중 일부에 의해 수행되는 예시적인 읽기 및 쓰기 동작(operation)을 나타내는 표를 도시한다;

도 9a는 2 개의 프로세싱 유닛에 의한 예시적인 읽기 및 쓰기를 보여주는 표를 도시한다;

도 9b는 로그 항목이 만들어질 수 있을 때 CCP 유닛 정보 및 캐시 라인 플래그 비트를 제공하는 환경 대(versus) CCP 인덱스 정보 및 캐시 라인 플래그 비트를 제공하는 환경을 비교하는 표를 예시하는 예를 도시한다;

도 10a는 메모리 어드레스의 상이한 부분의 예, 및 연관 캐시와의 관계를 도시한다;

도 10b는 연관 캐시에서 캐시 미스(miss) 및 캐시 제거(eviction)를 로깅하는 예를 도시한다.

발명을 실시하기 위한 구체적인 내용

[0009] 본 명세서의 실시예는 어떤 데이터가 트레이스 파일에 로깅되어야 하는지를 결정하기 위하여 프로세서의 공유 캐시를 그 캐시 코히어런스 프로토콜과 함께 사용함으로써 "시간 여행" 디버깅 기록을 향상시킨다. 이렇게 하면 이전 접근법과 비교할 때 트레이스 파일 크기를 몇 배 정도 줄일 수 있으므로 트레이스 기록의 오버헤드가 상당히 줄어든다.

[0010] 도 1은 캐시 코히어런스 프로토콜 데이터를 사용하여 공유 캐시를 통해 코드 실행의 "비트까지-정확한(bit-accurate)" 트레이스의 기록을 용이하게 하는 예시적인 컴퓨팅 환경(100)을 도시한다. 도시된 바와 같이, 실시예는 예를 들어 하나 이상의 프로세서(102), 시스템 메모리(103), 하나 이상의 데이터 스토어(data store)(104) 및/또는 입력/출력 하드웨어(105)와 같은 컴퓨터 하드웨어를 포함하는 특수 목적 또는 범용 컴퓨터 시스템(101)을 포함하거나 이용할 수 있다.

[0011] 본 발명의 범위 내의 실시예는 컴퓨터 실행 가능 명령어 및/또는 데이터 구조를 운반 또는 저장하기 위한 물리적 매체 및 다른 컴퓨터 판독 가능 매체를 포함한다. 이러한 컴퓨터 판독 가능 매체는, 컴퓨터 시스템(101)에 의해 액세스될 수 있는 임의의 이용 가능한 매체일 수 있다. 컴퓨터 실행 가능 명령어 및/또는 데이터 구조를 저장하는 컴퓨터 판독 가능 매체는 컴퓨터 저장 디바이스이다. 컴퓨터 실행 가능 명령어 및/또는 데이터 구조를 운반하는 컴퓨터 판독 가능 매체는 전송 매체이다. 따라서, 비제한적인 예로서, 본 발명의 실시예는 적어도 두 가지의 명백히 상이한 종류의 컴퓨터 판독 가능 매체: 컴퓨터 저장 디바이스 및 전송 매체를 포함할 수 있다.

[0012] 컴퓨터 저장 디바이스는 컴퓨터 실행 가능 명령어 및/또는 데이터 구조를 저장하는 물리적 하드웨어 디바이스이다. 컴퓨터 저장 디바이스는 RAM, ROM, EEPROM, 솔리드 스테이트 드라이브("SSD"), 플래시 메모리, 위상 변화 메모리("PCM"), 광 디스크 스토리지, 자기 디스크 스토리지 또는 기타 자기 저장 디바이스, 또는 컴퓨터 실행 가능 명령어 또는 데이터 구조의 형태로 프로그램 코드를 저장하는 데 사용될 수 있고, 본 발명의 개시된 기능을 구현하기 위하여 컴퓨터 시스템(101)에 의해 액세스 및 실행될 수 있는 임의의 다른 하드웨어 디바이스와 같은 다양한 컴퓨터 하드웨어를 포함한다. 따라서, 예를 들어, 컴퓨터 저장 디바이스는 이하 논의되는 바와 같이, 도시된 시스템 메모리(103), 컴퓨터 실행 가능 명령어 및/또는 데이터 구조를 저장할 수 있는 도시된 데이터 스토어(104), 또는 온-프로세서(on-processor) 스토리지와 같은 다른 스토리지를 포함할 수 있다.

[0013] 전송 매체는 컴퓨터 실행 가능 명령어 또는 데이터 구조의 형태로 프로그램 코드를 운반하는 데 사용될 수 있고 컴퓨터 시스템(101)에 의해 액세스될 수 있는 네트워크 및/또는 데이터 링크를 포함할 수 있다. "네트워크"는 컴퓨터 시스템 및/또는 모듈 및/또는 다른 전자 디바이스 사이에서 전자 데이터의 전송을 가능하게 하는 하나 이상의 데이터 링크로서 정의된다. 정보가 네트워크 또는 다른 통신 연결(connection)(하드와이어드, 무선, 또는 하드와이어드와 무선의 조합)을 통해 컴퓨터 시스템으로 전송되거나 제공될 때, 컴퓨터 시스템은 연결을 전송 매체로 볼 수 있다. 상기 조합은 또한 컴퓨터 판독 가능 매체의 범위 내에 포함되어야 한다. 예를 들어, 입력/출력 하드웨어(105)는 컴퓨터 실행 가능한 명령어 또는 데이터 구조의 형태로 프로그램 코드를 운반하는 데 사용될 수 있는 네트워크 및/또는 데이터 링크를 연결하는 하드웨어(예를 들어, 네트워크 인터페이스 모듈("NIC(network interface module)"))를 포함할 수 있다.

[0014] 또한, 다양한 컴퓨터 시스템 컴포넌트에 도달하면, 컴퓨터 실행 가능 명령어 또는 데이터 구조 형태의 프로그램 코드가 전송 매체로부터 컴퓨터 저장 디바이스로(또는 그 반대로) 자동적으로 전송될 수 있다. 예를 들어, 네트워크 또는 데이터 링크를 통해 수신된 컴퓨터 실행 가능 명령어 또는 데이터 구조는 NIC(예를 들어, 입력/출력 하드웨어(105)) 내의 RAM에 버퍼링된 후, 결국 컴퓨터 시스템(101)에서 시스템 메모리(103) 및/또는 덜 휘발성인 컴퓨터 저장 디바이스(예를 들어, 데이터 스토어(104))로 전송될 수 있다. 따라서, 컴퓨터 저장 디바이스는, 전송 매체를 또한(또는 심지어는 주로) 이용하는 컴퓨터 시스템 컴포넌트에 포함될 수 있음을 이해하여야 한다.

[0015] 컴퓨터 실행 가능 명령어는 예를 들어 프로세서(102)에서 실행될 때 컴퓨터 시스템(101)이 특정 기능 또는 기능의 그룹을 수행하게 하는 명령어 및 데이터를 포함한다. 컴퓨터 실행 가능 명령어는 예를 들어, 어셈블리 언어와 같은 이진, 중간 포맷 명령어, 또는 심지어 소스 코드일 수 있다.

[0016] 당업자는 본 발명이 개인용 컴퓨터, 데스크탑 컴퓨터, 랩탑 컴퓨터, 메시지 프로세서, 핸드 헬드 디바이스, 멀티 프로세서 시스템, 마이크로 프로세서 기반 또는 프로그래머블 가전 제품, 네트워크 PC, 미니 컴퓨터, 메인프레임 컴퓨터, 휴대 전화, PDA, 태블릿, 호출기, 라우터, 스위치 등을 포함하는 많은 유형의 컴퓨터 시스템 구성을 갖는 네트워크 컴퓨팅 환경에서 실시될 수 있음을 이해할 것이다. 본 발명은 또한 네트워크를 통해 (하드와이어드 데이터 링크, 무선 데이터 링크, 또는 하드와이어드 및 무선 데이터 링크의 조합에 의해) 링크된 로컬 및 원격 컴퓨터 시스템 둘 다가 작업을 수행하는 분산 시스템 환경에서 실시될 수 있다. 이와 같이, 분산 시스템 환경에서, 컴퓨터 시스템은 복수의 구성 컴퓨터 시스템을 포함할 수 있다. 분산 시스템 환경에서 프로그램

모듈은 로컬 및 원격 메모리 저장 디바이스 둘 다에 있을 수 있다.

[0017] 당업자는 또한 본 발명이 클라우드 컴퓨팅 환경에서 실시될 수 있음을 이해할 것이다. 필수는 아니지만 클라우드 컴퓨팅 환경은 분산될 수 있다. 분산될 때 클라우드 컴퓨팅 환경은 조직 내에서 국제적으로 분산되고/되거나 여러 조직에 걸쳐 소유되는 컴포넌트를 가질 수 있다. 이 설명 및 다음의 청구범위에서, "클라우드 컴퓨팅"은 구성 가능한 컴퓨팅 리소스(예를 들어, 네트워크, 서버, 스토리지, 애플리케이션, 및 서비스)의 공유 풀에 대한 온-디맨드 네트워크 액세스를 가능하게 하기 위한 모델로서 정의된다. 적절하게 배치(deploy)될 때 "클라우드 컴퓨팅"의 정의는 이러한 모델로부터 얻어질 수 있는 다른 많은 이점 중 어느 것으로도 제한되지 않는다.

[0018] 클라우드 컴퓨팅 모델은 온-디맨드 셀프 서비스, 광범위한 네트워크 액세스, 리소스 풀링, 빠른 탄력성, 측정되는 서비스(measured service) 등과 같은 다양한 특성으로 구성될 수 있다. 클라우드 컴퓨팅 모델은 또한 예를 들어, "SaaS"(Software as a Service), "PaaS"(Platform as a Service) 및 "IaaS"(Infrastructure as a Service)와 같은 다양한 서비스 모델의 형태로 제공될 수 있다. 클라우드 컴퓨팅 모델은 또한 프라이빗 클라우드, 커뮤니티 클라우드, 퍼블릭 클라우드, 하이브리드 클라우드 등과 같은 상이한 배치 모델을 사용하여 배치될 수도 있다.

[0019] 클라우드 컴퓨팅 환경과 같은 일부 실시예는 각각 하나 이상의 가상 머신을 실행할 수 있는 하나 이상의 호스트를 포함하는 시스템을 포함할 수 있다. 동작 중에 가상 머신은 운영 체제 및 아마 하나 이상의 다른 애플리케이션도 지원하는 운영 컴퓨팅 시스템을 에뮬레이션한다. 일부 실시예에서, 각각의 호스트는 가상 머신의 관점에서 추상화된 물리적 리소스를 사용하여 가상 머신을 위한 가상 리소스를 에뮬레이션하는 하이퍼바이저를 포함한다. 하이퍼바이저는 또한 가상 머신 간에 적절한 격리를 제공한다. 따라서, 임의의 주어진 가상 머신의 관점에서, 하이퍼바이저는 가상 머신이 물리적 리소스의 외관(예를 들어, 가상 리소스)과만 인터페이스하더라도 가상 머신이 물리적 리소스와 인터페이스하고 있다는 착시를 제공한다. 물리적 리소스의 예는 프로세싱 용량, 메모리, 디스크 공간, 네트워크 대역폭, 미디어 드라이브 등을 포함한다.

[0020] 도시된 바와 같이, 데이터 스토어(104)는 예를 들어, 트레이서(104a), 운영 체제 커널(104b) 및 애플리케이션(104c)(예를 들어, 트레이서(104a) 및 하나 이상의 트레이스 파일(104d)에 의한 트레이싱의 주체인 애플리케이션)과 같은 애플리케이션 프로그램을 나타내는 컴퓨터 실행 가능 명령어 및/또는 데이터 구조를 저장할 수 있다. 이들 프로그램이 (예를 들어, 프로세서(102)를 사용하여) 실행되고 있을 때, 시스템 메모리(103)는 런타임 데이터 구조, 컴퓨터 실행 가능 명령어 등과 같은 대응하는 런타임 데이터를 저장할 수 있다. 따라서, 도 1은 런타임 애플리케이션 코드(103a) 및 애플리케이션 런타임 데이터(103b)(예를 들어, 각각은 애플리케이션(104c)에 대응함)를 포함하는 것으로서 시스템 메모리(103)를 도시한다.

[0021] 트레이서(104a)는 애플리케이션(104c)과 같은 애플리케이션의 실행의 비트까지-정확한 트레이스를 기록하고, 트레이스 파일(104d)에 트레이스 데이터를 저장하는 데 사용될 수 있다. 일부 실시예에서, 트레이서(104a)는 독립형 애플리케이션인 반면, 다른 실시예에서 트레이서(104a)는 운영 체제 커널(104b), 하이퍼바이저, 클라우드 패브릭 등과 같은 다른 소프트웨어 컴포넌트 내에 통합된다. 트레이스 파일(104d)은 데이터 스토어(104)에 저장된 것으로 도시되어 있지만, 트레이스 파일(104d)은 또한 시스템 메모리(103) 또는 어떠한 다른 저장 디바이스에 배타적으로 또는 일시적으로 기록될 수 있다.

[0022] 도 1은 프로세서(102)의 내부 하드웨어 컴포넌트의 단순화된 표현을 포함한다. 도시된 바와 같이, 각 프로세서(102)는 복수의 프로세싱 유닛(102a)을 포함한다. 각 프로세싱 유닛은 물리적(즉, 물리적 프로세서 코어) 및/또는 논리적(즉, 둘 이상의 애플리케이션 스레드가 물리적 코어에서 실행되는 하이퍼스레딩을 지원하는 물리적 코어에 의해 제공되는 논리적 코어)일 수 있다. 따라서, 예를 들어, 프로세서(102)가 일부 실시예에서 단일 물리적 프로세싱 유닛(코어)만을 포함할 수 있지만, 그 단일 물리적 프로세싱 유닛에 의해 제공된 둘 이상의 논리적 프로세싱 유닛(102a)을 포함할 수 있다.

[0023] 각 프로세싱 유닛(102a)은 애플리케이션(예를 들어, 트레이서(104a), 운영 커널(104b), 애플리케이션(104c) 등)에 의해 정의되는 프로세서 명령어를 실행하고, 명령어는 사전 정의된 프로세서 명령어 세트 아키텍처(instruction set architecture, ISA) 중에서 선택된다. 각 프로세서(102)의 특정 ISA는 프로세서 제조사 및 프로세서 모델에 기초하여 달라진다. 일반적인 ISA는 INTEL, INC.의 IA-64 및 IA-32 아키텍처, ADVANCED MICRO DEVICES, INC.의 AMD64 아키텍처 및 ARM HOLDINGS, PLC의 다양한 Advanced RISC Machine("ARM") 아키텍처를 포함하지만, 많은 수의 다른 ISA가 존재하고, 본 발명에 의해 사용될 수 있다. 일반적으로, "명령어(instruction)"는, 프로세서에 의해 실행 가능한, 외부에서 볼 수 있는(즉, 프로세서 외부의) 가장 작은 코드 단위이다.

[0024] 각 프로세싱 유닛(102a)은 공유 캐시(102b)로부터 프로세서 명령어를 획득하고, 공유 캐시(102b)의 데이터에 기초하여, 레지스터(102d)의 데이터에 기초하여, 그리고/또는 입력 데이터 없이 프로세서 명령어를 실행한다. 일반적으로, 공유 캐시(102b)는 시스템 메모리(103) 및/또는 다른 캐시와 같은 백킹 스토어의 일부의 온-프로세서 사본을 저장하는 소량(즉, 전형적인 양의 시스템 메모리(103)에 비해 작음)의 랜덤 액세스 메모리이다. 예를 들어, 애플리케이션 코드(103a)를 실행할 때, 공유 캐시(102b)는 애플리케이션 런타임 데이터(103b)의 일부를 포함한다. 프로세싱 유닛(102a)이 공유 캐시(102b)에 아직 저장되지 않은 데이터를 요구하면, "캐시 미스(cash miss)"가 발생하고, 그 데이터는 시스템 메모리(103)로부터 페치(fetch)된다(공유 캐시(102b)로부터 일부 다른 데이터를 잠재적으로 "제거(evict)"한다).

[0025] 전형적으로, 공유 캐시(102b)는 복수의 " 캐시 라인(cache line)"을 포함하고, 이들 각각은 백킹 스토어로부터의 메모리 청크를 저장한다. 예를 들어, 도 2는 복수의 캐시 라인(203)을 포함하는 공유 캐시(200)의 적어도 일부의 예를 도시하며, 이들 각각은 어드레스 부분(address portion)(201) 및 값 부분(value portion)(202)을 포함한다. 각 캐시 라인(203)의 어드레스 부분(201)은, 해당 라인에 대응하는 백킹 스토어(예를 들어, 시스템 메모리(103))에 어드레스를 저장할 수 있고, 값 부분(202)은 초기에 백킹 스토어로부터 수신된 값을 저장할 수 있다. 값 부분(202)은 프로세싱 유닛(102a)에 의해 수정될 수 있고, 결국 백킹 스토어로 다시 제거될 수 있다. 타원에 의해 표시된 바와 같이, 공유 캐시(200)는 다수의 캐시 라인을 포함할 수 있다. 예를 들어, 최신 INTEL 프로세서는 512 개 이상의 캐시 라인을 포함하는 레이어-1(layer-1) 캐시를 포함할 수 있다. 이 캐시에서, 각 캐시 라인은 일반적으로 8 바이트(64 비트) 메모리 어드레스를 참조하여 64 바이트(512 비트) 값을 저장하는 데 사용될 수 있다.

[0026] 각 캐시 라인(203)의 어드레스 부분(201)에 저장된 어드레스는 시스템 메모리(103)의 실제 메모리 어드레스와 같은 물리적 어드레스일 수 있다. 대안적으로, 각 캐시 라인(203)의 어드레스 부분(201)에 저장된 어드레스는 가상 어드레스일 수 있고, 이는 추상화(abstraction)를 제공하기 위하여 물리적 어드레스에 할당된 어드레스이다. 그러한 추상화는 예를 들어 프로세서(102)에서 실행되는 상이한 프로세스 사이의 메모리 분리를 용이하게 하기 위하여 사용될 수 있다. 가상 어드레스가 사용될 때, 프로세서(102)는 변환 색인 버퍼(translation lookaside buffer, TLB)(102f)(보통, 메모리 관리 유닛(memory management unit(MMU)의 일부임)를 포함할 수 있고, 이는 물리적 메모리 어드레스와 가상 메모리 어드레스 간의 매핑을 유지한다.

[0027] 공유 캐시(102b)는 코드 캐시 부분 및 데이터 캐시 부분을 포함할 수 있다. 예를 들어, 애플리케이션 코드(103a)를 실행할 때, 공유 캐시(102b)의 코드 부분은 애플리케이션 코드(103a)에 저장된 프로세서 명령어의 적어도 일부를 저장하고 공유 캐시(102b)의 데이터 부분은 애플리케이션 런타임 데이터(103b)의 데이터 구조의 적어도 일부를 저장한다. 종종 프로세서 캐시는 별도의 티어/레이어(예를 들어, 레이어 1(L1), 레이어 2(L2) 및 레이어 3(L3))로 분할되며, 일부 티어(예를 들어, L3)는 잠재적으로 프로세서(102)와 분리되어 존재할 수 있다. 따라서, 공유 캐시(102b)는 이들 레이어 중 하나(L1)를 포함하거나, 복수의 이들 레이어를 포함할 수 있다.

[0028] 다수의 캐시 레이어가 사용될 때, 프로세싱 유닛(102a)은 최하위 레이어(L1)와 직접 상호 작용한다. 대부분의 경우에, 레이어 사이에서 데이터가 흐른다(예를 들어, 읽기를 할 때 L3 캐시는 시스템 메모리(103)와 상호 작용하고, L2 캐시에 데이터를 서비스하고, L2 캐시는 차례로 L1 캐시에 데이터를 서비스한다). 프로세싱 유닛(102a)이 쓰기를 수행할 필요가 있을 때, 캐시는, 프로세싱 유닛(102a) 간에 공유되었던, 영향을 받는 데이터를 가졌던 캐시가 더 이상 해당 데이터를 갖지 않도록 보장하기 위해 조화된다(coordinate). 이 조화는 캐시 코히어런스 프로토콜(나중에 논의됨)을 사용하여 수행된다.

[0029] 캐시는 포괄적(inclusive)이거나 배타적(exclusive)이거나, 포괄적 및 배타적 거동 둘 다를 포함할 수 있다. 예를 들어, 포괄적 캐시에서 L3 레이어는 그 아래에 있는 L2 레이어의 데이터의 상위 집합(superset)을 저장하고, L2 레이어는 그 아래에 있는 L1 레이어의 상위 집합 저장할 것이다. 배타적 캐시에서, 레이어는 분리(disjoint)될 수 있다 - 예를 들어, L1 캐시가 필요로 하는 데이터가 L3 캐시에 존재하면, 레이어는 데이터, 어드레스 등과 같은 정보를 스왑(swap)할 수 있다.

[0030] 각 프로세싱 유닛(102)은 또한 마이크로코드(102)를 포함하고, 마이크로코드(102)는 프로세서(102)의 동작을 제어하는 제어 로직(즉, 실행 가능한 명령어)을 포함하고, 일반적으로 프로세서(102)에 의해 실행 애플리케이션에 노출되는 프로세서 ISA와 프로세서의 하드웨어 사이의 인터프리터(interpreter)로서 기능한다. 마이크로코드(102)는 ROM, EEPROM 등과 같은 온-프로세서 스토리지에서 구현될 수 있다.

[0031] 레지스터(102d)는 프로세서(102)의 ISA에 기초하여 정의되고 프로세서 명령어에 의해 읽기 및/또는 쓰기가 되는 하드웨어 기반 저장 장소(location)이다. 예를 들어, 레지스터(102d)는, 명령어에 의해 사용을 위하여 공유 캐

시(102b)로부터 페치된 값을 저장하고, 명령어의 실행의 결과를 저장하고/하거나, 명령어를 실행하는 것의 부작용 중 일부(예를 들어, 값의 부호의 변경, 값이 제로에 도달, 캐리(carry)의 발생 등), 프로세서 사이클 카운트 등과 같은 상태(status) 또는 상태(state)를 저장하기 위하여 일반적으로 사용된다. 따라서, 일부 레지스터(102d)는, 프로세서 명령어를 실행함으로써 야기되는 일부 상태(state) 변경을 시그널링하는 데 사용되는 "플래그"를 포함할 수 있다. 일부 실시예에서, 프로세서(102)는 또한 프로세서 동작의 상이한 양태를 제어하는 데 사용되는 제어 레지스터를 포함할 수 있다.

