



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2024-0082925
(43) 공개일자 2024년06월11일

(51) 국제특허분류(Int. Cl.)
G06F 21/52 (2013.01) G06F 21/55 (2013.01)
(52) CPC특허분류
G06F 21/52 (2013.01)
G06F 21/55 (2013.01)
(21) 출원번호 10-2022-0167071
(22) 출원일자 2022년12월02일
심사청구일자 2022년12월02일

(71) 출원인
울산과학기술원
울산광역시 울주군 언양읍 유니스트길 50
(72) 발명자
백웅기
울산광역시 울주군 언양읍 유니스트길 50
김소웅
울산광역시 울주군 언양읍 유니스트길 50
한명균
울산광역시 울주군 언양읍 유니스트길 50
(74) 대리인
특허법인(유한)아이시스

전체 청구항 수 : 총 4 항

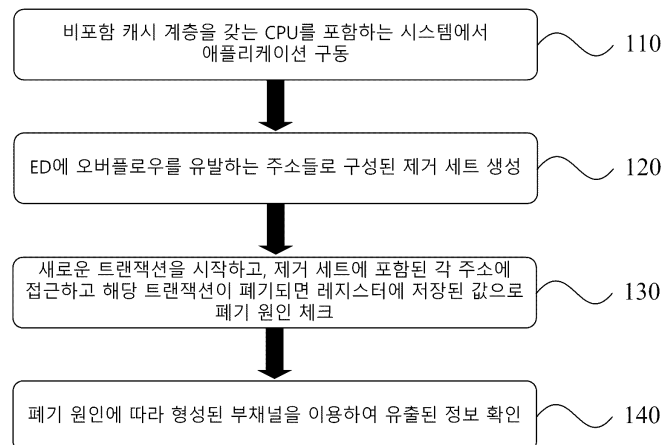
(54) 발명의 명칭 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법

(57) 요약

비