[0032] 일부 실시예에서, 프로세서(102)는 하나 이상의 버퍼(102e)를 포함할 수 있다. 이후에 논의되는 바와 같이, 버퍼(102e)는 트레이스 데이터를 위한 임시 저장 장소로서 사용될 수 있다. 따라서, 예를 들어, 프로세서(102)는 트레이스 데이터의 일부를 버퍼(102e)에 저장하고, 가용 메모리 버스 대역폭이 있을 때와 같이 적절한 시간에 그 데이터를 트레이스 파일(104d)로 플러시할 수 있다. 일부 구현에서, 버퍼(102e)는 공유 캐시(102b)의 일부일 수 있다.

[0033] 위에서 언급한 바와 같이, 공유 캐시(102b)를 소유한 프로세서는 캐시 코히어런스 프로토콜("CCP")에 따라 캐시를 동작시킨다. 특히, CCP는 다양한 프로세싱 유닛(102a)이 공유 캐시(102b)로부터 데이터를 읽고 공유 캐시(102b)에 데이터를 쓸 때 공유 캐시(102b)와 백킹 데이터 스토어(예를 들어, 시스템 메모리(103) 또는 다른 캐시)에서 데이터 사이의 일관성이 어떻게 유지되는지, 그리고 다양한 프로세싱 유닛(102a)이 항상 공유 캐시(102b)의 주어진 장소로부터 유효한 데이터를 읽는 것을 어떻게 보장하는지 정의한다. CCP는 일반적으로 프로세서(102)의 ISA에 의해 정의된 메모리 모델과 관련되고 이를 가능하게 한다.

[0034] 일반적인 CCP의 예는 MSI 프로토콜(즉, 수정(Modified), 공유(Shared) 및 무효(Invalid)), MESI 프로토콜(즉, 수정(Modified), 배타(Exclusive), 공유(Shared) 및 무효(Invalid)) 및 MOESI 프로토콜(즉, 수정(Modified), 소유(Owned), 배타(Exclusive), 공유(Shared) 및 무효(Invalid))을 포함한다. 이들 프로토콜 각각은 공유 캐시(102b)에서의 개별 장소(예를 들어, 라인)에 대한 상태를 정의한다. "수정(modified)" 캐시 장소는 공유 캐시(102b)에서 수정된 데이터를 포함하며, 따라서 백킹 스토어(예를 들어, 시스템 메모리(103) 또는 다른 캐시)의 대응하는 데이터와 잠재적으로 일치하지 않는다. "수정" 상태(state)를 갖는 장소가 공유 캐시(102b)로부터 제거될 때, 일반적인 CCP는, 캐시에게, 해당 캐시의 데이터가 백킹 스토어로 다시 쓰여지거나 다른 캐시가 이 책임을 인계 받는 것을 보장하도록 요구한다. "공유(shared)" 캐시 장소는, 백킹 스토어의 데이터로부터 수정되지 않았고, 읽기 전용 상태로 존재하며, 프로세싱 유닛(102a)에 의해 공유되는 데이터를 포함한다. 공유 캐시(102b)는 이 데이터를 백킹 스토어에 쓰지 않고 이 데이터를 제거할 수 있다. "무효(invalid)" 캐시 장소는 유효한 데이터를 포함하지 않고, 비어 있는 것으로 간주되어서 캐시 미스로부터 데이터를 저장하는 데 사용할 수 있다. "배타(exclusive)" 캐시 장소는 백킹 스토어와 일치하는 데이터를 포함하며 단일 프로세싱 유닛(102a)에 의해서만 사용된다. 배타 캐시 장소는 언제든지(즉, 읽기 요청에 응답하여) "공유(shared)" 상태로 변경되거나, 배타 캐시 장소에 쓸 때 "수정(modified)" 상태로 변경될 수 있다. "소유(owned)" 캐시 장소는 둘 이상의 프로세싱 유닛(102a)에 의해 공유되지만, 프로세싱 유닛 중 하나는 그에 대한 배타적인 변경 권한을 갖는다. 해당 프로세싱 유닛은, 변경을 할 때, 다른 프로세싱 유닛에게 직접 또는 간접적으로 통지하며, 왜냐하면 통지받은 프로세싱 유닛이 CCP 구현에 기초하여 무효화하거나 업데이트하여야 할 수 있기 때문이다.

[0035] 상이한 CCP이 캐시 코히어런스를 추적하고 그 캐시 코히어런스 데이터를 트레이서(104a)에 이용 가능하게 하는 그래뉼래리티(granularity)는 변할 수 있다. 예를 들어, 스펙트럼의 한쪽 끝에서 일부 CCP는 프로세싱 유닛 별로, 또한 캐시 라인 별로 캐시 코히어런스를 추적한다. 그러므로, 이들 CCP는 각 캐시 라인이 각 프로세싱 유닛과 관련됨에 따라 해당 캐시 라인의 상태를 추적할 수 있다. 도 6a 내지 도 6d와 관련하여 다음의 예에서 설명되는 바와 같이, 이것은 단일 캐시 라인이 각 프로세싱 유닛(102a)과 관련됨에 따라 해당 캐시 라인은 해당 캐시 라인의 상태에 대한 정보를 가질 수 있음을 의미한다. 다른 CCP는 덜 세분화되어 있으며 캐시 라인의 레벨로만 캐시 코히어런스를 추적한다(프로세싱 유닛 별 정보가 없음). 스펙트럼의 다른 쪽 끝에서 프로세서 제조사는 효율성을 위하여서만 캐시 라인의 레벨로만 캐시 코히어런스를 추적하도록 선택할 수 있는데, 이는 오직 하나의 프로세서만이 한 번에 한 라인을 배타적으로 소유(배타, 수정 등)할 수 있기 때문이다. 중간 그래뉼래리티의 예로서, CCP는 현재의 캐시 라인 상태를 갖는 프로세싱 유닛에 대한 인덱스(예를 들어, 4-프로세싱 유닛 프로세서에 대한 0, 1, 2, 3)뿐만 아니라 캐시 라인 별로 캐시 코히어런스를 추적할 수 있다.

[0036] 실시예는 애플리케이션(104c) 및/또는 운영 체제 커널(104b)의 실행의 비트까지-정확한 트레이스를 효율적으로 기록하기 위하여 프로세서의 공유 캐시(102b)를 이용한다. 이들 실시예는 프로세서(102)(공유 캐시(102b)를 포함)가 반-폐쇄(semi-closed) 또는 준-폐쇄(quasi-closed) 시스템을 형성한다는 관찰 위에 구축된다. 예를 들어, 프로세스에 대한 데이터의 일부(즉, 코드 데이터 및 런타임 애플리케이션 데이터)가 공유 캐시(102b)에 일단 로

딩되면, 프로세서(102)는 짧은 시간 동안에 아무 입력 없이도 반-폐쇄 또는 준-폐쇄 시스템으로서 자체적으로 실행될 수 있다. 특히, 프로세싱 유닛(102a) 중 하나 이상은 공유 캐시(102b)의 데이터 부분에 저장된 런타임 데이터를 사용하고 레지스터(102d)를 사용하여, 공유 캐시(102b)의 코드 부분으로부터 명령어를 실행한다.

[0037] (예를 들어, 프로세싱 유닛(102a)이 실행하고 있거나 실행할 예정이거나 실행할 수 있는 명령어가, 공유 캐시(102b)에 아직 있지 않은 코드 또는 런타임 데이터를 액세스하기 때문에) 프로세싱 유닛(102a)이 약간의 정보 유입을 필요로 할 때, "캐시 미스"가 일어나고, 그 정보를 시스템 메모리(103)로부터 공유 캐시(102b)로 가져온다. 예를 들어, 실행되는 명령어가 애플리케이션 런타임 데이터(103b) 내의 메모리 어드레스에서 메모리 동작을 수행할 때 데이터 캐시 미스가 발생하면, 그 메모리 어드레스로부터의 데이터를 공유 캐시(102b)의 데이터 부분의 캐시 라인 중 하나로 가져온다. 마찬가지로, 명령어가, 시스템 메모리(103)에 저장된 애플리케이션 코드(103a) 내의 메모리 어드레스에서 메모리 동작을 수행할 때 코드 캐시 미스가 발생하면, 그 메모리 어드레스로부터의 코드를 공유 캐시(102b)의 코드 부분의 캐시 라인 중 하나로 가져온다. 프로세싱 유닛(102a)은 이후 (예를 들어, 다른 캐시 미스 또는 캐싱되지 않은 읽기로 인해) 새로운 정보를 다시 공유 캐시(102b)로 가져올 때까지, 공유 캐시(102b) 내의 새로운 정보를 사용하여 실행을 계속한다.

[0038] 본 발명자는, 애플리케이션의 실행의 비트까지-정확한 표현을 기록하기 위하여, 트레이서(104a)는, 해당 애플리케이션의 스레드의 실행 동안의 공유 캐시(102b)로의 정보의 유입을 재생(reproduce)하기에 충분한 데이터를 기록할 수 있음을 관찰하였다. 이를 수행하는 것에 대한 제1 접근법은, 모든 캐시 미스 및 캐싱되지 않은 읽기(즉, 하드웨어 컴포넌트 및 캐싱할 수 없는 메모리로부터의 읽기)를, 각 데이터 조각을 공유 캐시(102b)로 가져온 때인 실행 중의 시간(예를 들어, 실행되는 명령어의 카운트 또는 어떤 다른 카운터를 사용)과 함께, 로깅(logging)함으로써, 공유 캐시(102b)로 가져온 모든 데이터를 기록하는 것이다.

[0039] 제1 접근법보다 상당히 더 작은 트레이스 파일을 생성하는 제2 접근법은, 각 프로세싱 유닛(102a)에 의해 "소비된(consumed)" 캐시 라인을 추적하고 기록하는 것이다. 본 명세서에서 사용되는 바와 같이, 프로세싱 유닛이 캐시 라인의 현재 값을 알고 있을 때, 프로세싱 유닛은 캐시 라인을 "소비하였다". 이는 해당 프로세싱 유닛이, 캐시 라인의 현재 값을 쓴 프로세싱 유닛이기 때문이거나, 해당 프로세싱 유닛이 캐시 라인에 대한 읽기를 수행했기 때문일 수 있다. 이 제2 접근법은, 프로세서(102)가 각 캐시 라인에 대하여 해당 캐시 라인을 소비한 하나 이상의 프로세싱 유닛(102a)을 식별할 수 있게 하는, 공유 캐시(102b)에 대한 확장을 수반한다.

[0040] 본 명세서의 실시예에 따르면, 제3 접근법은, 프로세서의 CCP를 이용하여, 파일(104d)에 기록할 "소비된" 캐시 라인의 서브 세트를 결정하는 것이며, 이는 여전히 공유 캐시(102b)의 활동(activity)이 재생될 수 있게 할 것이다. 이 제3 접근법은 제1 접근법과 제2 접근법 둘 다보다 트레이스 파일이 훨씬 작으므로 트레이싱 오버헤드가 상당히 더 낮아진다.

[0041] 본 명세서의 일부 실시예는 프로세싱 유닛/스레드에 대응하는 트레이스 데이터 스트림을 기록한다. 예를 들어, 트레이스 파일(104)은 각 프로세싱 유닛에 대한 하나 이상의 개별 트레이스 데이터 스트림을 포함할 수 있다. 이들 실시예에서, 각 트레이스 데이터 스트림 내의 데이터 패킷에는, 해당 데이터 패킷이 적용되는 프로세싱 유닛의 식별이 부재할 수 있는데, 이는 이러한 정보가 트레이스 데이터 스트림 자체에 기초하여 내재하기 때문이다. 이들 실시예에서, 컴퓨터 시스템(101)이 다수의 프로세서(102)(즉, 상이한 프로세서 소켓 내에 있음)를 포함하는 경우, 트레이스 파일은 상이한 프로세서(102) 내의 각각의 프로세싱 유닛(102a)에 대하여 하나 이상의 상이한 트레이스 데이터 스트림을 가질 수 있다. 복수의 데이터 스트림까지도 단일 스레드에 사용될 수 있다. 예를 들어, 일부 실시예는, 하나의 데이터 스트림을, 스레드에 의해 사용되는 프로세싱 유닛과 연관시키고, 하나 이상의 추가 데이터 스트림을, 스레드에 의해 사용되는 각 공유 캐시와 연관시킬 수 있다.

[0042] 다른 실시예에서, 트레이스 파일(104)은 프로세서(102)에 대한 단일 트레이스 데이터 스트림을 포함할 수 있고, 각 데이터 패킷에서 해당 데이터 패킷이 어느 프로세싱 유닛에 적용되는지 식별할 수 있다. 이들 실시예에서, 컴퓨터 시스템(101)이 다수의 프로세서(102)를 포함하는 경우, 트레이스 파일(104)은 다수의 프로세서(102) 각각에 대하여 개별 트레이스 데이터 스트림을 포함할 수 있다. 트레이스 파일의 레이아웃에 관계없이, 각 프로세싱 유닛(102a)에 대한 데이터 패킷은 일반적으로 다른 프로세싱 유닛에 독립적으로 기록되어, 상이한 프로세싱 유닛(102a)에서 실행된 상이한 스레드가 독립적으로 리플레이될 수 있게 한다. 그러나 트레이스 파일에는 명시적이든 내재적이든, 상이한 스레드 간의 부분적 서열화(ordering)를 제공하는 일부 정보가 포함될 수 있다.

[0043] 도 3은 CCP 데이터를 사용하여 캐시 기반 트레이스 기록을 수행하기 위한 방법(300)의 흐름도를 도시한다. 방법(300)은, 트레이서(104a)가 애플리케이션(104c) 및/또는 운영 체제 커널(104b)을 트레이싱함에 따라서, 프로세서(102)에 의해 수행되는 행위(act)를 포함할 수 있다. 프로세서(102)에 의해 수행되는 행위는 프로세서(102)의

하드-코딩된 로직, 소프트-코딩된 로직(즉, 마이크로코드(102c)) 및/또는 트레이서(104a), 운영 체제 커널(104b) 또는 하이퍼바이저와 같은 다른 소프트웨어 애플리케이션에 기초할 수 있다. 도 3은 행위의 시퀀스를 도시하지만, 실시예는 이들 행위 중 다수를 임의의 순서로 수행할 수 있으며, 일부는 심지어 병렬로 수행될 수 있음을 이해할 것이다. 이와 같이, 방법(300)에 도시된 행위의 시퀀스는 비제한적이다.

[0044] 도시된 바와 같이, 방법(300)은 캐시와 백킹 스토어 사이의 상호 작용을 검출하는 행위(301)를 포함한다. 일부 실시예에서, 행위(301)는, 복수의 캐시 라인 중 특정 캐시 라인과 하나 이상의 백킹 스토어 사이의 상호 작용을 야기하는 동작(operation)을 검출하는 단계를 포함한다. 예를 들어, 프로세싱 유닛(102a) 중 하나에서 애플리케이션(104c) 또는 운영 체제 커널(104b)의 스레드를 실행하는 동안, 프로세싱 유닛은 공유 캐시(102b)의 라인과 백킹 스토어(예를 들어, 시스템 메모리(103) 또는 다른 캐시) 사이의 상호 작용을 야기할 수 있다. 검출은, 예를 들어, 프로세서(102)의 마이크로코드(102c)의 실행에 기초하여 프로세서(102)에 의해 수행될 수 있다.

[0045] 방법(300)은 또한, 상호 작용을 야기한 프로세싱 유닛을 식별하는 행위(302)를 포함한다. 일부 실시예에서, 행위(302)는 동작을 야기한 복수의 프로세싱 유닛 중 특정 프로세싱 유닛을 식별하는 단계를 포함한다. 예를 들어, 마이크로코드(102c)의 실행에 기초하여, 프로세서(102)는 프로세싱 유닛(102a) 중 어느 것이 행위(301)에서 검출된 동작을 야기했는지 식별할 수 있다.

[0046] 방법(300)은 또한, 프로세싱 유닛에 대하여 로깅이 인에이블되었는지 여부를 결정하는 행위(303)를 포함한다. 일부 실시예에서, 행위(303)는 특정 프로세싱 유닛에 대하여 로깅이 인에이블되었다는 것을 결정하기 위하여 하나 이상의 로깅 제어 비트를 사용하는 것을 포함한다. 예를 들어, 프로세서(102)는, 행위(302)에서 식별된 프로세싱 유닛이, 하나 이상의 로깅 제어 비트에 기초하여 로깅을 인에이블시켰는지 여부를 결정할 수 있다. 로깅 제어 비트의 로깅을 사용하면 상이한 프로세싱 유닛의 로깅을 동적으로 인에이블 및 디스에이블할 수 있다. 따라서, 로깅 제어 비트를 사용함으로써, 트레이서(104a)는 어느 스레드가 트레이싱되고 있는지 및/또는 상이한 스레드의 실행의 어느 부분이 동적으로 트레이싱되고 있는지 동적으로 제어할 수 있다.

[0047] 로깅 제어 비트의 특정 형태 및 기능은 변할 수 있다. 일부 실시예에서, 예를 들어, 로깅 제어 비트는 제어 레지스터와 같은 레지스터(102d) 중 하나의 일부이다. 이들 실시예에서, 단일 로깅 제어 비트는 하나의 프로세싱 유닛(102a) 또는 복수의 프로세싱 유닛(102a)에 대응할 수 있다. 따라서, 레지스터(102d)는 (예를 들어, 모든 프로세싱 유닛 또는 특정 프로세싱 유닛 또는 프로세싱 유닛의 서브 세트에 대응하는) 단일 로깅 제어 비트를 포함하거나, (예를 들어, 각각이 하나 이상의 프로세싱 유닛에 대응하는) 복수의 로깅 제어 비트를 잠재적으로 포함할 수 있다. 다른 실시예에서, 로깅 제어 비트는 캐시와 백킹 스토어 사이의 상호 작용을 야기한 명령어에 대응하는 어드레스 공간 식별자(address space identifier, ASID) 및/또는 프로세스 컨텍스트 식별자(process-context identifier, PCID)를 포함하거나, 아니면 이와 연관된다. 따라서, 예를 들어, 방법(300)은, 프로세싱 유닛이 하나 이상의 특정 ASID/PCID와 연관된 명령어를 실행하고 있을 때만 프로세싱 유닛을 트레이싱할 수 있다. 이러한 방식으로, 방법(300)은 지정된 어드레스 공간 및/또는 특정 프로세스 컨텍스트만을 기록할 수 있다. 조합 또한 가능하다. 예를 들어, 로깅 제어 비트는 레지스터(102d) 중 하나 이상에 저장될 수 있지만, 현재 ASID/PCID 값에 기초하여 설정(set)/소거(clear)될 수 있다. 로깅 제어 비트의 형태에 관계없이, 일부 실시예는 컨텍스트 스위치에서 로깅 제어 비트를 설정/소거할 수 있어서, 방법(300)이 특정 스레드만을 트레이싱할 수 있게 한다.

[0048] 방법(300)은 또한, 캐시 라인이 로깅에 참여하는지 여부를 결정하는 행위(304)를 포함한다. 일부 실시예에서, 행위(304)는, 적어도 특정 프로세싱 유닛에 대하여 로깅이 인에이블되는 것에 기초하여, 특정 캐시 라인이 로깅의 참여자(participant in logging)인지 여부를 결정하는 단계를 포함한다. 예를 들어, 프로세서(102)는 행위(301)에서 검출된 동작에 관련된 캐시 라인이 로깅에 관련되는지 여부를 결정할 수 있다. 나중에 더 상세히 논의되는 바와 같이, 공유 캐시(102b) 내의 비트의 사용 또는 캐시 웨이-로킹(way-locking)의 사용과 같은, 검출을 위하여 사용될 수 있는 몇 가지 메커니즘이 있다.

[0049] 방법(300)은 또한, 트레이스에 로깅될 데이터가 있음을 식별하기 위하여 CCP를 사용하는 행위(305)를 포함한다. 예를 들어, 프로세서(102)는 동작의 결과로서 캐시 상태에서 어떤 천이가 발생했는지, 그리고 그러한 천이가 데이터의 로깅을 정당화하는지를 결정하기 위하여 CCP를 참조할 수 있다. 트레이스 데이터를 식별하기 위하여 CCP를 사용하는 자세한 예는 나중에 도 6a 내지 도 9b와 관련하여 제공된다.

[0050] 방법(300)은 또한, CCP를 사용하여 적절한 데이터를 트레이스에 로깅하는 행위(306)를 포함한다. 일부 실시예에서, 행위(306)는 데이터가 트레이스에 로깅되게 하는 단계를 포함하고, 데이터는 동작을 리플레이하는 데 사용될 수 있다. 데이터가 트레이스 파일에 로깅되어야 할 때, 특정 프로세싱 유닛에 대응하는 트레이스 데이터 스

트림 또는 일반적으로 프로세서(102)에 대응하는 트레이스 데이터 스트림과 같은, 적절한 트레이스 데이터 스트림에 하나 이상의 데이터 패킷이 추가될 수 있다. 적절한 트레이스 데이터 스트림이 프로세서(102)에 일반적으로 대응하면, 하나 이상의 데이터 패킷은 특정 프로세싱 유닛을 식별할 수 있다. 트레이스 데이터 스트림이 일반적으로 프로세서(102)에 대응하는 경우, 데이터 스트림에서의 데이터 패킷의 내재된 순서 자체는, 다수의 데이터 스트림이 사용되는 경우 이용 불가능할 수 있는 일부 추가 서열화 정보를 제공한다는 점에 주목하여야 한다.

[0051] 공유 캐시(102b)가 다수의 캐시 레벨을 포함할 때, 일부 실시예에서, 방법(300)은 시스템 메모리(103)와 상호작용하는 캐시 레벨에서 동작하는데, 이는 캐시 미스를 처리하는 캐시 레벨이 해당 캐시 레벨이기 때문이라는 것을 주목하여야 한다. 이 레벨에서 동작하면 각 프로세싱 유닛(102a)의 캐시 활동이 중복(즉, 유닛의 활동을 두 번 이상 나타냄)되지 않고 표현될 수 있게 한다. 따라서, 예를 들어, 컴퓨터 시스템(101)이 2 개의 프로세서(102)(즉, 2 개의 프로세서 소켓)를 포함하고 소켓 당 하나의 "포괄적(inclusive)" L3 캐시뿐만 아니라, L3 캐시 아래의 "포괄적" L2 캐시를 포함한다면, 일부 실시예에서 방법(300)은 L3 캐시에서 동작한다. 방법(300)은 또한, 다수의 캐시 레벨에서 동작할 수 있다. 예를 들어, 컴퓨터 시스템(101)이 하나의 프로세서(102)(즉, 하나의 프로세서 소켓)를 포함하고 소켓에 대한 하나의 "배타적(exclusive)" L3 캐시뿐만 아니라 L3 캐시 아래의 "포괄적" L2 캐시를 포함한다면, 방법(300)이 동작할 수 있는 캐시는 L3 캐시 및 L2 캐시 둘 다이다. 혼합 포괄적인/배타적 거동을 나타내는 캐시 내 로깅의 추가 예는 아래에서 논의된다.

[0052] 행위(304)와 관련하여 위에서 언급된 바와 같이, 캐시 라인이 "로깅 참여자"인지 여부를 결정하기 위하여 프로세서(102)에 의해 사용될 수 있는 몇몇 메커니즘이 있다. 하나는, 플래그로서, 프로세싱 유닛 식별자로서, 또는 프로세서 인덱스로서 사용될 수 있는 하나 이상의 추가 "어카운팅 비트"로 공유 캐시(102b)의 각 라인을 확장하는 것이다. 이들 "어카운팅 비트"를 제어하기 위한 로직은 프로세서의 마이크로코드(102c)의 일부일 수 있다.

[0053] 이 실시예를 설명하기 위하여, 도 4a는, 도 2의 공유 캐시(200)와 유사하게, 하나 이상의 추가 어카운팅 비트(401)로 캐시 라인(404) 각각을 연장하는 예시적인 공유 캐시(400a)를 도시한다. 따라서, 각 캐시 라인(404)은 어카운팅 비트(401), 종래의 어드레스 비트(402), 및 값 비트(403)를 포함한다.

[0054] 일부 구현에서, 각 캐시 라인의 어카운팅 비트(401)는, 캐시 라인이 트레이스 로깅에 참여하는지 여부를 표시하기 위하여 프로세서(102)에 의해 사용되는 플래그(즉, 온 또는 오프)로서 기능하는 단일 비트를 포함한다. 프로세서의 CCP가 충분한 그래뉼래리티를 갖는다면(예를 들어, 각 캐시 라인이 각 프로세싱 유닛과 관련됨에 따라, 또는 캐시 라인의 코히어런스 상태를 소유하는 프로세싱 유닛에 대한 인덱스를 참조하여, CCP가 각 캐시 라인에 대한 코히어런스 상태를 추적한다면), 이 단일 비트는 로버스트한 완전히 결정론적인(fully-deterministic) 트레이스(즉, 트레이싱된 실행의 완전 재구성 가능성(full reconstruct-ability)을 보장하는 트레이스)를 기록하는 것을 용이하게 하기에 충분할 수 있다.

[0055] 다른 구현에서, 각 라인의 어카운팅 비트(401)는 복수의 비트를 포함한다. 복수의 비트는 여러 방식으로 사용될 수 있다. 본 명세서에서 "유닛 비트(unit bit)"라고 지칭하는 한 가지 접근법을 사용하면, 각 캐시 라인의 어카운팅 비트(401)는, 프로세서(102)의 프로세싱 유닛(102a)의 수(예를 들어, 프로세서(102)가 하이퍼-스레딩을 지원한다면 논리적 프로세싱 유닛의 수, 또는 하이퍼-스레딩이 지원되지 않는다면 물리적 프로세싱 유닛의 수)와 동일한 수의 유닛 비트를 포함할 수 있다. 이들 유닛 비트는 프로세서(102)에 의해 사용되어서, 어느 하나 이상의 특정 프로세싱 유닛이 캐시 라인을 소비했는지 추적할 수 있다(또는 캐시 라인이 소비되지 않았다면, 프로세싱 유닛 중 어느 것도 캐시 라인을 소비하지 않았다고 표기(note)할 수 있다). 따라서, 예를 들어, 2 개의 프로세싱 유닛(102a)에 의해 공유되는 공유 캐시(102b)는 각 캐시 라인에 대하여 2 개의 유닛 비트를 포함할 수 있다. 각 캐시 라인에 추가되는 이들 유닛 비트와 관련하여, 실시예는 프로세서의 마이크로코드(102c)를 확장하여, 각 프로세싱 유닛 대신에 캐시 라인 내의 현재 값이 로깅되었는지(즉, 트레이스 파일(104d)에 로깅되었는지) 아니면 이와 달리 프로세싱 유닛에 알려져 있는지를 추적하기 위하여 이들 유닛 비트를 이용한다. 프로세서의 CCP가 더 거친(coarser) 그래뉼래리티를 가진다면(예를 들어, CCP가 캐시 라인의 레벨에서만 코히어런스 상태를 추적한다면), 이들 유닛 비트는 로버스트한 트레이스를 용이하게 하기 위하여 추가 정보를 제공할 수 있다. 예를 들어, 캐시 라인이 CCP에 의해 공유 또는 배타로 마킹된다면, 어느 프로세싱 유닛이 캐시 라인을 공유하는지 또는 어느 프로세싱 유닛이 배타성을 갖는지를 식별하기 위하여 유닛 비트가 사용될 수 있다.

[0056] 본 명세서에서 "인덱스 비트(index bit)"라고 지칭되는 다른 접근법을 사용하면, 각 캐시 라인의 어카운팅 비트(401)는, 로깅에 참여하는 컴퓨터 시스템(101)의 프로세서(102)의 프로세싱 유닛(102a) 각각에 대한 인덱스를 나타내기에 충분한 수의 인덱스 비트를, "예약된(reserved)" 값(예를 들어, -1)과 함께, 포함할 수 있다. 예를

들어, 컴퓨터 시스템(101)의 프로세서(102)가 128 개의 프로세싱 유닛(102a)을 포함하면, 이들 프로세싱 유닛은 캐시 라인 당 7 개의 인덱스 비트만을 사용하여 인덱스 값(예를 들어, 0 내지 127)에 의해 식별될 수 있다. 일부 실시예에서, 프로세서가 캐시 라인을 로깅하지 않았다는 것을 표시하기 위하여 하나의 인덱스 값(예를 들어, "무효(invalid)")이 예약된다. 따라서, 이는 7 개의 인덱스 비트가 실제로 127 개의 프로세싱 유닛(102a) 및 예약된 값을 나타낼 수 있다는 것을 의미할 것이다. 예를 들어, 이진 값 0000000 내지 1111110은 인덱스 장소 0 내지 126(십진수)에 대응할 수 있고, 이진 값 1111111(예를 들어, 해석에 따라 십진수 -1 또는 127)은 "무효"에 대응하여, 프로세서가 대응 캐시 라인을 로깅하지 않았음을 표시할 수 있으며, 이 표기법(notation)은 구현에 따라 다를 수 있다. 따라서, 유닛 비트는, 캐시 라인이 트레이스 로깅에 참여하고 있는지(예를 들어, -1 이외의 값) 추적하기 위하여, 그리고 캐시 라인을 소비한 특정 프로세싱 유닛(예를 들어, 가장 최근에 캐시 라인을 소비한 프로세싱 유닛)에 대한 인덱스로서, 프로세서(102)에 의해 사용될 수 있다. 이 제2 접근법은 공유 캐시(102b)에서의 오버헤드가 거의 없이 다수의 프로세싱 유닛을 지원하는 이점을 가지며, 제1 접근법보다 그래뉼래리티가 적다(즉, 한 번에 하나의 프로세싱 유닛만이 식별된다)는 단점이 있다. 다시, 프로세서의 CCP가 더 거친 그래뉼래리티를 갖는다면(예를 들어, CCP가 캐시 라인 레벨에서만 코히어런스 상태를 추적한다면), 이들 인덱스 비트는 로버스트한 트레이스를 용이하게 하기 위하여 추가 정보를 제공할 수 있다. 예를 들어, 캐시 라인이 CCP에 의해 공유 또는 배타로 마킹된다면, 캐시 라인을 공유하는 적어도 하나의 프로세싱 유닛을 식별하기 위하여 또는 어느 프로세싱 유닛이 배타성을 가지는지 식별하기 위하여 인덱스 비트가 사용될 수 있다.

[0057] 캐시 라인이 로깅 참여자인지 여부를 결정하기 위하여 프로세서(102)에 의해 사용될 수 있는 또 다른 메커니즘은 도 4a와 관련하여 논의된 개념을 이용할 수 있지만, 추가적인 어카운팅 비트(401)로 각 캐시 라인을 확장하지 않고도 가능하다. 대신에, 이 메커니즘은 어카운팅 비트를 저장하기 위하여 캐시 라인(404) 중 하나 이상을 예약한다. 도 4b는 메모리 어드레스(402) 및 값(403)을 저장하는 종래의 캐시 라인(405)뿐만 아니라, 종래의 캐시 라인(405)에 적용되는 어카운팅 비트를 저장하기 위한 하나 이상의 예약된 캐시 라인(406)을 포함하는 공유 캐시(400b)의 예를 도시한다. 예약된 캐시 라인(406)의 비트는, 종래의 캐시 라인(405) 중 상이한 캐시 라인에 각각 대응하는 어카운팅 비트의 상이한 그룹에 할당된다. 이들 어카운팅 비트의 그룹은 구현에 따라 플래그 비트, 유닛 비트, 또는 인덱스 비트로서 기능할 수 있다.

[0058] 캐시 라인이 로깅 참여자인지 여부를 결정하기 위하여 프로세서(102)에 의해 사용될 수 있는 또 다른 메커니즘은 연관 캐시 및 웨이-로킹을 이용하는 것이다. 프로세서의 공유 캐시(102b)는 일반적으로 시스템 메모리(103)보다 훨씬(종종 여러 자리수만큼) 작기 때문에, 따라서 공유 캐시(102b)에 라인이 있는 것보다 일반적으로 시스템 메모리(103)에 훨씬 더 많은 메모리 장소(memory location)가 존재한다. 이와 같이, 각 프로세서는 시스템 메모리의 다수의 메모리 장소를 캐시의 라인에 매핑하기 위한 메커니즘을 정의한다. 프로세서는 일반적으로 직접(direct) 매핑과 연관(associative) 매핑이라는 두 가지 일반적인 기법 중 하나를 이용한다. 직접 매핑을 사용하면, 시스템 메모리(103)의 상이한 메모리 장소가 캐시의 단 하나의 라인에 매핑되어, 각 메모리 장소는 오직 캐시의 특정 라인에만 캐싱될 수 있다.

[0059] 반면에, 연관 매핑을 사용하면, 시스템 메모리(103)의 상이한 장소가 공유 캐시(102b)의 다수의 라인 중 하나에 캐싱될 수 있다. 도 5는 연관 캐시 매핑의 예(500)를 도시한다. 여기서, 캐시(502)의 캐시 라인(504)은 2 개의 캐시 라인으로 각각 이루어진 상이한 어드레스 그룹으로 논리적으로 파티셔닝되고, 어드레스 그룹은, 2 개의 캐시 라인(504a 및 504b)의 제1 그룹(인덱스 0으로 식별됨) 및 2 개의 캐시 라인(504c 및 504d)의 제2 어드레스 그룹(인덱스 1로 식별됨)을 포함한다. 어드레스 그룹의 각 캐시 라인은, 캐시 라인(504a)이 인덱스 0, 웨이 0으로 식별되고, 캐시 라인(504b)이 인덱스 0, 웨이 1로 식별되는 등과 같이 되도록, 상이한 "웨이(way)"와 연관된다. 추가로 도시된 바와 같이, 메모리 장소(503a, 503c, 503e 및 503g)(메모리 인덱스 0, 2, 4 및 6)는 인덱스 0에 매핑된다. 이와 같이, 시스템 메모리에서의 이들 장소 각각은 인덱스 0에서의 그룹 내의 임의의 캐시 라인(즉, 캐시 라인(504a 및 504b))에 캐싱될 수 있다. 도시된 맵핑의 특정 패턴은 단지 예시적이고 개념적인 목적을 위한 것이며, 메모리 인덱스가 캐시 라인에 맵핑될 수 있는 유일한 방식으로 해석되어서는 안 된다.

[0060] 연관 캐시는 일반적으로 N-웨이 연관 캐시로 지칭되며, 여기서 N은 각각의 어드레스 그룹에서의 "웨이"의 수이다. 따라서, 도 5의 캐시(500)는 2-웨이 연관 캐시로 지칭될 수 있다. 프로세서는 일반적으로 N이 2의 거듭 제곱(예를 들어, 2, 4, 8 등)인 N-웨이 캐시를 구현하며, (본 명세서의 실시예는 임의의 특정 N-값, 또는 N-값의 서브 세트로 제한되지는 않지만) 4 및 8의 N 값이 일반적으로 선택된다. 특히, 1-웨이 연관 캐시는, 각 어드레스 그룹에 하나의 캐시 라인만 포함되므로, 일반적으로 직접 매핑(direct-mapped) 캐시와 동등하다. 또한 N이 캐시의 라인 수와 같으면, 캐시는 캐시의 모든 라인을 포함하는 단일 어드레스 그룹을 포함하므로 완전 연관 캐시(fully associative cache)라고 지칭된다. 완전 연관 캐시에서 임의의 메모리 장소는 캐시의 임의의 라인에

캐싱될 수 있다.

[0061] 도 5는 일반적인 원리를 설명하기 위하여 시스템 메모리 및 캐시의 단순화된 뷰를 나타낸다는 점에 주목한다. 예를 들어, 도 5는 개별 메모리 장소를 캐시 라인에 매핑하지만, 캐시의 각 라인은 일반적으로 시스템 메모리의 다수의 어드레서블(addressable) 장소에 관한 데이터를 저장한다는 것을 이해할 것이다. 따라서, 도 5에서, 시스템 메모리(501)의 각 장소(503a 내지 503h)는 실제로 복수의 어드레서블 메모리 장소를 나타낼 수 있다. 부가적으로, 매핑은 시스템 메모리(501)의 실제 물리적 어드레스와 캐시(502)의 라인 사이의 매핑일 수 있거나, 가상 어드레스의 중간 레이어를 사용할 수 있다.

[0062] 캐시 라인이 웨이-로킹의 사용을 통해 로깅에 참여하는지 여부를 결정하기 위하여 연관 캐시가 사용될 수 있다. 웨이-로킹은 어떤 목적을 위해 캐시에서 특정 웨이(way)를 로킹(lock)하거나 예약(reserve)한다. 특히, 본 명세서의 실시예는 트레이싱되고 있는 스레드에 대한 하나 이상의 웨이를 예약하기 위하여 웨이-로킹을 사용하고, 따라서 로킹/예약된 웨이는 그 스레드의 실행과 관련된 캐시 미스를 저장하기 위하여 배타적으로 사용된다. 따라서, 도 5를 다시 참조하면, 트레이싱되는 스레드에 대하여 "웨이 0"이 로킹된다면, 캐시 라인(504a 및 504c)(즉, 인덱스 0, 웨이 0 및 인덱스 1, 웨이 0)은 그 스레드의 실행에 관한 캐시 미스에 대하여 배타적으로 사용될 것이고, 나머지 캐시 라인은 다른 모든 캐시 미스에 대하여 사용될 것이다. 따라서, 특정 캐시 라인이 로깅의 참여자인지 여부를 결정하기 위하여, 프로세서(102)는 단지, 캐시 라인이, 트레이싱되고 있는 스레드에 대하여 예약된 웨이의 일부인지 여부를 결정하면 된다.

[0063] 도 6a 내지 도 6d는 도 1, 도 2, 도 4a, 도 4b 및 도 5의 맥락에서 도 3의 방법(300)을 적용한 구체적인 예(600)를 도시한다. 도 6a는 공유 캐시(102b)의 단일 라인 상에서 4 개의 프로세싱 유닛(102a)(즉, P0 내지 P3)에 의한 읽기 및 쓰기 활동을 보여주는 제1 표(600a)를 도시한다. 도 6b는 이들 읽기 및 쓰기에 기초하여 추적된(예를 들어, 프로세서의 CCP를 사용하여 추적된) 캐시 코히어런스 상태의 일 실시예를 표시하는 제2 표(600b)를 도시한다. 도 6c는 만약 어카운팅 비트가 사용된다면, (도 4a 및 4b와 관련하여 설명된 바와 같이) 공유 캐시(102b)의 어카운팅 비트에 저장될 수 있는 것을 보여주는 제3 표(600c)를 도시한다. 오직 한 유형의 어카운팅 비트(즉, 라인 당 유닛 비트, 라인 당 인덱스 비트 또는 라인 당 플래그 비트)만이 전형적으로 사용될 것이지만, 표(600c)는 설명의 완전성을 위하여 유닛 비트(603), 인덱스 비트(604), 및 플래그 비트(605) 각각을 도시한다. 마지막으로, 도 6d는 각 동작과 관련하여 트레이스 파일(104d)에 잠재적으로 쓰여질 수 있는 예시적인 유형의 로그 데이터(606)를 보여주는 제4 표(600d)를 도시한다.

[0064] 설명의 간략화를 위하여, 표(600a)는 한 번에 오직 하나의 프로세싱 유닛(102a)에 의한 동작을 도시하지만, 여기에 설명된 원리는 동시(concurrent) 활동(예를 들어, 2개 이상의 프로세싱 유닛에 의한 동일한 캐시 라인의 동시 읽기)이 있는 상황에 적용됨을 이해할 것이다. 또한, 도 6a 내지 도 6d와 관련하여 설명된 예들은 추적이 프로세싱 유닛(P0 내지 P2)에 대하여 인에이블되고 프로세싱 유닛(P3)에 대하여 디스에이블되는 것으로 가정한다. 예를 들어, 위에서 논의된 바와 같이, 이것은 제어 레지스터의 복수의 비트와 같은 각 프로세싱 유닛에 대응하는 비트에 의해 제어할 수 있다.

[0065] 초기에, 설명의 편의를 위하여, 이 예는 위에서 논의된 CCP(즉, MSI, MESI 및 MOESI)에서 사용된 캐시 라인 상태(즉, 수정(Modified), 소유(Owned), 배타(Exclusive), 공유(Shared) 및 무효(Invalid))로부터 도출된 단순화된 캐시 라인 상태를 사용할 것이다. 이 단순화에서, 이들 상태는 "읽기" 상태(즉, 캐시 라인이 읽혀짐) 또는 "쓰기" 상태(즉, 캐시 라인에 쓰기됨)에 매핑된다. 아래 표 1은 이러한 맵핑의 한 예를 보여준다. 이러한 매핑은 예제로만 사용되며 제한적이지 않다는 점에 유의해야 한다. 예를 들어, 여기에 논의된 것 이외의 CCP 및 상태가 존재할 수 있으며, 당업자는 본 명세서의 개시를 고려하여, 유사한 맵핑이 많은 상이한 CCP로 이루어질 수 있음을 인식할 것이다.

표 1

[0066]

프로토콜 상태	매핑된 상태
수정	쓰기
소유	읽기
배타	쓰기
공유	읽기
무효	매핑 없음 - 캐시 라인이 빈 것으로 간주됨

[0067] 특히, 실시예는 프로세서(102)로부터 무슨 데이터가 이용 가능한지에 따라 그리고/또는 구현 선택에 기초하여,

다양한 레벨로 CCP 데이터를 로깅할 수 있다. 예를 들어, CCP 데이터는, "매핑된" CCP 상태(예를 들어, 표 1에 표시된 것)에 기초하여, 프로세서(102)에 의해 보일 수 있는 실제 CCP 상태(예를 들어, 수정, 소유, 배타, 공유 및/또는 무효)에 기초하여, 그리고/또는 심지어 프로세서(102)에 의해 전형적으로 보이지 않을 수 있는 하위(lower) 레벨 "원시(raw)" CCP 데이터에 기초하여 로깅될 수 있다.

[0068] 도 6a 내지 도 6d를 참조하면, 표(600a)는 식별자(ID)를 도시하는 제1 열(601)을 포함하며, 이는 동작들 사이에서의 글로벌 순서를 지정하는 데 사용된다. 표(600a)는 또한 각각이 프로세싱 유닛 중 하나에 대응하는 4 개의 추가 열(602a 내지 602d)을 포함한다. 단순함을 위하여, 이 예는 글로벌 ID를 사용하지만, 실제로는 각 프로세싱 유닛이 일반적으로 자신의 독립적인 식별자 세트를 사용하여 동작들을 서열화(order)할 것임을 이해할 것이다. 이들 ID는 명령어 카운트(instruction count, IC), 또는 "점프 카운트(jump count)" + 프로그램 카운터와 같은 동작들 사이의 서열화를 지정하기 위한 임의의 다른 적절한 메커니즘을 포함할 수 있다. 이 예에서는 MSI, MESI 및 MOESI CCP와 일치하는 방식으로 메모리를 사용하지만 단순함을 위하여 "수정", "공유" 및 "무효" 상태만 사용한다는 점에 유의해야 한다. 그러나 일부 CCP는 트레이스 항목(entry)을 강력하게 서열화하기 위하여 트레이스에(예를 들어, 모든 패킷에 또는 가끔의 패킷에) 또한 기록될 수 있는 고유하고/하거나 단조 증가하는 자신만의 ID를 제공할 수 있다는 것을 주목하여야 한다. CCP가 그러한 ID를 제공하지 않더라도, 소켓 타이머(예를 들어, TSC)의 값 또는 다른 서열화 가능한 ID가 잠재적으로 사용될 수 있다.

[0069] 표(600a)에 도시된 바와 같이, 식별자 ID[0]에서 프로세싱 유닛(P0)은 읽기를 수행하고, 이는 캐시 미스를 야기하여, 데이터(DATA[1])를 캐시 라인으로 가져오게 한다. 이에 대응하여, 표(600b)는 프로세서의 CCP는 캐시 라인이 이제 P0에 의해 "공유"되었다고 표기하는 것을 도시한다. 표(600c)는 만일 유닛 비트(603)가 사용된다면, 유닛 비트는 프로세싱 유닛(P0)이 캐시 라인을 소비(즉, 읽기)하였음을 표시하고(그리고 프로세싱 유닛(P1 내지 P3)은 소비하지 않았음을 표시함), 만일 인덱스 비트(604)가 사용된다면, 인덱스 비트는 P0가 캐시 라인을 소비하였다고 표시하고, 만일 플래그 비트(605)가 사용된다면, 플래그 비트는 어떤 프로세싱 유닛이 캐시 라인을 소비하였음을 표시한다고 도시한다. 이 상태가 주어지면, 행위(303)에서 프로세서(102)는 P0에 대하여 로깅이 인에이블되었다고 결정하고, 행위(304)에서 (즉, 유닛 비트(603), 인덱스 비트(604), 플래그 비트(605) 또는 웨이-로킹을 사용하여) 캐시 라인이 로깅에 참여한다고 결정할 것이다. 따라서, 행위(306)에서, 프로세서(102)는 필요한 경우, CCP를 이용하여 적절한 데이터를 트레이스 파일에 로깅할 것이다. 여기서 캐시 라인은 무효(빈(empty)) 상태로부터 읽기(표 600a)/공유(표 600b) 상태로 되고 있기 때문에, 데이터는 로깅되어야 한다. 표(600d)의 로그 데이터(606)에 도시된 바와 같이, 프로세서(102)는 필요한 경우에(즉, 데이터 패킷이 프로세싱 유닛 당 개별 데이터 스트림에 또는 단일 데이터 스트림에 로깅되고 있는지에 따라) 프로세싱 유닛(P0); 캐시 라인 어드레스(@); 명령어 카운트 또는 어떤 다른 카운트; 및 캐시 라인으로 가져온 데이터(DATA[1])를 표기할 수 있다. 위에서 논의된 바와 같이, 명령어 카운트는 일반적으로 프로세싱 유닛 특유의 값일 것이지만, 단순함을 위하여 표(600d)는 대응하는 글로벌 ID(즉, 이 예에서 IC[0])를 참조하여 명령어 카운트를 참조한다.

[0070] 캐시 라인 어드레스(@) 및 데이터(예를 들어, DATA[1])는 일부 실시예에서 트레이스 파일(104d) 내에서 압축될 수 있음에 주목한다. 예를 들어, 메모리 어드레스는 이전에 기록된 메모리 어드레스에서 "상위(high)" 비트를 (명시적으로 또는 암시적으로) 참조함으로써 메모리 어드레스의 "상위" 비트를 기록하지 않음으로써 압축될 수 있다. 데이터 값의 비트를 복수의 비트를 각각 포함하는 복수의 그룹으로 그룹화하고, 각 그룹을 대응하는 "플래그" 비트와 관련시킴으로써 데이터가 압축될 수 있다. 그룹이 특정 패턴(예를 들어, 모두 0, 모두 1 등)과 같으면 플래그 비트를 설정할 수 있으며 해당 비트 그룹을 트레이스에 저장할 필요가 없다.

[0071] 다음에, 표(600a)는 ID[1]에서 프로세싱 유닛(P1)이 데이터(DATA[1])를 읽어서, 캐시 라인에서 읽기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0 및 P1에 의해 "공유"된다고 표기하는 것을 도시한다. 표(600c)는 프로세싱 유닛(P0 및 P1)이 캐시 라인을 소비하였거나(유닛 비트 603), P1이 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비하였음(플래그 비트 605)을 보여주고 있다. 인덱스 비트(604)가 여전히 P1 대신 P0를 참조하는 것이 또한 옳을 것이라는 것에 주목한다. 표(600d)는 CCP를 사용하여 프로세서(102)가 동작의 기록이 로깅되어야 한다고 결정하는 것을 도시한다. 도시된 바와 같이, 프로세서(102)는 프로세싱 유닛(P1); 캐시 라인 어드레스(@); 명령어 카운트(IC[1]); 캐시 라인이 읽기(공유) 상태에서 읽기(공유) 상태로 되었다는 것; P0가 이전 캐시 라인에 대한 이전 액세스 권한을 가졌지만, 이제 P0 및 P1이 액세스 권한을 가진다는 것을 표기할 수 있다.

[0072] 다음으로, 표(600a)는, ID[2]에서 프로세싱 유닛(P0)이 데이터(DATA[2])를 쓰면서, 캐시 라인에 쓰기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0에 의해 "수정"되어 있고, P1에 대하여 "무효"라고 표기하는 것을 도시한다. 표(600c)는 오직 프로세싱 유닛(P0)이 캐시 라인을 소비하였거나(즉, 값을

업데이트하였거나)(유닛 비트 603), 프로세싱 유닛(P0)이 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비했다는 것을 보여준다(플래그 비트 605). 표(600d)는 캐시 라인이 쓰기/수정되었기 때문에 CCP를 사용하여 프로세서(102)가 동작의 기록이 로깅될 필요가 있다고 결정하는 것을 보여준다. 도시된 바와 같이, 프로세서(102)가 프로세싱 유닛(P0); 캐시 라인 어드레스(@); 명령어 카운트(IC[2]); 캐시 라인이 읽기(공유) 상태로부터 쓰기(수정) 상태로 되었다는 것; 및 P0 및 P1이 이전 캐시 라인에 대한 이전 액세스 권한을 가졌으나, 이제 P0만이 액세스 권한을 가진다는 것을 표기할 수 있다.

[0073] 어느 프로세싱 유닛이 캐시 라인에 대한 이전 액세스를 가졌는지에 관한 정보는, 표(600b)에 도시된 CCP 상태를 사용하여 발견될 수 있다는 것을 주목하여야 한다. 그러나, 어떤 CCP(예를 들어, 캐시 라인 레벨에서만 코히어런스 상태를 추적하는 CCP)는 그렇게 하기에 충분한 정보를 유지하지 않을 수 있다는 것을 주목하여야 한다. 대안적으로 유닛 비트(603)가 사용된다면, 이 정보가 유닛 비트로부터 도출될 수 있다. 따라서, 도 6d에 도시된 로그 데이터(606)는 이 정보를 유지하는 로버스트한 CCP, 또는 유닛 비트(603)의 사용을 가정한다.

[0074] 이들 중 어느 것도 사용되지 않으면(예를 들어, CCP가 로버스트하지 않고, 인덱스 비트(604), 플래그 비트(605) 또는 웨이-로킹이 유닛 비트(603) 대신에 사용된다면), 로그 데이터(606)는 덜 철저하거나 더 클 수 있다. 제1 예로서, CCP가 캐시 라인 레벨에서만 코히어런스 상태를 추적하고 인덱스 비트(604)가 사용된다면, 이 둘은 캐시 라인 상태가 (모든 프로세싱 유닛에 대하여) 무효인 것, (수정한 프로세싱 유닛의 인덱스와 함께) 수정된 것, (배타 상태로 만든 프로세싱 유닛의 인덱스와 함께) 배타인 것, 또는 공유인 것(및 모든 프로세싱 유닛이 액세스를 가질 수 있음)을 식별하기 위하여 사용될 수 있다. 이는 캐시 라인을 공유로부터 수정 또는 배타로 변경하여야 할 시간일 때, 캐시 라인을 공유하기 위하여 더 세밀한(granular) CCP에 의해 알려졌을 프로세싱 유닛만이 아니라, 모든 프로세싱 유닛에게 통지되어야 한다는 단점을 가지고, 더 간단한 하드웨어 구현을 초래할 수 있다. 제2 예로서, 인덱스 비트(604)는 캐시 라인을 액세스한 마지막 프로세싱 유닛을 식별하는 데 사용될 수 있다. 그런 다음, 캐시가 포괄적이라면(즉, 많은 읽기가 L2 또는 L1 캐시 레벨에서의 액세스 뒤에 숨겨짐), 프로세싱 유닛이 동일한 캐시 라인을 읽고 있는 경우에도 L3 캐시는 동일한 프로세싱 유닛으로부터 비교적 적은 반복 요청을 볼 수 있다. 읽기->읽기에 대한 모든 인덱스 변경을 로깅한 다음, 읽기->쓰기, 쓰기->쓰기 및 쓰기->읽기가 또한 인덱스를 로깅하게 하는 것은, 잠재적으로 약간 더 큰 트레이스라는 비용으로, 유닛 비트(603)의 사용과 동일한 데이터를 제공한다. 제3 예로서, 각각의 캐시 라인은 단일 플래그 비트를 포함할 수 있지만, CCP는 캐시 라인의 코히어런스 상태를 소유하는 프로세싱 유닛에 대한 인덱스를 참조하여 각 캐시 라인에 대한 코히어런스 상태를 추적할 수 있다. 여기서, 트레이스는 유닛 비트가 사용되었거나 CCP가 개별 프로세싱 유닛을 추적한 경우보다 더 많은 캐시 라인 이동을 기록할 수 있지만, 트레이스는 여전히 완전히 결정론적일 수 있다. 각 프로세싱 유닛에 관한 정보를 가질 때와 프로세서 인덱스에 관한 정보만을 가질 때의 트레이스 파일 크기를 간략하게 비교한 것이 이하 도 9a 및 도 9b와 관련하여 나타난다.

[0075] 도 6a로 되돌아가면, 표(600a)는 ID[3]에서 프로세싱 유닛(P1)이 데이터(DATA[2])를 읽어서 캐시 라인으로부터 읽기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0 및 P1에 의해 "공유"되었음을 표기한다는 것을 도시한다. 표(600c)는 프로세싱 유닛(P0 및 P1)이 캐시 라인을 소비하였거나(유닛 비트 603), P1이 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비하였음을(플래그 비트 605) 보여주고 있다. 인덱스 비트(604)가 여전히 P1 대신 P0를 참조하는 것이 또한 정확할 것이라는 것에 주목하여야 한다. 표(600d)는, 캐시 라인이 쓰기(수정) 상태로부터 읽기(공유) 상태로 되었기 때문에, CCP를 사용하여, 프로세서(102)가 동작의 기록이 로깅될 필요가 있다고 결정하는 것을 보여준다. 도시된 바와 같이, 프로세서(102)는 프로세싱 유닛(P1); 캐시 라인 어드레스(@); 명령어 카운트(IC[3]); 캐시 라인이 쓰기(수정) 상태로부터 읽기(공유) 상태로 된 것; P0가 이전 캐시 라인에 대한 이전 액세스 권한을 갖고 있었지만 이제 P0 및 P1이 액세스 권한을 가진다는 것을 표기할 수 있다.

[0076] 다음으로, 표(600a)는 ID[4]에서 프로세싱 유닛(P0)이 이번에는 데이터(DATA[3])를 쓰면서, 다시 캐시 라인에 대한 쓰기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 다시 P0에 의해 "수정"되고 P1에 대하여 "무효"라고 표기하는 것을 보여준다. 표(600c)는 프로세싱 유닛(P0)만이 캐시 라인을 소비하였거나(유닛 비트 603), P0가 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비하였음(플래그 비트 605)을 나타낸다. 표(600d)는, 캐시 라인이 쓰기/수정되었기 때문에, CCP를 사용하여, 프로세서(102)가 동작의 기록이 로깅될 필요가 있다고 결정하는 것을 도시한다. 도시된 바와 같이, 프로세서(102)는 프로세싱 유닛(P0); 캐시 라인 어드레스(@); 명령어 카운트(IC[4]); 캐시 라인이 읽기(공유) 상태로부터 쓰기(수정) 상태로 된 것; P0 및 P1이 이전 캐시 라인에 대한 이전 액세스 권한을 갖고 있었지만 이제 P0만이 액세스 권한을 가진다는 것을 표기할 수 있다.

[0077] 다음에, 표(600a)는 ID[5]에서 프로세싱 유닛(P2)이 데이터(DATA[3])를 읽어서 캐시 라인으로부터 읽기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0 및 P2에 의해 "공유"되었다고 표기하는 것을 보여준다. 표(600c)는 프로세싱 유닛(P0 및 P2)이 캐시 라인을 소비하였거나(유닛 비트 603), P2가 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비하였음을(플래그 비트 605) 도시한다. 인덱스 비트(604)가 여전히 P2 대신 P0를 참조하는 것이 또한 정확할 것이라는 것에 주목하여야 한다. 표(600d)는, 캐시 라인이 쓰기(수정) 상태로부터 읽기(공유) 상태로 되었기 때문에, CCP를 사용하여, 프로세서(102)가 동작의 기록이 로깅될 필요가 있다고 결정하는 것을 도시한다. 도시된 바와 같이, 프로세서(102)는 프로세싱 유닛(P2); 캐시 라인 어드레스(@); 명령어 카운트(IC[5]); 캐시 라인이 쓰기(수정) 상태로부터 읽기(공유) 상태로 된 것; P0가 이전 캐시 라인에 대한 이전 액세스 권한을 갖고 있었지만 이제 P0 및 P2가 액세스 권한을 가진다는 것을 표기할 수 있다.

[0078] 다음에, 표(600a)는 ID[6]에서 프로세싱 유닛(P1)이 또한 데이터(DATA[3])를 읽어서 캐시 라인으로부터의 읽기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0, P1 및 P2에 의해 "공유"되었음을 보여준다. 표(600c)는 프로세싱 유닛(P0, P1 및 P2)이 캐시 라인을 소비하였거나(유닛 비트 603), P1이 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비하였음을(플래그 비트 605) 보여준다. 인덱스 비트(604)가 여전히 P1 대신 P0 또는 P2를 참조하는 것이 또한 옳을 것이라는 것에 주목하여야 한다. 표(600d)는 CCP를 사용하여 프로세서(102)가 동작의 기록이 로깅되어야 한다고 결정하는 것을 도시한다. 도시된 바와 같이, 프로세서(102)는 프로세싱 유닛(P1); 캐시 라인 어드레스(@); 명령어 카운트(IC[6]); 캐시 라인이 읽기(공유) 상태에서 읽기(공유) 상태로 된 것; P0 및 P2가 이전 캐시 라인에 대한 이전 액세스 권한을 갖고 있었지만 이제 P0, P1, P2가 액세스 권한을 가진다는 것을 표기할 수 있다.

[0079] 다음에, 표(600a)는 ID[7]에서 프로세싱 유닛(P3)이 또한 데이터(DATA[3])를 읽어서 캐시 라인으로부터의 읽기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0, P1, P2 및 P3에 의해 "공유"되었다고 표기하는 것을 보여준다. 표(600c)는 유닛 비트(603), 인덱스 비트(604) 및 플래그 비트(605) 중 어느 것도 업데이트되지 않았음을 나타낸다. 이는 P3에 대하여 로깅이 디스에이블되어 있기 때문이고, 트레이싱의 목적 상, 읽기를 수행함으로써 캐시 라인을 "소비"하지 않았기 때문이다. 표(600d)는 데이터가 로깅되지 않았음을 보여준다. 이것은 행위(303)에서 프로세서(102)가 P3에 대하여 로깅이 인에이블되지 않았다고 결정할 것이기 때문이다.

[0080] 다음으로, 표(600a)는 ID[8]에서 프로세싱 유닛(P3)이 데이터(DATA[4])를 쓰면서, 캐시 라인의 쓰기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 P0, P1 및 P2에 대하여 이제 "무효"이고 P3에 의해 "수정"된다고 표기하는 것을 보여준다. 표(600c)는 유닛 비트(603), 인덱스 비트(604) 및 플래그 비트(605)가 모두 캐시 라인이 어느 프로세싱 유닛에 의해서도 소비되지 않는다는 것을 반영한다는 것을 보여준다. 이는 P3에 대하여 로깅이 디스에이블되어 있기 때문이고, 따라서, 트레이싱의 목적 상, 쓰기를 수행할 때 캐시 라인을 "소비"하지 않았기 때문이며; 또한, 쓰기가 다른 프로세싱 유닛에 대한 캐시 라인의 값을 무효화했다. 표(600d)는 로깅된 데이터가 없음을 보여준다. 다시, 이것은 행위(303)에서 프로세서(102)가 P3에 대하여 로깅이 인에이블되지 않았다고 결정할 것이기 때문이다.

[0081] 다음에, 표(600a)는 ID[9]에서 프로세싱 유닛(P0)이 데이터(DATA[5])를 쓰면서, 캐시 라인에 쓰기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0에 의해 "수정"되고, P3에 대하여 "무효"라고 표기하는 것을 나타낸다. 표(600c)는 프로세싱 유닛이 캐시 라인을 소비하지 않았음을 보여준다. 이는, 표(600d)에 반영된 것처럼, 이 동작과 관련하여 로그 항목이 만들어지지 않았기 때문이다. 쓰여진 데이터는 P0의 스레드의 명령어의 정상적인 실행을 통해 재현될 것이므로 로그 항목이 전혀 만들어질 필요가 없다. 그러나, 이 상황에서 트레이스에 항목이 선택적으로 쓰여질 수 있어서(즉, 로깅이 인에이블된 프로세싱 유닛에 의해 로깅되지 않은 캐시 라인에 쓸 수 있어서), 트레이스의 소비자(consumer)에게 추가 데이터를 제공할 수 있다. 이 상황에서 로그 항목은 캐시 라인 값의 읽기와 DATA[5]의 쓰기로서 취급될 수 있다.

[0082] 다음에, 표(600a)는 ID[10]에서 프로세싱 유닛(P2)이 데이터(DATA[5])를 읽어서 캐시 라인으로부터 읽기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0 및 P2에 의해 "공유"되었다고 표기하는 것을 도시한다. 표(600c)는 프로세싱 유닛(P2)이 캐시 라인을 소비하였거나(유닛 비트 603), P2가 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비하였음을(플래그 비트 605) 보여주고 있다. 표(600d)는, 값 캐시 라인이 이전에 로깅되지 않았기 때문에(즉, ID[9]에 로깅되지 않았기 때문에), CCP를 사용하여, 프로세서(102)가 동작의 기록이 로깅될 필요가 있다고 결정하는 것을 도시한다. 도시된 바와 같이, 프로세서(102)는 프로세싱 유닛(P2); 캐시 라인 어드레스(@); 명령어 카운트(IC[10]); 데이터(DATAA[5])

를 캐시 라인에 갖고 온 것; P2가 캐시 라인에 대한 액세스 권한을 가진다는 것을 표기할 수 있다. 또한, 특정 CCP 및 어카운팅 비트가 무슨 정보를 제공하느냐에 따라, P0가 또한 캐시 라인에 대한 액세스 권한을 갖는다고 로깅하는 것이 가능할 수 있다.

[0083] 다음에, 표(600a)는 ID[11]에서 프로세싱 유닛(P1)이 또한 데이터(DATA[5])를 읽어서, 캐시 라인으로부터 읽기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0, P1 및 P2에 의해 "공유"되었다고 표기하는 것을 도시한다. 표(600c)는 프로세싱 유닛(P1 및 P2)이 캐시 라인을 소비하였거나(유닛 비트 603), P1이 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비하였음(플래그 비트 605)을 나타낸다. 인덱스 비트(604)가 여전히 P1 대신 P2를 참조하는 것이 또한 옳을 것이라는 것에 주목하여야 한다. 표(600d)는 CCP를 사용하여 프로세서(102)가 동작의 기록이 로깅되어야 한다고 결정하는 것을 도시한다. 도시된 바와 같이, 프로세서(102)는 프로세싱 유닛(P1); 캐시 라인 어드레스(@); 명령어 카운트(IC[11]); 캐시 라인이 읽기(공유) 상태에서 읽기(공유) 상태로 된 것; P2가 이전 캐시 라인에 대한 이전 액세스 권한을 갖고 있었지만 이제 P1 및 P2가 액세스 권한을 가진다는 것을 표기할 수 있다. ID[10]에서 P2에 의해 로깅되었으므로 값(DATA[5])은 로깅될 필요가 없다는 것에 주목하여야 한다.

[0084] 다음에, 표(600a)는 ID[12]에서 프로세싱 유닛(P0)이 또한 데이터(DATA[5])를 읽어서 캐시 라인으로부터 읽기를 수행하는 것을 도시한다. 표(600b)는 프로세서의 CCP가 캐시 라인이 이제 P0, P1 및 P2에 의해 "공유"되었다고 여전히 표기하는 것을 도시한다. 표(600c)는 프로세싱 유닛(P0, P1 및 P2)이 캐시 라인을 소비하였거나(유닛 비트 603), P0가 캐시 라인을 소비하였거나(인덱스 비트 604), 어떤 프로세싱 유닛이 캐시 라인을 소비하였음을(플래그 비트 605) 도시한다. 인덱스 비트(604)가 여전히 P0 대신에 P1 또는 P2를 참조하는 것이 또한 옳을 것이라는 것에 주목하여야 한다. 표(600d)는 CCP를 사용하여 프로세서(102)가 동작의 기록이 로깅되어야 한다고 결정할 수 있음을 보여준다. 이 경우에, 프로세서(102)는 프로세싱 유닛(P0); 캐시 라인 어드레스(@); 명령어 카운트(IC[12]); 캐시 라인이 읽기(공유) 상태에서 읽기(공유) 상태로 된 것; P1 및 P2가 이전 캐시 라인에 대한 이전 액세스 권한을 갖고 있었지만 이제 P0, P1 및 P2가 액세스 권한을 가진다는 것을 표기할 수 있다. P2로부터 이용 가능하기 때문에, 값(DATA[5])이 로깅되지 않는다.

[0085] 대안적으로, P0이 이미 캐시 라인의 값을 갖기 때문에(즉, ID[9]에서 그 값을 썼기 때문에), 프로세서(102)가 ID[12]에서만 P0을 참조하는 것이 가능할 수 있다. 트레이싱되고 있는 P0를 참조하는 정보 없이 값(즉, DATA[5])을 복구하기 위한 리플레이에서 휴리스틱스가 사용될 수 있기 때문에 ID[12]에서의 임의의 로깅을 하지 않는 것이 심지어 가능할 수 있다. 그러나, 이러한 기법은 계산 비용이 많이 들고 리플레이가 "이탈된(derailed)" 시기를 검출하는 시스템의 능력을 감소시킬 수 있다. 휴리스틱의 예는 프로세싱 유닛에 걸친 메모리 액세스가 일반적으로 (CCP 데이터를 기반으로) 강하게 순서화(order)되므로, 리플레이가 주어진 메모리 장소에 대하여 이들 유닛에 걸친 마지막 값을 사용할 수 있다는 것을 인식하는 것이다.

[0086] 다음에, 표(600a)는 ID[13]에서 캐시 라인이 제거되는 것을 도시한다. 결과적으로, 표(600b)는 CCP 항목이 비어 있음을 나타내고, 표(600c)는 어카운팅 비트가 어느 프로세싱 유닛도 캐시 라인을 소비하지 않았다고 반영하는 것을 나타내고, 표(600d)는 로깅된 데이터가 없음을 나타낸다.

[0087] 완전성을 위하여, 로그 데이터(606)는 모든 종료 액세스 상태(즉, 어느 프로세싱 유닛이 이제 캐시 라인에 대한 액세스 권한을 가지는지)를 나열하지만, 이 정보는 잠재적으로 암시적이고 트레이스 파일 크기는 그것을 생략함으로써 감소될 수 있음에 주목한다. 예를 들어, 쓰기->읽기의 천이에서, 읽기 후 액세스 권한을 갖는 프로세싱 유닛의 리스트는 항상 이전 액세스 권한을 가진 프로세싱 유닛과 읽기를 수행한 프로세싱 유닛이다. 읽기->쓰기의 천이 또는 쓰기->쓰기의 천이에서, 쓰기 후 쓰기 액세스 권한이 있는 프로세싱 유닛의 리스트는 항상 쓰기를 수행한 프로세싱 유닛이다. 읽기->읽기의 천이에서, 읽기 후 액세스 권한을 갖는 프로세싱 유닛의 리스트는 항상 천이 전에 액세스 권한을 갖는 프로세싱 유닛과 읽기를 수행한 프로세싱 유닛이다.

[0088] 일반적으로, 완전히 결정론적인 트레이스 파일을 생성하기 위하여, CCP는 프로세싱 유닛에 걸친(예를 들어, P0에서 P1로) 모든 천이(즉, 쓰기->읽기, 쓰기->쓰기, 읽기->쓰기 및 읽기->읽기)가 로깅되도록 지시할 것이다. 그러나 동일한 프로세싱 유닛 내의 천이(예를 들어, P0에서 P0으로)는 로깅될 필요가 없다. 이들은 해당 프로세싱 유닛에서 실행된 스레드의 정상적인 실행을 통해 재현될 것이므로 로깅될 필요가 없다.

[0089] 상기 예에 로깅된 것과 같은 데이터를 사용하고, 기록이 행해진 프로세서(102)에 의해 사용되는 CCP에 대한 추가 지식을 이용하여, 각 스레드에서 발생된 동작들의 전체 서열화가 재구성될 수 있고, 상이한 프로세싱 유닛 사이의 동작들의 적어도 부분적인 서열화가 재구성될 수 있음을 이해할 것이다. 따라서, 상기 동작들 각각은, 트레이스 파일(104d)에 모두 명시적으로 기록되지 않았더라도, 인덱싱 프로세스 및/또는 트레이스 파일의 리플

레이를 통해 재구성될 수 있다.

[0090] 일부 실시예에서, 트레이서(104a)는 프로세싱 유닛에 걸친 동작들의 서열화의 로깅을 향상시키기 위하여 트레이스 파일(104d)에 추가 데이터 패킷들을 기록할 수 있다. 예를 들어, 트레이서(104a)는 스레드에 걸친 단조 증가 숫자(MIN)(또는 다른 카운터/타이머)를 갖는 이벤트의 전체 서열화를 제공하기 위하여 MIN(또는 다른 카운터/타이머)과 같은 서열화 정보를 일부 이벤트와 함께 기록할 수 있다. 이들 MIN을 사용하여, 스레드에 걸쳐서 "서열화가능하도록(orderable)" 정의된 이벤트에 대응하는 데이터 패킷을 식별할 수 있다. 이러한 이벤트는 스레드가 공유 메모리를 통해 상호 작용하는 방법과 메모리에서 그들의 데이터의 공유 사용을 정의하는 "트레이스 메모리 모델"을 기반으로 정의될 수 있다. 다른 예로서, 트레이서(104a)는 정의된 결정론적 알고리즘 및 정의된 레지스터 세트(예를 들어, 프로그램 카운터, 스택, 범용 레지스터 등)에 기초하여 (주기적으로 또는 무작위로) 프로세서 상태의 해시를 기록할 수 있다. 다른 예로서, 트레이서(104a)는 (주기적으로 또는 무작위로) 캐시 라인 데이터를 강제적으로 로깅할 수 있다. 또 다른 예로서, 트레이서(104a)는 암시적으로 운반하는 데이터의 전부 또는 일부(예를 들어, 몇 비트)의 해시를, 로깅되는 트레이스 "천이" 패킷에 포함할 수 있다. 따라서, 이 암시적 데이터가 리플레이에서 재구성될 때, 암시적 데이터의 적절한 부분이 해시되고 이들 천이 패킷에 매칭되어 그 서열화를 식별하는 데 도움이 될 수 있다. 예를 들어, 캐시 라인이 공유 상태에 있다면 CCP가 캐시 라인과 연관된 프로세서 인덱스를 추적할 수 없는 경우에 이것이 유용할 수 있다.

[0091] 트레이서(104a)가 서열화를 향상시키기 위하여 트레이스 파일(104d)에 추가 데이터 패킷을 기록할 때, 프로세싱 유닛에 걸친 일부 천이의 기록을 생략하는 것이 가능할 수 있다. 예를 들어, 스레드에 걸쳐 일부 읽기->읽기 천이 기록을 생략할 수 있다. 일부 읽기의 서열화가 트레이스 및 CCP에 기초하여 완전히 재구성되지 못할 수 있기 때문에 일부 상황에서는 "약화된(weakened)" 비결정론적 트레이스가 발생할 수 있지만, 추가 서열화 정보(예를 들어, MIN, 프로세서 상태의 해시, 추가 캐시 라인 데이터)는, 트레이스의 리플레이를 "이탈"하지 않는 읽기의 유효한 서열화를 찾기 위한, 리플레이 동안의 서치 공간을 줄이는 데 도움이 될 수 있다. 스레드에 걸친 통한 읽기->읽기 천이의 일부를 생략하는 것의 이점은, 트레이싱을 용이하게 하기 위한, 프로세서(101)에 대한 잠재적으로 단순화된 수정 및 트레이스 크기를 포함한다.

[0092] 도 7a는 프로세서가 어떻게 추적되는지에 따라 일부 읽기->읽기 천이가 트레이스로부터 생략될 수 있는 예를 도시한다. 도 6a와 유사하게, 도 7a는 글로벌 ID(701)를 갖는 표(700a) 및 3 개의 프로세싱 유닛(P0 내지 P2)에 대응하는 3 개의 열(702a 내지 702c)을 포함한다. 일부 읽기->읽기 천이를 생략하는 것은 두 가지 관찰 위에서 구축된다. 첫째, 쓰기가 서열화될 필요가 있다; 그러나, 두 개의 연속 쓰기 사이의 모든 읽기(예를 들어, ID[3] 내지 ID[7]에서의 읽기)는 동일한 값을 읽을 것이므로, 해당 읽기 사이의 순서는 관련이 없다(따라서 그러한 읽기->읽기 천이를 생략하는 트레이스는 결정론적일 수 있다). 둘째, 리플레이 때 읽기가 쓰기를 "교차(cross)"하게 하는 것(즉, 동일한 캐시 라인에의 읽기 및 쓰기가 잘못된 순서로 리플레이되는 것)은 정확한 데이터가 리플레이에 사용되고 있지 않음을 의미한다; 그러나 이러한 실수를 피하기 위한 데이터(예를 들어, MIN 등)를 가지는 것은 유효한 서열화를 식별하는 데 도움이 될 것이다.

[0093] 표(700a)에 도시된 예에서, 프로세싱 유닛(P2)은 공유 데이터에 대한 읽기만을 수행하고, 그 공유된 읽기는 다른 읽기로부터 "도용(steal)"만 한다(예를 들어, ID[9]가 캐시 라인을 공유된 채 두는 것으로 가정함). 임의의 읽기->읽기 천이(즉, ID[4] 내지 ID[7] 및 ID[10])에 대하여 로그 항목이 만들어지지 않으면, P2의 읽기를 올바르게 배치하기 위한 정보가 트레이스에 없을 것이다. 쓰기에 기초하여 P2는 DATA[1] 값을 절대 읽지 않았고(즉, ID[2]에서의 쓰기가 P2로부터 도용하지 않았으므로), P2의 읽기->읽기 천이(즉, ID[4], ID[7], 및 ID[10])에 대한 로그 항목이 없다는 것으로 결론을 내릴 수 있고, P2에 대하여 결론을 내릴 수 있는 모든 것은 ID[2]와 ID[8] 사이에 P2에 의한 적어도 하나의 읽기가 있었다는 것이다. 그러나 만일 ID[4] 및 ID[10]에 대한 로그 항목이 있었다면, 로깅될 필요가 없을 수 있는 나머지 읽기(즉, 도 7B에 표시된 바와 같이 ID[5] 내지 ID[7])의 위치를 찾을 수 있다. 이들 읽기 각각은 마지막으로 로깅된 읽기(즉, ID[4]에서의 읽기)와 동일한 쓰기 간(inter-write) 섹션에 속한다. 따라서 이러한 읽기는 쓰기가 어디서부터 도용하는가에 기초하여 위치를 찾을 수 있다(그리고 읽기로부터 도용하는 동작이 없으면 그 후에는 다음에 로깅된 패킷까지 쓰기가 없다).

[0094] 표(700a)의 관점에서, 도 7b는 로깅 데이터를 나타내는 표(700b)를 도시하고, "유닛 비트"가 사용된다면 기록될 수 있는 도 7a에서 강조된 읽기->읽기 천이를 생략한다. 도 7c는 "인덱스 비트"가 사용되고 읽기 시에 인덱스가 업데이트된다면 기록될 수 있는 로깅 데이터를 나타내는 표(700c)를 도시한다.

[0095] 위에서 간략히 언급한 바와 같이, 일부 캐시는 포괄적 및 배타적 레이어(즉, 비-완전-포괄(non-fully-inclusive) 캐시)를 모두 포함한다. 본 명세서에 기술된 로깅 기법은 이들 캐시뿐만 아니라 순전히 포괄적 또는

배타적 캐시에도 적용 가능하다. 예로서, 도 8a는 2 개의 프로세서(801a/801b)(예를 들어, 대응하는 소켓 내의 2 개의 프로세서)를 포함하는 컴퓨팅 환경(800a)을 도시한다. 각 프로세서(801)는 4 개의 프로세싱 유닛(802a/802b)(예를 들어, 물리적 또는 논리적 프로세싱 유닛)을 포함한다. 각 프로세서(801)는 또한 L1 레이어(803a/803b), L2 레이어(804a/804b) 및 L3 레이어(805a/805b)를 포함하는 3-레이어 캐시를 포함한다. 도시된 바와 같이, 각 캐시는 프로세싱 유닛(802) 중 하나에 각각 대응하는 4 개의 L1 캐시(803)를 포함한다. 또한, 각 캐시는 프로세싱 유닛(802) 중 2 개에 각각 대응하는 2 개의 L2 캐시(804)를 포함한다. 또한 각 캐시는 프로세서(801) 내의 모든 프로세싱 유닛(802)에 대한 하나의 L3 캐시(805)를 포함한다. 프로세싱 유닛 및 일부 캐시는 개별적으로 식별되며, 예를 들어 프로세싱 유닛(802a) 프로세서(801a)는 A0 내지 A3으로 식별되고, L2 캐시는 A4 및 A5로 식별되며, L3 캐시는 A6으로 식별된다. 프로세서(801b)의 대응하는 컴포넌트에 대하여 유사한 식별자가 사용된다. 프로세싱 유닛(A0, A1, A2, B0 및 B1)과 연관된 별표(*)는 이러한 프로세싱 유닛에 대하여 로깅이 인에이블되었음을 표시한다.

[0096] 컴퓨팅 환경(800a)에서, 캐시는 포괄적 및 배타적 거동의 혼합을 나타낼 수 있다. 예를 들어, 프로세싱 유닛(A0)만이 캐시 라인을 사용하고 있을 때 프로세서(801a)의 A6 L3 캐시가 해당 캐시 라인을 저장하는 것은 비효율적일 수 있다. 대신, 이 경우 캐시 라인은 A0의 L1 캐시 및 A4 L2 캐시에 저장될 수 있지만 A1의 L1 캐시 또는 A5 L2 캐시 또는 하위(lower) 캐시에는 저장되지 않을 수 있다. 공간을 확보하기 위하여 일부 캐시는 이 상황에서 A6 L3 캐시가 해당 캐시 라인을 제거할 수 있도록 한다. 이러한 상황이 발생할 때, A1은 포괄적인 캐시에서 일반적인 것처럼 A4 L2 캐시로부터 캐시 라인을 얻을 수 있다. 그러나, 캐시 라인이 A6 L3 캐시 또는 A5 L2 캐시에 존재하지 않기 때문에, 일부 캐시 구현은 또한 A0의 L1 캐시에서 A2 또는 A3의 L1 캐시로와 같이 캐시 라인의 측면(lateral) 이동을 허용할 수 있다. 이것은 CCP를 사용하여 트레이싱하는 데 몇 가지 문제가 제시할 수 있다. 아래 예는 그러한 상황에서 CCP를 사용하여 어떻게 트레이싱이 수행될 수 있는지 보여준다.

[0097] 도 8b는 일부 프로세싱 유닛(802)에 의해 수행된 예시적인 읽기 및 쓰기 동작을 도시하는 표(800b)를 도시한다. 표(800b)의 포맷은 표(600a)의 포맷과 유사하다. 컴퓨팅 환경(800a) 및 표(800b)를 고려하여, 각각 상이한 캐시 거동을 사용하는 3 개의 상이한 로깅 예가 이제 주어진다. 이러한 예는 CCP를 사용하여 로깅하기 위한 다음의 원칙의 맥락에서 설명된다:

[0098] (1) 일반적으로, 어드레스(캐시 라인)가 "로깅되지 않음(not logged)"에서 "로깅됨(logged)"으로 될 때(즉, 캐시 라인이 행위(304)에서 로깅에 참여한다는 결정에 기초하여) 데이터를 로깅한다;

[0099] (2) 일반적으로 캐시 라인이 "로깅됨"에서 "로깅되지 않음" 또는 "제거(evicted)"로 될 때 로깅하지 않는다(그러나, 이 데이터가 로깅되면 로그는 여전히 유효할 것임). 그러나 제거를 로깅하는 것은 유효하다. 그렇게 하면 트레이스 크기가 증가하지만, 트레이스 데이터 스트림 간의 서열화를 식별하는 데 도움이 되고 트레이스의 리플레이가 "이탈"된 시기를 식별하는 데 도움이 되며 추가 트레이스 분석을 제공할 수 있는 추가 정보를 제공한다. 트레이스 분석과 관련하여, 제거를 로깅하는 것은 캐시가 어떻게 사용되었는가에 대한 더 많은 정보를 제공할 수 있고, 실행된 코드의 성능 특성을 식별하는 데 사용될 수 있으며, 주어진 캐시 라인이 특정 값을 저장한 시간 창을 정확히 찾아내는 데 도움이 될 수 있다. 제거를 로깅하기 위한 실시예는 도 10a 및 도 10b와 관련하여 나중에 논의된다;

[0100] (3) 캐시 라인이, 새로운 정보를 제공하는 방식으로 코어 또는 캐시 코히어런스 상태에 걸쳐 이동할 때 이동(movement)을 로깅한다;

[0101] (4) 프로세싱 유닛이 쓰기를 수행할 때, 다른 모든 프로세싱 유닛에 대한 캐시 라인을 무효화한다. 캐시 라인이 프로세싱 유닛에 대하여 아직 로깅되지 않았다면, 시스템은 캐시 라인을 로깅하지 않거나 쓰기를 (i) 읽기(즉, 캐시 라인을 로깅하고 로깅 추적을 턴온함)와 함께 (ii) 캐시 라인 내의 메모리가 프로세싱 유닛에 읽기 가능하다고 가정할 경우, 쓰기로서 다룰 수 있다. 프로세서가 로깅을 턴오프하고 쓰기를 로깅하지 않는 것이 합법적이지만 덜 효율적일 수 있으나, 이것은 리플레이 시에 재구성될 필요가 있는 정보를 잃으며, 평균적으로, 나중에 전체 캐시 라인의 데이터를 로깅하는 것보다 참조를 로깅하는 것이 더 저렴하기 때문에 그것은 비효율적인 트레이스 크기일 수 있다;

[0102] (5) 나중에 트레이스를 재구성하는 것을 돕기 위하여 (예를 들어, 상기 원리 2에서와 같이) 오버-로깅(over-log)하는 것이 유효하다. 이것은 트레이스 크기를 증가시키지만, 정확성에는 영향을 미치지 않는다. 예를 들어, 일부 읽기->읽기 천이는 (도 7a 내지 7c와 관련하여 위에서 설명한 바와 같이) 생략될 수 있지만, 쓰기로 시작하거나 끝나는 교차-코어(cross-core) 천이는 명시적 또는 암시적으로 로깅되어야 한다. 다른 예에서, 실시예는 (예를 들어, 추가 서열화 정보 및/또는 해시를 제공하는) 추가 데이터 패킷을 언제든지 트레이스에 추가할 수

있다. 또 다른 예에서, 실시예는 로깅이 쓰기 상태로의 CCP 천이 후 캐시 라인에 처음으로 쓰기가 행해질 때(committed) 로깅할 수 있다(즉, 추론적 실행으로 인해 캐시 라인이 쓰기 상태로 천이하지만 실제로는 거기에 어떠한 쓰기도 행하지 않기 때문이다). 이러한 쓰기를 로깅하면 나중에 코어 별 트레이스 스트림을 쉽게 분리할 수 있다. 또 다른 예에서, 실시예는 간접 점프(indirect jump), 또는 트레이스 데이터 스트림을 분리할 때 서치 공간을 신속하게 감소시키는 데 도움이 되는 다른 정보를 로깅할 수 있다;

[0103] (6) 완전하지 않은(non-full) 로그(즉, 모든 천이가 로깅된 것이 아닌 로그)를 여전히 사용하여 트레이스를 리플레이할 수 있다. 이로 인해 리플레이 시간에 누락된 조각을 계산하기 위한 추가 계산 비용이 발생할 수 있다.

[0104] 도 8c에 도시된 제1 예에서, CCP는 프로세싱 유닛 당 캐시 라인 상태를 추적한다(즉, 각 코어는 자신의 읽기 및 쓰기 상태를 가진다). 이 예에서 캐시는, 로깅 시간에 사용할 수 없는 교차-캐시(cross-cash) 또는 교차-소켓(cross-socket) 이동을 하는 데이터가 있을 수 있다는 점을 제외하면 포괄적 캐시와 매우 유사하게 거동한다. 간결함을 위하여, 이러한 예에서, 프로세싱 유닛(802)은 "코어"로 지칭되고, 프로세서(801a 및 801b)는 프로세서 A 및 B 또는 소켓 A 및 B로 지칭된다. 또한, 로깅될 수 있는 데이터 유형을 나타내기 위해 "ID:Core:From:Transition(즉, from->to)"의 단순화된 로깅 표기법이 사용된다. 이 표기법은 그 때마다 더 자세히 설명된다. 제1 예에서 로깅에는 다음이 포함될 수 있다:

[0105] ID[0]에서, "0:A0:R[DATA]->[1]", 즉 ID[0]에서, 상기 원칙 1에 따라 코어 A0가 DATA[1]을 읽었다고 로깅한다.

[0106] ID[1]에서, "1:B0:R[DATA]->[1]", 즉, ID[1]에서, 상기 원칙 1에 또한 따라 코어 B0이 DATA[1]을 읽었다고 로깅한다. A0이 이미 데이터를 로깅되게 했음을 프로세서 B의 캐시가 인식하지 못하면 프로세서 B가 그것을 자체적으로 로깅한다. 대안적으로 A0가 DATA[1]이 로깅되게 했음을 프로세서 B의 캐시가 인식하면 로그 항목에 "1:B0:R[A0]->R"이 포함될 수 있다.

[0107] ID[2]에서, "2:A1:R[A0]->R", 즉 ID[2]에서, 코어 A1이 읽기->읽기 천이를 했고, A0이 액세스 권한을 가졌다고 로깅한다. 캐시 라인 상태는 프로세서 B와 공유되기 때문에, 항목은 "2:A1:R:[A0,B0]->R"일 수 있고, 즉 ID[2]에서 A1이 읽기->읽기 천이를 했고, A0 및 B0가 액세스 권한을 가졌다고 로깅한다. 소켓을 교차하는 것은 일반적으로 소켓 내에서 로깅하는 것보다 더 비싸므로, 읽기->읽기 천이에 첫 번째 로그 항목이 선호될 수 있다. 그러나 소켓을 교차하는 쓰기로/로부터 로깅할 때 로깅은 또한 소켓을 교차한다.

[0108] ID[3]에서, 일부 실시예는 아무것도 로깅하지 않는다. 대안적으로 A2 코어가 아직 아무것도 로깅하지 않았으며 가장 먼저 로깅하는 것은 쓰기이므로, 이것은 읽기->쓰기로서 로깅될 수 있다. 어느 쪽이든, 쓰기가 발생했기 때문에 다른 코어는 자신의 캐시 라인 상태를 무효화시킨다. (예를 들어, 트레이스 데이터 측면에서) ID[3]에서 읽기->쓰기를 로깅하는 비용은 일반적으로 ID[4]에서 실제 데이터를 로깅하는 것보다 적으므로 여기서 로깅하는 것이 유리할 수 있다. 이 경우 로그 항목에 "3:A2:R[A0,B1,B0]->W"가 포함될 수 있으며, 즉, A2 코어가 읽기->쓰기 천이를 하였고, A0, B1 및 B0 코어가 액세스 권한을 가졌다.

[0109] ID[4]에서 발생하는 것은 ID[3]에서 무엇이 로깅되었는지에 의존한다. ID[3]에서 아무것도 로깅되지 않았다면, 데이터가 로깅된다(즉, "4:A2:R[DATA]->[2]"). 반면에, 패킷이 ID[3]에서 로깅되었다면, 로깅할 것이 없다.

[0110] ID[5]에는 코어를 교차하는 읽기가 있다. 그러나 A2 코어가 여전히 캐시 라인을 수정 상태(또는 동등한 상태)로 둔다면, 캐시 라인은 요청에 서비스한다(메모리로부터 서비스될 수 없다). 그 경우 소켓 B는 이것이 소켓 A에서 나왔음을 알 것이고 데이터를 다시 로깅하는 것을 피할 수 있으며; 그것은 "5:B0:W[A2]->R"로 로깅될 수 있다. 캐시가 메인 메모리에서 데이터를 얻었다면(이것은 소켓 A가 메인 메모리를 업데이트할 수 있었고 해당 라인의 캐시 코히어런스 상태를 공유할 수 있었다면, 그럴 수 있음) 항목은 "5:B0:R[DATA]->2"일 수 있다.

[0111] ID[6]에서 동작은 정상 읽기이다. ID[2]에서의 읽기처럼, 소켓 B는 소켓 A의 데이터에 대하여 알거나 알지 못할 수 있다. 안다면, 로그 항목은 "6:B1:R[B0, A2]->R"을 포함할 수 있고; 그렇지 않으면 "6:B1:R[B0]->R"을 포함할 수 있다.

[0112] ID[7]에서, B0에 대한 캐시 라인이 제거되지 않았다면 로깅할 것이 없다. B0에 대한 캐시 라인이 제거되었다면, 프로세서 B는, 다른 코어에서 오는 것으로 데이터를 로깅하거나 캐시 라인 데이터를 로깅할 것이다. 소켓에서 다른 코어들이 아닌 하나의 코어를 이렇게 제거하는 것은, 완전히 포괄적인 캐시에서 일반적으로 발생하지 않는다. 완전히 포괄적인 캐시에서 소켓 내의 임의의 코어가 그 L1 캐시에 캐시 라인을 가지고 있다면, L3은 캐시 라인을 가지고 있으므로, 캐시 라인은 한 코어에 대하여 제거될 수 없지만, 다른 코어에 대하여서는 그렇지 않다.

[0113] ID[8]에서, A0 코어는 그 이후로 로깅한 것이 없고 로깅할 제1 동작이 쓰기이기 때문에, 이는 ID[3]에서의 동작과 유사하다. 프로세서 A는 이것을 읽기->쓰기로서 로깅할 수 있고; 대안적으로, 그러나 아마도 덜 바람직하게는, 프로세서 A는 아무것도 로깅하지 않을 수 있다. 패킷이 로깅되면 소켓 A가 소켓 B를 볼 수 있는지 여부에 따라 그 내용이 달라질 것이다. 소켓 A가 소켓 B를 볼 수 없다면, 패킷은 "8:A0:R[A2]->W"를 포함할 수 있지만, 볼 수 있다면, 패킷은 "8:A0:R[B0,B1,A2]->W"를 포함할 수 있다.

[0114] ID[9]에서, 패킷이 ID[8]에 로깅되었다면(이미 로깅된 캐시에 대한 쓰기이기 때문에) 로깅할 것이 없지만, 이미 로깅되지 않았다면 다른 코어에 대한 캐시 라인 상태는 전형적으로 무효화된다.

[0115] ID[10]에서, 로깅은 ID[8]에서 무엇이 로깅되었는지에 의존한다. ID[8]에 로깅된 데이터가 없는 경우 여기에서 로깅이 행해져야 하므로, 패킷은 "10:A1:R[DATA]->[4]"를 포함할 수 있다. 패킷이 ID[8]에 로깅되었다면, 이는 정상적인 쓰기->읽기 패킷이다(예를 들어, "10:A1:W[A0]->R").

[0116] ID[11]에서, 읽기->읽기 천이가 로깅된다. 패킷이 ID[8]에서 로깅되었다면, A0는 코어의 소스 목록에 있으며(예를 들어, "11:A2:R[A0,A1]->R"); 그렇지 않으면 A0가 목록에 없다(예를 들어,"11:A2:R[A1]->R").

[0117] ID[12]에서, 소켓 B가 소켓 A를 볼 수 있다면, 이것은 읽기->읽기 패킷이다(예를 들어, "12:B0:R[A0,A1,A2]->R"). 그렇지 않으면 그것은 풀(full) 데이터 로그이다(예를 들어, "12:B0:R[DATA]->[4]").

[0118] ID[13]에서, 데이터는 B0로부터 오고, 보인다면 소켓 A로부터도 온다(예를 들어, "13:B1:R[A0,A1,A2,B0]->R"). 쓰기가 ID[8]에서 로깅되지 않은 경우 목록은 코어 A0를 생략할 수 있다.

[0119] ID[14]에서 패킷이 ID[8]에 이미 로깅된 경우에는 아무것도 로깅될 필요가 없다. 그렇지 않으면 A0는 A1 및 A2로부터 데이터를 얻을 것이고, 보인다면, 잠재적으로 소켓 B로부터도 얻을 수 있다. 이와 같이, 패킷은 "14:A0:R[A1,A2,B0,B1]->R"을 포함할 수 있다.

[0120] 이 예는 소켓을 함께 로깅했지만, 스레드가 개별적으로 로깅될 수 있는 방식과 유사하게, 각각의 소켓을 분리하여 로깅하는 것이 옳을 것이라는 점에 주목한다. 이로 인해 더 큰 트레이스가 발생할 수 있지만 프로세서의 교차-소켓 통신 메카니즘을 변경할 필요가 없다는 장점이 있다.

[0121] 또한, 시간상 임의의 순간에 캐시 라인이 제거될 수 있으며, 이는 데이터가 다른 코어로부터 수집되거나 재로깅될 필요가 있음을 의미할 것이다. 예를 들어, ID[11] 이전에 A0가 캐시 라인을 제거시킨 경우, A2는 A1으로부터 값을 얻을 것이다. A1과 A0가 모두 제거된 경우 프로세서 A는 캐시 라인 값을 A2의 트레이스에 로깅하여야 할 수 있다.

[0122] 마지막으로, 일부 프로세서는 데이터가 다른 소켓으로부터 온다는 것을 알 수 있지만, 그 소켓의 어느 코어인지 알지 못할 수 있다. 이러한 경우 프로세서는 선행자(precedence)(소스)를 소켓 ID로서 로깅하거나, 데이터 자체를 로깅하거나, 소켓 ID와 데이터의 해시를 로깅할 수 있다(즉, 교차 소켓 액세스를 서열화하는 데 도움이 될 수 있지만, 트레이스에 전체 데이터를 로깅할 필요는 없다).

[0123] 도 8d에 도시된 제2 예에서, CCP는 각각의 코어의 캐시 코히어런시를 개별적으로 추적하는 대신 인덱스를 사용한다. 이 환경에서 인덱스는 소켓을 교차하여 또는 소켓 내에서(intra-socket) 추적될 수 있다. 교차 소켓 대 소켓 내 통신의 성능으로 인해 후자의 경우(소켓 내)가 더 실용적일 수 있다. 소켓 내에서 인덱스가 추적될 때, 트레이스는 데이터가 소켓을 교차하여 이동할 때 무언가를 로깅하여야 할 수 있다. 이것은 다른 소켓으로부터 인덱스를 로깅하는 것(그러나 이것은 결정론적 트레이스를 위해 충분히 고유하여야 할 필요는 없음), 캐시 라인 값의 하나 이상의 부분의 해시를 로깅하는 것, 또는 데이터가 전송되었다는 것을 표시하기 위해 전송 소켓(sending socket)의 트레이스에 패킷을 로깅하는 것을 포함할 수 있다.

[0124] 비-완전-포괄 캐시를 사용할 때 코어 인덱스를 추적할 때, L1 캐시가 L3 캐시에 없는 데이터를 가지고 있을 수 있을 때 문제가 발생한다. 따라서 예를 들어 다음 시퀀스의 이벤트를 가정한다: (i) A0는 자신의 L1 캐시에서 라인을 가져온다(따라서 인덱스 비트는 A0를 참조함); (ii) A1은 자신의 L1 캐시에서 라인을 가져온다(따라서 인덱스 비트는 A1을 참조함); (iii) L3 캐시가 라인을 제거한다; (iv) A1은 L1 캐시로부터 라인을 제거한다; (v) A2는 L1 캐시에서 A0로부터 캐시 라인을 가져온다. 여기서 A2는 A0로부터 캐시 라인을 가져오지만, 인덱스는 A0를 참조하지 않는다. 이로 인해 트레이스로의 매핑을 로깅하는 것이 복잡해진다. 일부 솔루션은, 캐시 라인 데이터의 하나 이상의 부분의 해시와 같은 (전술한 바와 같은) 추가(extra) 정보를 추가하는 것, 범용 레지스터의 해시와 같은 중복 정보를 주기적으로 추가하는 것 등을 포함할 수 있다. 제거를 로깅하는 것 또한 도움이 될 수 있지만, 그것은 트레이스 파일 크기를 크게 증가시키고 로깅을 복잡하게 할 수 있다(예를 들어, L2 또

는 L3 캐시에 있는 L1 캐시 제거를 로깅하지 않고 L2 또는 L3 캐시에 없는 L1 캐시 제거를 로깅하는 것).

[0125] 일부 실시예에서, 데이터가 L3 캐시로부터 자식(child) L2 또는 L1 캐시로 이동할 때, 로그 항목은 인덱스가 변경되는 경우에만 만들어진다. 예를 들어, A0가 자신의 L1 캐시에 라인을 가지고 있고(따라서 인덱스 비트가 A0를 참조함), 그 후 A1이 자신의 L1 캐시에서 라인을 가져오고(A1에서 인덱스), 그 후 둘 다는 캐시 라인을 제거하지만 공통 L2(또는 L3)는 여전히 그 캐시 라인을 갖고 있다고 가정한다. L2 캐시가 A1에 서비스하면 로깅할 것이 없다. L2 캐시가 A0에 서비스하면, A0에 이미 데이터가 있는 것으로 알려진 경우 로그 항목이 만들어질 필요가 없지만; A0가 이미 데이터를 가지고 있는지 알 수 없는 경우(또는 결정할 수 없는 경우) 프로세서는 읽기->읽기를 로깅하여야 할 수 있다.

[0126] 표(800d)는 소켓이 독립적으로 로깅하고, 인덱스에 의해 추적이 수행되고, 추가적인 숨겨진 제거가 없으며, CCP에 영향을 미치고 로깅이 턴온될 때 발생하는 모든 쓰기가 로깅된다고 가정할 때, 표(800b)의 동작의 로그를 제시한다(예를 들어, 동일한 코어에 의한 연속 쓰기가 있고 쓰기 사이에 다른 코어 또는 다른 외부 엔티티에 의한 액세스가 없는 경우 하나의 쓰기가 로깅되어야 한다). 제2 예에서 로깅에는 다음이 포함될 수 있다:

[0127] ID[0]의 경우, "0:A0:R[DATA]->[1]".

[0128] ID[1]의 경우, "1:B0:R[DATA]->[1]" - 즉, 각각의 소켓이 개별적으로 로깅된다는 것을 상기한다.

[0129] ID[2]의 경우, "2:A1:R[A0]->R".

[0130] ID[3]의 경우, "3:A2:R[A1]->W".

[0131] ID[4]의 경우에는 아무것도 없다.

[0132] ID[5]의 경우, "5:B0:R[DATA]->[2]"이다. 이것은 ID[3]에서의 쓰기가 모든 소켓에 걸쳐 라인을 무효화하였고(위에서 언급한 바와 같이) 소켓이 독립적으로 트레이싱되고 있기 때문이다.

[0133] ID[6]의 경우, "6:B1:R[B0]->R".

[0134] ID[7]의 경우, B0에 대한 캐시 라인이 제거되지 않았다면 로깅할 것이 없다.

[0135] ID[8]의 경우:"8:A0:R[A2]->W", (이 코어가 이전에 데이터를 로깅하지 않았음에도 불구하고) 로깅 비트가 온이기 때문이다. 이 항목은 인덱스를 사용하면 어떻게 소켓의 마지막 소유자에 대한 지식만 있는지 보여준다.

[0136] ID[9]의 경우, 로깅할 것이 없다.

[0137] ID[10]의 경우, "10:A1:W[A0]->R".

[0138] ID[11]의 경우, "11:A2:R[A1]->R".

[0139] ID[12]의 경우, "12:B0:R[데이터]->[4]". 이는 캐시 라인이 ID[8]에서 모든 소켓에 걸쳐 무효화되었기 때문이다.

[0140] ID[13]의 경우, "13:B1:R[B0]->R".

[0141] ID[14]의 경우, "14:A0:R[A2]->R". ID[11]에서 인덱스는 A2로 업데이트되었다는 것을 주목하여야 한다. 또한 프로세서 당 상태(유닛 비트)가 정보를 전달할 수 있기 전에 인덱스가 해당 정보를 전달하지 않기 때문에 이 코어에 이미 데이터(즉, ID[9])가 있음이 알려지지 않을 것이라는 것을 또한 주목하여야 한다.

[0142] 제3 예에서, 환경(800a)의 캐시는 어느 코어가 캐시 라인에 대한 마지막 공유(읽기) 액세스를 갖는지를 추적할 수 없다. 따라서 이 예에서는 마지막으로 읽은 자의 인덱스를 추적할 수 없는데, 그렇게 하기 위한 비트가 없기 때문이다. 여기서, CCP는 (어떤 코어로도 맵핑되지 않는) 하나의 인덱스 값을 사용하여 공유 라인을 시그널링하고, 다른 인덱스 값을 사용하여 무효 라인을 시그널링하고, (예를 들어, MSI 프로토콜을 사용하여) "수정" 상태에 대하여 프로세서 인덱스를 사용한다. 이 제3 예에서, 로깅에는 코어의 인덱스 대신 패킷에서의 캐시의 인덱스를 로깅하는 것이 포함될 수 있다. 부모(parent)로부터 자녀(child)로의 이동은 로깅될 필요가 없지만 추가 데이터로서 로깅될 수 있다. 부모로부터 자녀로의 이동이 로깅되지 않으면, 로그를 해석하기 위하여 부모로부터 자녀로의 캐시 체계(hierarchy)가 제공될 필요가 있을 수 있다.

[0143] 위에서 언급한 바와 같이, 일부 환경에서, 캐시의 각 캐시 라인은 단일 플래그 비트를 포함할 수 있었지만, 프로세서의 CCP는 캐시 라인의 코히어런스 상태를 소유하는 프로세싱 유닛에 대한 인덱스를 참조하여 각 캐시 라인에 대한 코히어런스 상태를 추적할 수 있다. 언급한 바와 같이, 이는 완전히 결정론적인 트레이스를 생성하지

만, 프로세싱 유닛 당 정보를 갖는 경우(예를 들어, 캐시 라인 당 플래그 비트와 결합하여, 프로세싱 유닛 별로 추적하는 CCP)보다 더 큰 트레이스를 초래할 수 있다. 도 9a와 도 9b는 이 두 상황에서 로깅이 어떻게 상이할 수 있는지 보여준다(즉, CCP 유닛 정보 + 캐시 라인 플래그 비트 대 CCP 인덱스 + 캐시 라인 플래그 비트). 도 9a는 2 개의 프로세싱 유닛(P0 및 P1)에 의한 읽기 및 쓰기를 나타내는 표(900a)를 도시하고, 도 9b는 이들 2 개의 환경에서 언제 로그 항목이 만들어질 수 있는지 비교하는 표(900b)를 도시한다. 이들 예에서, 플래그 비트가 시작되어 클리어되고, 유닛/인덱스 비트는 어느 프로세싱 유닛도 캐시 라인에 대한 액세스 권한을 가지고 있지 않음을 표시한다고 가정한다.

[0144] 초기에, CCP가 유닛 정보를 추적하고 캐시 라인이 플래그 비트를 사용하면, 로깅은 다음과 같이 진행될 수 있다. 표(900b)에 도시된 바와 같이, ID[0]에서, 로깅되지 않은 캐시 라인에 대한 쓰기이기 때문에, 아무 것도 로깅될 필요가 없다(대안적으로, 쓰기 전의 값이 로깅될 수 있고, 플래그 비트가 플립 온(flip on)될 수 있다). 이 시점에서 CCP는 P0 또는 P1 어느 것도 캐시 라인에 액세스하지 않음을 v표기할 수 있다. ID[1]에서 캐시 라인 데이터가 P1에 대하여 로깅될 수 있다. 플래그 비트가 턴온될 수 있으며 CCP는 P1가 캐시 라인에 액세스할 수 있음을 표기할 수 있다. ID[2]에서 P0이 P1으로부터 캐시 라인을 가져와서 읽기->읽기 패킷이 로깅될 수 있다(플래그 비트가 온이었기 때문에 이것이 로깅되며 CCP는 P0가 액세스 권한이 없었다고 결정하는 데 사용된다). 플래그 비트가 이미 온이고, CCP는 P0이 이제 또한 캐시 라인의 상태에 액세스 권한을 가지고 있다고 표기한다. ID[3]에서 아무 것도 로깅할 필요가 없다(캐시 라인은 이미 이 코어에 대한 로그에 있다). 이는 플래그 비트가 온이기 때문에 결정되었고 CCP는 P1가 이미 캐시 라인에 액세스하였음을 표시한다. ID[4]에서 P0에 대하여 읽기->쓰기 패킷이 로깅될 수 있다. 이것은 플래그 비트가 온이기 때문이고, P0이 이미 캐시 라인에 액세스했기 때문이다. 이것이 쓰기였으므로 CCP는 다른 모든 프로세서에 대한 캐시 라인을 무효화할 수 있다(즉, P0에는 액세스 권한이 있고 P1에는 없다). ID[5]에서 쓰기->읽기 패킷이 P1에 대하여 로깅될 수 있다. 이는 플래그 비트가 온이지만, P1은 (CCP에 의해 표시된 바와 같이) 데이터를 트레이스에 갖고 있지 않기 때문이다. ID[4] 및 ID[5]에 있는 두 개의 참조 패킷은 ID[4]에 아무것도 로깅하지 않고 이후 ID[5]에서 데이터를 로깅하여야 하는 것보다 더 작다는 것을 주목하여야 한다. CCP는 P1가 P0 외에도 이제 캐시 라인에 액세스할 수 있다고 표기한다.

[0145] 이제 CCP가 인덱스 정보만 추적하고 캐시 라인이 플래그 비트를 사용하는 경우 로깅은 다음과 같이 진행될 수 있다. 표(900b)에 도시된 바와 같이, ID[0]에서, 플래그 비트가 오프이고 이것이 쓰기이기 때문에 아무것도 로깅될 필요가 없다. 이전과 같이, 메모리가 P0에 의해 읽기 가능하다면, 이것은 대안적으로 읽기 + 쓰기로서 로깅될 수 있다. ID[1]에서 캐시 라인 데이터가 P1에 대하여 로깅될 수 있다. 플래그 비트가 턴온될 수 있고 CCP는 P1를 가리키도록 인덱스를 업데이트한다. ID[2]에서 P0에 대한 읽기->읽기 패킷이 로깅될 수 있다. 이것은 플래그 비트가 이미 온이고 인덱스가 P1에 있기 때문이다. CCP는 P0에 대한 인덱스를 업데이트할 수 있다. ID[3]에서 P1에 대하여 읽기->읽기 패킷이 로깅될 수 있다. 이 경우는 이제 ID[2]와 구별되지 않는데, 그 이유는 두 경우 모두 인덱스가 다른 프로세서에 있고, 플래그 비트가 온이고, 캐시 라인이 공유 상태에 있기 때문이다. CCP는 P1에 대한 인덱스를 업데이트할 수 있다. ID[4]에서 P0에 대한 읽기->쓰기 패킷이 로깅될 수 있다. 플래그 비트가 온이므로, 패킷은 참조로 로깅할 수 있다. 이것은 CCP의 P0에 대한 인덱스를 업데이트한다. ID[5]에서 P1에 대한 쓰기->읽기 패킷이 로깅될 수 있다. 플래그 비트가 온이고, 패킷은 참조로 로깅된다. 캐시 라인은 공유 상태로 이동하므로 CCP는 P1에 대한 인덱스를 업데이트한다. 표(900b)에 도시된 바와 같이, 인덱스 케이스는 유닛 케이스보다 더 큰 트레이스 파일을 생성하지만 여전히 완전히 결정론적인 트레이스를 생성한다.

[0146] 본 명세서의 일부 실시예는 추후에(예를 들어, 이전 예의 각각에서 ID[4]) 캐시 라인 데이터를 기록하는 것이 아니라, (가능할 때) 다른 프로세싱 유닛에 의해 소유된 데이터를 참조하는 데이터 패킷을 기록하는 것이 트레이스 파일 크기의 관점에서 유리할 수 있음을 나타내었다. 다른 이점은 또한 참조로 기록(recording)으로부터 나올 수도 있다. 예를 들어, 리플레이시 참조되는 일련의 로그 항목이 있는 경우 캐시 라인 데이터에서 외부 개입이 발생하지 않은 것으로 추론할 수 있다. 이는 풀 캐시 라인 데이터가 다시 로깅될 때 캐시 라인이 제거되었거나 무효화되었음을 의미하기 때문이다. 따라서, 로그 항목이 엄격하게 필요하지 않은 상황에서도 참조로 로그 항목을 포함하면, 리플레이때 또는 디버깅에 유용한 정보가 될 수 있는 외부 개입의 부재에 대한 암시적 정보를 제공할 수 있다.

[0147] 일부 구현에서, 트레이스 항목(예를 들어, 상기 "@" 항목)에 기록된 어드레스는 물리적 메모리 어드레스를 포함한다. 이러한 구현에서, 프로세서(102)는 TLB(102f)의 하나 이상의 항목을 트레이스 파일(104d)에 기록할 수 있다. 이것은 상이한 프로세싱 유닛에 대한 트레이스 데이터 스트림의 일부이거나, 하나 이상의 추가 트레이스 데이터 스트림의 일부일 수 있다. 이를 통해 리플레이 소프트웨어가 나중에 이러한 물리적 어드레스를 가상 어드

레스에 매핑할 수 있게 할 것이다.

[0148] 또한, 물리적 어드레스는 때때로(예를 들어, 사용자 모드의 레벨에서 기록할 때) "비밀(secret)" 정보로 간주될 수 있기 때문에, 일부 실시예는 물리적 어드레스 자체보다는 실제 물리적 어드레스의 일부 표현을 기록한다. 이 표현은 물리적 어드레스를 드러내지 않고 그 식별자를 물리적 어드레스에 고유하게 매핑하는 임의의 표현일 수 있다. 하나의 예는 각 물리적 어드레스의 해시일 수 있다. 이들 표현이 사용되고 TLB(102f)의 항목이 트레이스 파일(104d)에 기록될 때, 프로세서(102)는 물리적 어드레스 대 가상 어드레스가 아니라, 이들 표현 및 가상 어드레스 사이의 매핑을 기록한다.

[0149] 언급한 바와 같이, 프로세서(102)는 하나 이상의 버퍼(102e)를 포함할 수 있다. 이들 버퍼는 실제로 그 항목을 트레이스 파일(102f)에 쓰기 전에 트레이스 파일 항목을 위한 임시 저장 장소로서 사용될 수 있다. 따라서, 행위(305)로 인해 데이터가 트레이스에 로깅될 때, 행위(305)는 데이터를 버퍼(102e)에 쓰는 것을 포함할 수 있다. 일부 실시예에서, 프로세서(102)는 트레이스 데이터를 프로세서(102) 및 메모리 버스에 쓰는 것의 영향을 감소시키기 위하여 지연된 로깅 기법을 사용한다. 이들 실시예에서, 프로세서(102)는 트레이스 데이터를 버퍼(102e)에 저장하고, 메모리 버스 상에 이용 가능한 대역폭이 존재하거나 버퍼(102e)가 풀(full)일 때까지 트레이스 파일(102f)로의 쓰기를 지연시킬 수 있다.

[0150] 또한 언급된 바와 같이, 일부 실시예는 캐시 제거를 로깅할 수 있다. 도 10a 및 10b는 연관 캐시의 특성을 활용하는 효율적인 방식으로(즉, 트레이스 파일 크기의 면에서) 어떻게 캐시 제거가 로깅될 수 있는지의 일부 실시예를 도시한다. 초기에, 도 10a는 메모리 어드레스의 상이한 부분의 예(1000) 및 연관 캐시와의 관계를 도시한다. 도시된 바와 같이, 메모리 어드레스는 어드레스의 하위 비트이고 일반적으로 0인 제1 복수의 비트(1001)를 포함한다. 메모리 어드레스는 전형적으로 메모리 어드레스 크기(예를 들어, 32 비트, 64 비트 등)에 정렬되기 때문에 제1 복수의 비트(1001)는 0이다. 따라서, 제1 복수의 비트(1001)의 수는 메모리 어드레스의 크기에 의존한다. 예를 들어, 메모리 어드레스가 32 비트(즉, 2^5 비트)이면, (메모리 어드레스가 32의 배수가 되도록) 제1 복수의 비트(1001)는 5 비트를 포함하고 메모리 어드레스가 64 비트(즉, 2^6)이면, (메모리 어드레스가 64의 배수가 되도록) 제1 복수의 비트(1001)는 6 비트를 포함한다. 메모리 어드레스는 또한 메모리 어드레스의 데이터가 저장되어야 하는 연관 캐시에서 특정 어드레스 그룹을 결정하기 위하여 프로세서(102)에 의해 사용될 수 있는 제2 복수의 비트(1002)를 포함한다. 예를 들어, 도 10a의 예(1000)에서, 제2 복수의 비트(1002)는 3 개의 비트를 포함하며, 이는 8 개의 어드레스 그룹을 갖는 연관 캐시에 대응할 것이다. 따라서 제2 복수의 비트(1002)의 수는 연관 캐시의 특정 기하학적 구조에 의존한다. 메모리 어드레스는 또한 메모리 어드레스의 나머지 상위 비트를 포함하는 제3 복수의 비트(1003)를 포함한다.

[0151] 도 10a의 맥락에서,도 10b는 연관 캐시에서 캐시 미스 및 캐시 제거를 로깅하는 예(1004)를 도시한다. 처음에, 예(1004)는 3 개의 메모리 어드레스 1005(즉, 어드레스 1024), 1006(즉, 어드레스 @2112) 및 1007(즉, 어드레스 @2048)을 도시한다. 도 10b는 또한 각각 4 가지 웨이를 포함하는 8 개의 그룹을 갖는 연관 캐시(1010)를 도시한다. 이러한 그룹과 웨이의 이진 아이덴티티는 괄호 안에 있는 해당 10 진수 표시와 함께 열 1008(그룹) 및 1009(웨이)에 표시된다. 따라서, 예를 들어, 캐시(1010)에서 캐시 라인(0, 0), 즉, 그룹 0, 웨이 0은 그룹 '000'(열 1008) 및 웨이 '00'(열 1009)으로서 이진수로 도시되고; 캐시(1010)에서 캐시 라인(0, 1), 즉, 그룹 0, 웨이 1은 그룹 '000'(열 1008) 및 웨이 '01'(열 1009)으로서 이진수로 도시되고; 캐시(1010)에서 캐시 라인(8, 3), 즉, 그룹 8, 웨이 3이 그룹 '111'(열 1008) 및 웨이 '11'(열 1009)으로서 이진수로 도시될 때까지 계속된다.

[0152] 이제, 어드레스 1005(즉, @ 1024)에 제1 캐시 미스가 있다고 가정하자. 여기서, 제2 복수의 비트(1002)는 '000'이므로, 프로세서(102)는 그것이 캐시(1010)의 그룹 0에 어드레스 1005에 대응하는 데이터를 저장하여야 한다고 결정할 수 있다. 그룹 0의 특정 웨이는 일반적으로 프로세서 특유의 로직에 의해 선택된다. 그러나, 예(1004)의 목적을 위하여, 데이터가 (화살표로 도시된 바와 같이) 웨이 0, 1011a로 저장되었다고 가정하자. 이 캐시 미스와 관련하여, 트레이서(104a)에 의해 기록된 로그 데이터는 데이터가 저장된 메모리 어드레스(즉, @1024) 및 웨이(즉, 웨이 0)를 포함할 수 있다. 트레이스에 메모리 어드레스를 저장하는 데 필요한 비트 수를 줄이기 위하여 여러 가지 압축 기법을 사용할 수 있다는 것에 주목하여야 한다. 그룹(즉, 그룹 0)은 메모리 어드레스의 제2 복수의 비트(1002)로부터 획득될 수 있기 때문에 로깅될 필요가 없다.

[0153] 다음에, 어드레스 1006(즉, @2112)에 제2 캐시 미스가 존재한다고 가정하자. 이번에, 제2 복수의 비트(1002)는 '010'이므로 프로세서(102)는 그것이 캐시(1010)의 그룹 2에 어드레스 1006에 대응하는 데이터를 저장하여야 한다고 결정할 수 있다. 다시, 그룹 2의 특정 웨이는 일반적으로 프로세서 특유의 로직에 의해 선택된다. 그러나,

예(1004)의 목적을 위하여, 데이터가 (화살표로 도시된 바와 같이) 웨이 0 1011b에 저장되어 있다고 가정하자. 이 캐시 미스와 관련하여, 트레이서(104a)에 의해 기록된 로그 데이터는 데이터가 저장된 메모리 어드레스(즉, @ 2112) 및 웨이(즉, 웨이 0)를 포함할 수 있다. 또한, 그룹(즉, 그룹 2)은 메모리 어드레스의 제2 복수의 비트(1002)로부터 획득될 수 있기 때문에 로깅될 필요가 없다.

[0154] 이제 어드레스 1007(즉, @ 2048)에 3번째 캐시 미스가 있다고 가정하자. 제2 복수의 비트들(1002)은 다시 '00 0'이므로 프로세서(102)는 그것이 캐시(1010)의 그룹 0에서 어드레스 1007에 대응하는 데이터를 저장하여야 한다고 결정할 수 있다. 특정한 웨이는 프로세서 특유의 로직에 의해 다시 선택되지만, 프로세서는 (화살표 1011c로 표시된 바와 같이) 웨이 0을 선택했다. 이 캐시 미스와 관련하여, 트레이서(104a)에 의해 기록된 로그 데이터는 메모리 어드레스(즉, @2048) 및 데이터가 저장된 웨이(즉, 웨이 0)를 포함할 수 있다. 다시, 그룹(즉, 그룹 0)은 메모리 어드레스의 제2 복수의 비트(1002)로부터 획득될 수 있기 때문에 로깅될 필요가 없다.

[0155] 이 캐시 라인(0,0)은 어드레스 1005에 현재 대응하기 때문에, 어드레스 1007에서의 이러한 제3 캐시 미스로 인해 어드레스 1005가 캐시(1010)로부터 제거된다. 그러나, 실시예는 이 제거를 기록(document)하는 임의의 트레이스 데이터를 기록하지 않을 수 있다. 이는 제거가 이미 트레이스에 있는 데이터로부터 유추될 수 있기 때문이다. 즉, 어드레스 1005의 첫 번째 캐시 미스가 웨이 0으로, 이와 함께 어드레스 1007의 두 번째 캐시 미스가 웨이 0으로 이어진다. 그룹(그룹 0)이 트레이스에 명시적으로 로깅될 필요가 없을 수 있지만, 이러한 어드레스로부터 유추될 수 있다. 따라서, 이 트레이스 데이터를 리플레이하면 제거를 재현할 수 있다.

[0156] 일부 제거는 캐시 미스 이외의 이벤트로부터 발생한다. 예를 들어, CCP는 상이한 캐시 간의 일관성을 유지하기 위하여 제거가 발생하게 할 수 있다. 예를 들어, 어드레스(1006)가 CCP 이벤트로 인해 캐시(1010)의 캐시 라인(2,0)으로부터 제거된다고 가정하자. 여기서, 제거는 제거의 그룹(즉, '010')과 웨이(즉, '00')를 기록함으로써 명시적으로 로깅될 수 있다. 특히, 제거된 어드레스는 어드레스 1006을 캐시 라인(2,0)으로 가져온 두 번째 캐시 미스를 로깅할 때 이미 캡처되었기 때문에 로깅될 필요가 없다. 따라서, 이 예에서, 제거는 단지 5 비트의 로그 데이터(임의의 압축 형태 이전)로 트레이스 파일(104d)에서 완전히 캡처될 수 있다.

[0157] 일부 실시예는 또한 프로세싱 유닛에서 실행되는 스레드가 보안 엔클레이브(secure enclave)와 상호 작용할 때에도 프로세싱 유닛의 활동을 안전하게 트레이싱할 수 있다. 당업자가 이해할 수 있는 바와 같이, 엔클레이브는 민감한 정보(예를 들어, 암호화 키, 자격 증명(credential), 생체 측정 데이터 등)를 잠재적으로 프로세서(102)에서 실행되는 가장 낮은 수준의 소프트웨어로부터 보호할 수 있는 하드웨어 기반 보안 특징이다. 따라서, 사용자 모드 프로세스로부터 민감한 정보를 보호하는 것 외에도, 엔클레이브는 심지어 커널 및/또는 하이퍼바이저로부터 민감한 정보를 보호할 수 있다. 많은 구현에서, 엔클레이브들은 프로세스의 어드레스 공간에 매핑된 메모리의 암호화된 부분으로서 실행 프로세스에 나타난다. 이것은 예를 들어 실행 프로세스 및 엔클레이브를 위하여 상이한 메모리 페이지 테이블을 사용함으로써 구현될 수 있다. 프로세스가 엔클레이브와 상호 작용할 때, 프로세스는 자신의 매핑된 메모리에 읽기/쓰기를 할 수 있고, 엔클레이브는 자신의 매핑된 메모리 및/또는 프로세스의 매핑된 메모리에 읽기/쓰기를 할 수 있다.

[0158] 제1 엔클레이브 인식 트레이싱 실시예는 프로세스가 상호 작용하는 엔클레이브를 트레이싱하지 않으면서 여전히 트레이싱된 프로세스가 완전히 리플레이되게 하는 동안 실행 프로세스를 트레이스한다. 이들 실시예에서, 실행 프로세스에 의해 그 어드레스 공간으로의 메모리 읽기는 본 명세서에서 이미 설명된 하나 이상의 메커니즘을 사용하여 트레이싱/로깅된다. 그러나, 엔클레이브로의 컨텍스트 스위치가 있을 때, 실시예는 트레이싱된 프로세스에 의해 이전에 읽혀지고 그 실행 동안 엔클레이브에 의해 쓰여진 임의의 메모리 장소를 추적할 수 있다. 엔클레이브로의 스위치 후 트레이싱된 프로세스가 다시 실행될 때, 이러한 메모리 장소는 트레이싱된 프로세스에 의해 로깅되지 않은 것으로 취급된다. 이렇게 하면 트레이싱된 프로세스가 이들 메모리 장소로부터 다시 읽을 경우(엔클레이브에 의해 해당 위치에 배치된 데이터를 잠재적으로 읽음) 이러한 읽기는 트레이스에 로깅된다. 효과적으로, 이것은 트레이싱된 프로세스에 보이는 엔클레이브의 실행의 부작용이 엔클레이브의 실행을 트레이싱할 필요 없이 트레이스에서 캡처됨을 의미한다. 이러한 방식으로, 트레이싱된 프로세스는 실제로 엔클레이브의 실행을 리플레이할 필요 없이(또는 심지어는 가능하지 않음) 이러한 부작용을 이용하여 나중에 리플레이될 수 있다. 트레이싱된 프로세스에 의해 이전에 읽혀졌고 어카운팅 비트(예를 들어, 플래그 비트, 유닛 비트, 인덱스 비트), 웨이 로킹, CCP 데이터 사용 등과 같이 실행 중에 엔클레이브에 의해 쓰여진 메모리 장소를 추적하는 데 사용할 수 있는 (이전에 설명된) 몇 가지 메커니즘이 있다.

[0159] 제2 엔클레이브 인식 트레이싱 실시예는 (예를 들어, 자신의 어드레스 공간에 대한 읽기와 같은 액세스에 기초한) 실행 프로세스를 트레이싱하면서, 또한 (예를 들어, 자신의 어드레스 공간에 대한 액세스 및/또는 트레이싱

된 프로세스의 어드레스 공간에 대한 액세스에 기초한) 엔클레이브를 트레이스한다. 이들 실시예는 커널/하이퍼바이저와 엔클레이브 사이에 필요한 신뢰 수준이 있을 때 구현될 수 있다. 이들 실시예에서, 엔클레이브의 실행과 관련된 트레이스 데이터는 별도의 트레이스 데이터 스트림에 로깅될 수 있고/있거나, 리플레이를 수행하는 임의의 엔티티가 엔클레이브의 별도의 트레이스 데이터 스트림 및/또는 엔클레이브의 실행과 관련된 트레이스 데이터를 해독하는 데 사용될 수 있는 암호화 키에 액세스하지 않고 엔클레이브를 리플레이할 수 없도록 암호화될 수 있다.

[0160] 제3 엔클레이브-인식 트레이싱 실시예는 제1 및 제2 실시예를 결합한다. 따라서, 이들 제3 실시예는 엔클레이브 자체의 트레이스(즉, 제2 실시예)와 함께, 프로세스의 엔클레이브 사용(즉, 제1 실시예)의 부작용을 포함하는 실행 프로세스의 트레이스를 기록할 수 있다. 이것은 트레이싱된 프로세스의 실행이 필수 특권 레벨 및/또는 암호화 키가 없는 사용자에 의해 리플레이될 수 있게 하면서, 필수 특권 레벨 및/또는 암호화 키를 가진 사용자는 또한 엔클레이브 자체의 실행을 리플레이할 수 있게 한다.

[0161] 이들 엔클레이브 트레이싱 실시예 각각은 엔클레이브를 넘어서서, 트레이싱된 엔티티가 트레이싱 동안 실행이 보호되어야 하는 다른 엔티티(이하, 보호된 엔티티(protected entity)로 지칭됨)와 상호 작용하는 임의의 상황까지 적용 가능하다. 예를 들어, 커널 모드 프로세스와 상호 작용하는 사용자 모드 프로세스를 트레이싱할 때 이들 실시예 중 임의의 것이 사용될 수 있다. 여기서 커널 모드 프로세스는 엔클레이브와 거의 동일하게 취급될 수 있다. 다른 예에서, 하이퍼바이저와 상호 작용하는 커널 모드 프로세스를 트레이싱할 때 이들 실시예 중 임의의 것이 사용될 수 있다. 여기서, 하이퍼바이저는 엔클레이브와 거의 동일하게 취급될 수 있다.

[0162] (예를 들어, 성능 또는 보안 고려 사항으로 인해) 실용적이지 않거나, (예를 들어, 하드웨어 지원 부족으로 인해) 불가능하거나, 트레이싱된 프로세스에 의해 이전에 읽혀진 어느 메모리 장소가 실행 동안 보호된 엔티티에 의해 쓰여지는지 추적하는 것이 바람직하지 않은 환경이 있을 수 있다. 이것은 전술한 엔클레이브 트레이싱 실시예의 사용을 방지할 수 있다. 그러나 이러한 상황에서 트레이싱을 위한 기법도 있다.

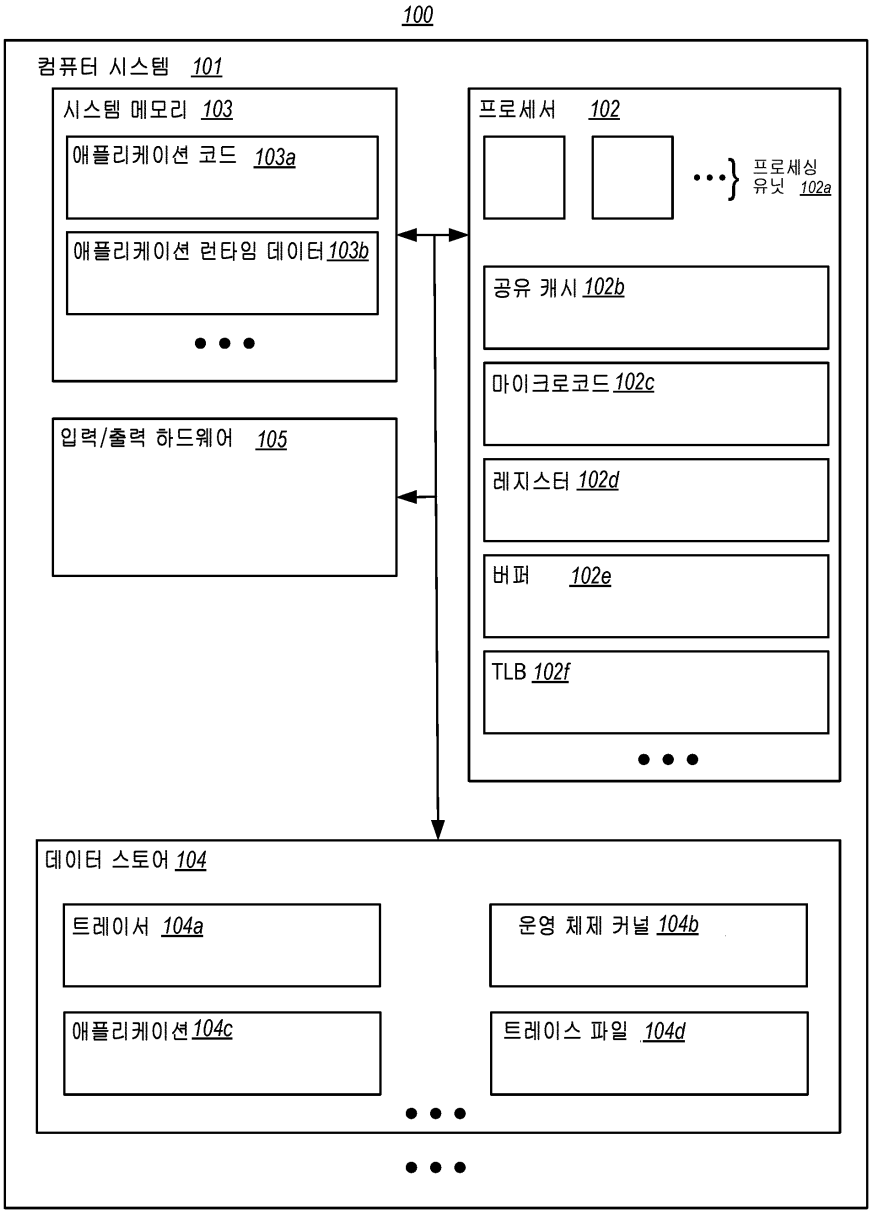
[0163] 제1 기법은 보호된 엔티티로부터의 컨텍스트 스위치 이후에 프로세서 캐시가 무효화된 것으로 취급하는 것이다. 프로세서 캐시를 무효화된 것으로 취급하면, 보호된 엔티티로부터 복귀된 후 트레이싱된 엔티티에 의한 읽기가 캐시 미스를 초래하게 하고 이는 로깅될 수 있다. 이러한 캐시 미스는 보호된 엔티티에 의해 트레이싱된 엔티티의 어드레스 공간에서 수정된 임의의 값과 트레이싱된 엔티티에 의해 추후 읽혀진 임의의 값을 포함할 것이다. 이 기법은 상술한 3 개의 실시예보다 더 많은 트레이스 데이터를 생성할 수 있지만, 트레이싱된 엔티티에 의해 의존된 보호된 엔티티의 실행 효과를 캡처한다. 일부 실시예에서, 이 제1 기법은 또한 보호된 엔티티로부터 트레이싱된 엔티티로 복귀될 때 하나 이상의 키 프레임(예를 들어, 프로세서 레지스터의 스냅 샷을 포함)을 기록할 수 있다. 키 프레임은 (즉, 보호된 엔티티의 실행 동안) 트레이스 데이터의 연속성이 부족하더라도, 보호된 엔티티로부터의 복귀 후에 트레이싱된 엔티티의 리플레이가 시작될 수 있게 한다.

[0164] 제2 기법은 보호된 엔티티에 의해 트레이싱된 엔티티의 어드레스 공간으로 수행되는 쓰기뿐만 아니라, 보호된 엔티티에 의한 트레이싱된 엔티티의 어드레스 공간으로부터의 읽기와 관련된 캐시 미스를 로깅하는 것이다. 이를 통해 트레이스를 리플레이하여, 보호된 엔티티의 쓰기를, 그들을 생성한 보호된 엔티티의 명령어에 액세스할 필요 없이 재현할 수 있다. 이것은 또한 보호된 엔티티가 읽고 트레이싱된 엔티티가 나중에 액세스한 (트레이싱된 엔티티의 어드레스 공간에 있는) 데이터에 대한 리플레이 액세스를 제공한다. (CCP 데이터와 같은 충분한 부기(bookkeeping) 정보가 이용 가능하다면), 보호된 엔티티의 (트레이싱된 엔티티의 어드레스 공간 내의) 쓰기를 로깅할 수 있지만, (그러한 읽기가 나중에 캐시를 무효로 취급하는 것 때문에 로깅될 것이라면) 읽기는 로깅할 수 없는 하이브리드 접근법이 가능하다.

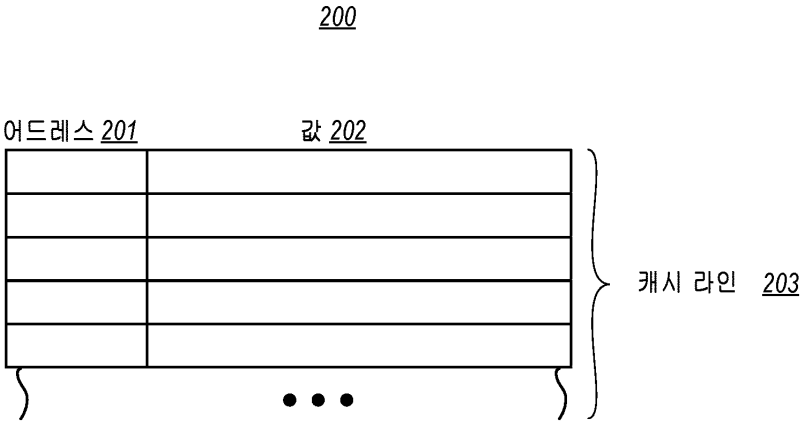
[0165] 본 발명은 그 사상 또는 본질적인 특성을 벗어나지 않고 다른 특정 형태로 구현될 수 있다. 설명된 실시예는 모든 면에서 단지 예시적이고 제한적이지 않은 것으로 간주되어야 한다. 그러므로, 본 발명의 범위는 전술한 설명이 아니라 첨부된 청구범위에 의해 표시된다. 청구범위의 의미 및 등가 범위 내에 있는 모든 변경은 그 범위 내에 포괄되어야 한다.

도면

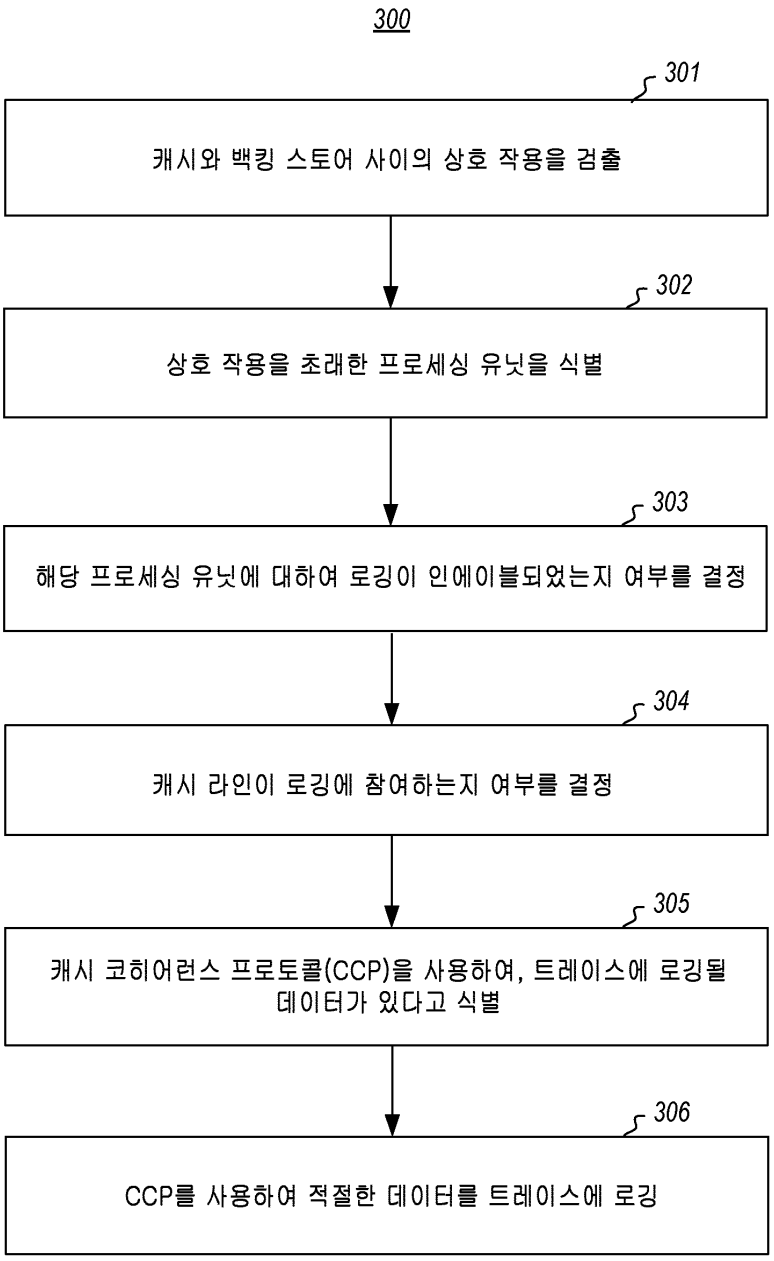
도면1


100
컴퓨터 시스템 101
시스템 메모리 103
애플리케이션 코드 103a
애플리케이션 런타임 데이터 103b
입력/출력 하드웨어 105
프로세서 102
프로세싱 유닛 102a
공유 캐시 102b
마이크로코드 102c
레지스터 102d
버퍼 102e
TLB 102f
데이터 스토어 104
트레이서 104a
운영 체제 커널 104b
애플리케이션 104c
트레이스 파일 104d

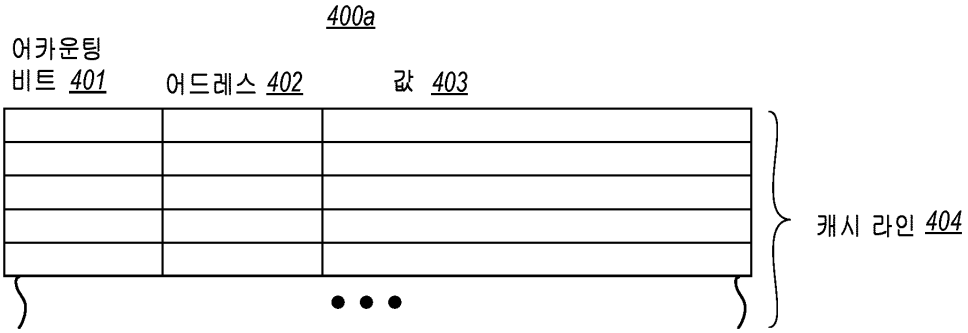
도면2



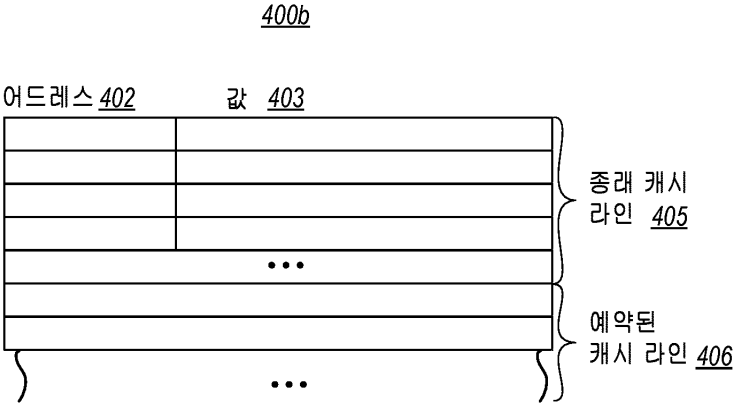
도면3


300
301
캐시와 백킹 스토어 사이의 상호 작용을 검출
302
상호 작용을 초래한 프로세싱 유닛을 식별
303
해당 프로세싱 유닛에 대하여 로깅이 인에이블되었는지 여부를 결정
304
캐시 라인이 로깅에 참여하는지 여부를 결정
305
캐시 코히어런스 프로토콜(CCP)을 사용하여, 트레이스에 로깅될 데이터가 있다고 식별
306
CCP를 사용하여 적절한 데이터를 트레이스에 로깅

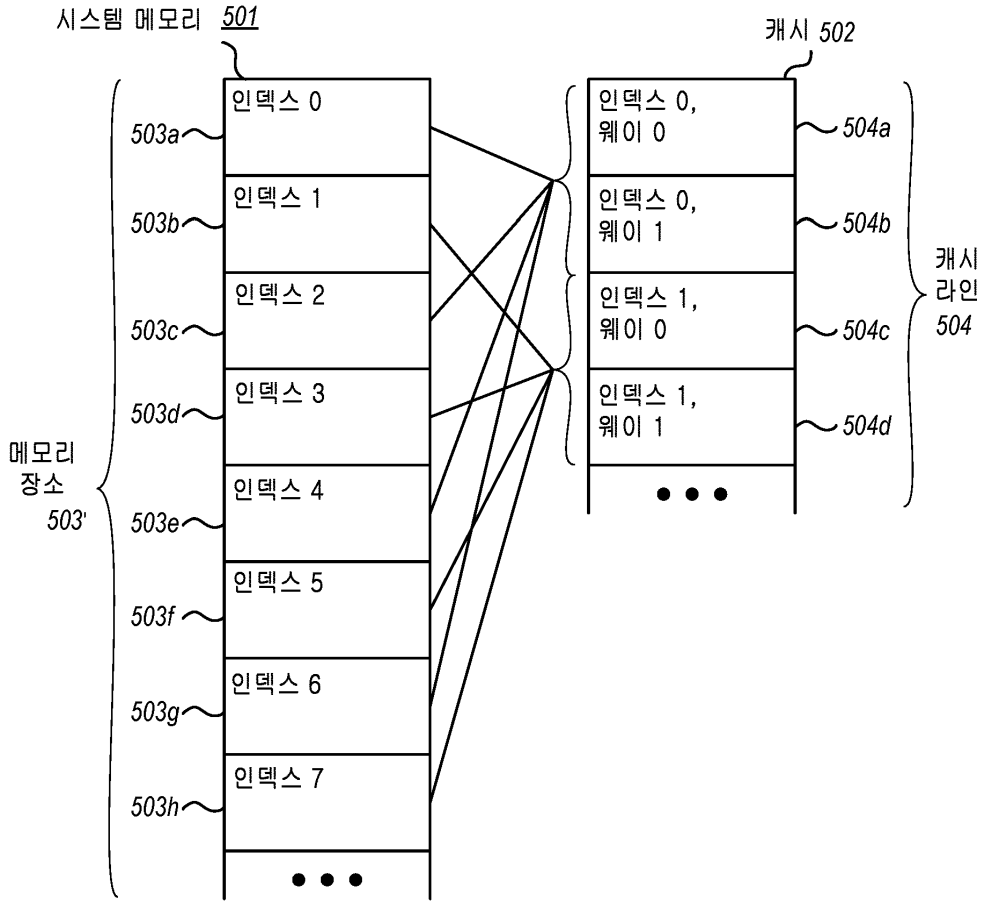
도면4a


400a
어카운팅 비트 401
어드레스 402
값 403
캐시 라인 404

도면4b


400b
어드레스 402
값 403
종래 캐시 라인 405
예약된 캐시 라인 406

도면5


시스템 메모리 501
캐시 502
인덱스 0
인덱스 1
인덱스 2
인덱스 3
인덱스 4
인덱스 5
인덱스 6
인덱스 7
503a
503b
503c
503d
503e
503f
503g
503h
메모리 장소 503'
인덱스 0, 웨이 0
인덱스 0, 웨이 1
인덱스 1, 웨이 0
인덱스 1, 웨이 1
504a
504b
504c
504d
캐시 라인 504

도면6a

600a

601	602a	602b	602c	602d
ID	**P0**	**P1**	**P2**	**P3**
0	R[1]			
1		R[1]		
2	W[2]			
3		R[2]		
4	W[3]			
5			R[3]	
6		R[3]		
7				R[3]
8				W[4]
9	W[5]			
10			R[5]	
11		R[5]		
12	R[5]			
13	라인을 제거			

도면6b

600b

601 ID	602a P0	602b P1	602c P2	602d P3
0	S			
1	S	S		
2	M	I		
3	S	S		
4	M	I		
5	S		S	
6	S	S	S	
7	S	S	S	S
8	I	I	I	M
9	M			I
10	S		S	
11	S	S	S	
12	S	S	S	
13	-	-	-	-

도면6c

600c

601 ID	603 유닛 비트				604 인덱스	605 플래그
0	1	0	0	0	0	1
1	1	1	0	0	1	1
2	1	0	0	0	0	1
3	1	1	0	0	1	1
4	1	0	0	0	0	1
5	1	0	1	0	2	1
6	1	1	1	0	1	1
7	1	1	1	0	1	1
8	0	0	0	0	-1	0
9	0	0	0	0	-1	0
10	0	0	1	0	2	1
11	0	1	1	0	1	1
12	1	1	1	0	0	1
13	0	0	0	0	-1	0

도면6d

600d

601 606

ID	로그 데이터
0	P0; @; IC[0]; DATA [1]
1	P1; @; IC[1]; R->R; P0->P0, P1
2	P0; @; IC[2]; R->W; P0, P1->P0
3	P1; @; IC[3];W->R; P0->P0, P1
4	P0; @; IC[4]; R->W; P0, P1->P0
5	P2; @; IC[5]; W->R; P0->P0, P2
6	P1; @; IC[6]; R->R; P0, P2->P0, P1, P2
7	
8	
9	
10	P2; @; IC[10]; DATA[5], P2
11	P1; @; IC[11]; R->R; P2->P1, P2
12	P0; @; IC[12]; R->R; P1, P2->P0, P1, P2
13	

도면7a

700a

701 ID	702a P0	702b P1	702c P2
0	R[1]		
1		R[1]	
2	W[2]		
3		R[2]	
4			R[2]
5	R[2]		
6		R[2]	
7			R[2]
8	W[3]		
9	R[3]		
10			R[3]

도면7b

700b

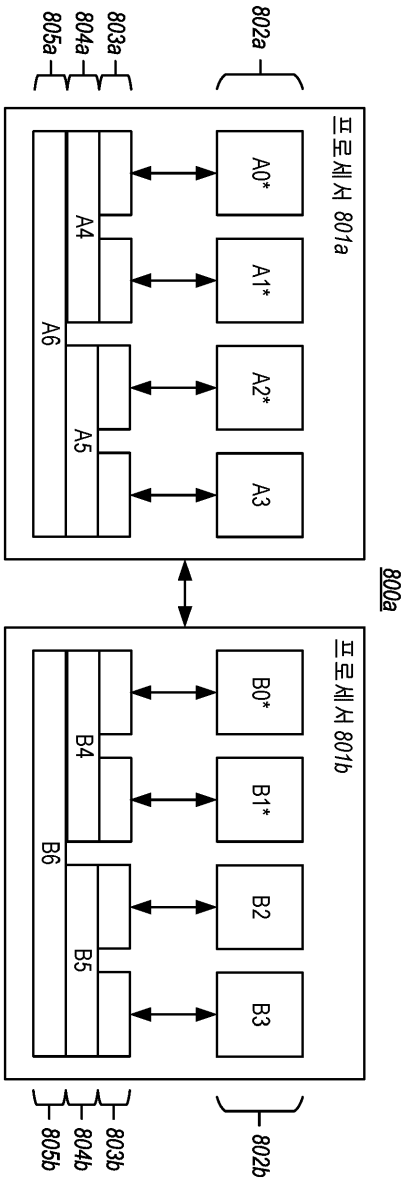
0	P0; @; IC[0]; DATA [1]
1	P1; @; IC[1]; R->R; P0->P0, P1
2	P0; @; IC[2]; R->W; P0, P1->P0
3	P1; @; IC[3];W->R; P0->P0, P1
4	P2; @; IC[4]; R->R; P0, P1->P0, P1, P2
5	
6	
7	
8	P0; @; IC[8]; R->W; P0, P1, P2->P0
9	
10	P2; @; IC[10]; R->R; P0->P0, P2

도면7c

700c

0	P0; @; IC[0]; DATA [1]
1	P1; @; IC[1]; R->R; P0-> P1
2	P0; @; IC[2]; R->W; P1->P0
3	P1; @; IC[3];W->R; P0->P1
4	P2; @; IC[4]; R->R; P1->P2
5	P0; @; IC[5]; R->R; P2->P0
6	P1; @; IC[6]; R->R; P0->P1
7	P2; @; IC[7]; R->R; P1->P2
8	P0; @; IC[8]; R->W; P2->P0
9	
10	P2; @; IC[10]; R->R; P0->P2

도면8a


802a
803a
804a
805a
프로세서 801a
A0*
A1*
A2*
A3
A4
A5
A6
800a
프로세서 801b
B0*
B1*
B2
B3
B4
B5
B6
802b
803b
804b
805b

도면8b

800b

ID	A0	A1	A2	A3	B0	B1	B2	B3
0	R[1]							
1					R[1]			
2		R[1]						
3			W[2]					
4			R[2]					
5					R[2]			
6						R[2]		
7					R[2]			
8	W[3]							
9	W[4]							
10		R[4]						
11			R[4]					
12					R[4]			
13						R[4]		
14	R[4]							

도면9a

900a

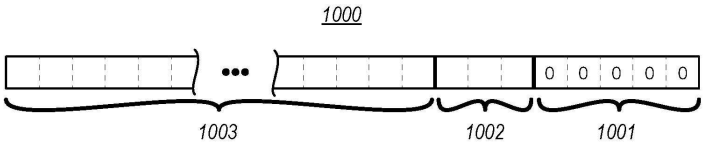
ID	P0	P1
0	W[1]	
1		R[1]
2	R[1]	
3		R[1]
4	W[2]	
5		R[2]

도면9b

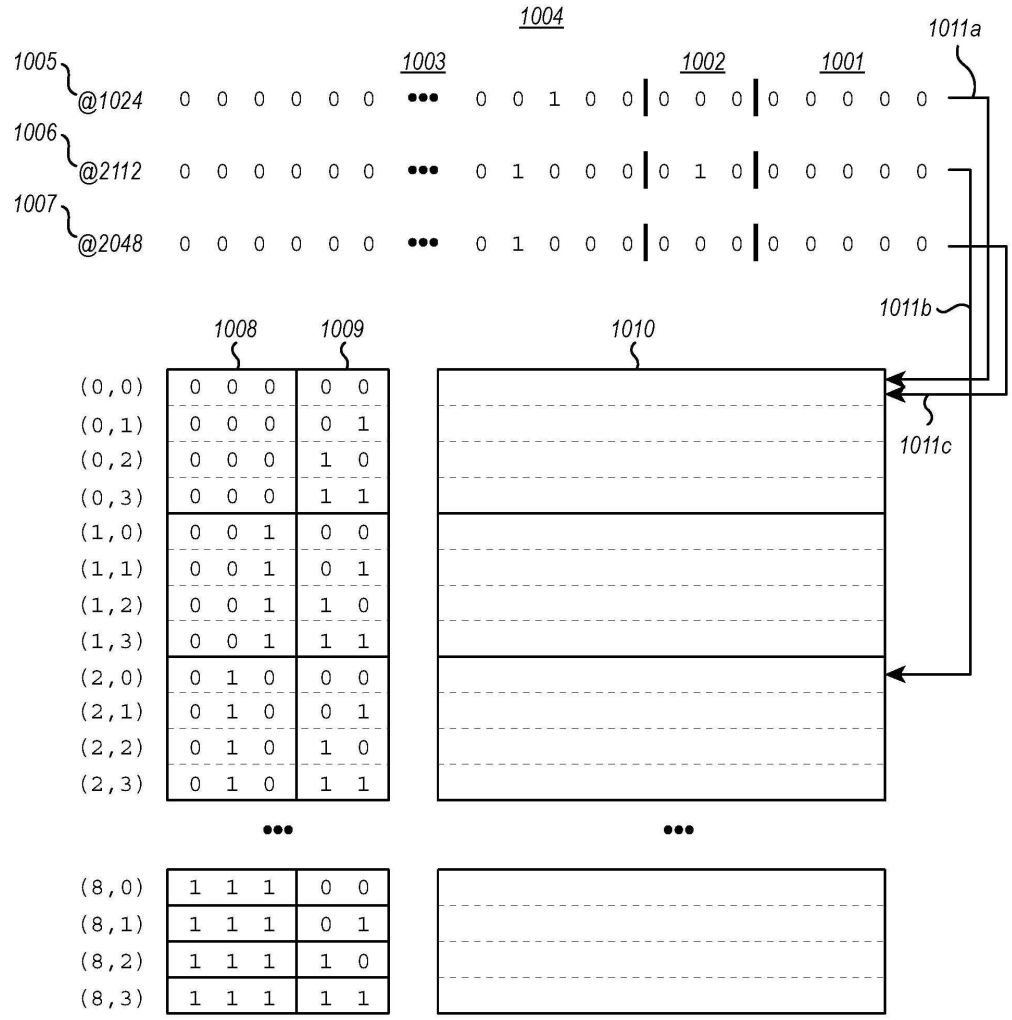
900b

ID	CCP 유닛 + 플래그	CCP 인덱스 + 플래그
0		
1	DATA 로깅	DATA 로깅
2	R->R 로깅	R->R 로깅
3		R->R 로깅
4	R->W 로깅	R->W 로깅
5	W->R 로깅	W->R 로깅

도면10a



도면10b


1004
1005
1003
1002
1001
1011a
@1024
0 0 0 0 0 0 ••• 0 0 1 0 0 0 0 0 0 0 0 0 0
1006
@2112
0 0 0 0 0 0 ••• 0 1 0 0 0 0 1 0 0 0 0 0 0
1007
@2048
0 0 0 0 0 0 ••• 0 1 0 0 0 0 0 0 0 0 0 0 0
1011b
1008
1009
1010
1011c
(0,0) 0 0 0 0 0
(0,1) 0 0 0 0 1
(0,2) 0 0 0 1 0
(0,3) 0 0 0 1 1
(1,0) 0 0 1 0 0
(1,1) 0 0 1 0 1
(1,2) 0 0 1 1 0
(1,3) 0 0 1 1 1
(2,0) 0 1 0 0 0
(2,1) 0 1 0 0 1
(2,2) 0 1 0 1 0
(2,3) 0 1 0 1 1
•••
•••
(8,0) 1 1 1 0 0
(8,1) 1 1 1 0 1
(8,2) 1 1 1 1 0
(8,3) 1 1 1 1 1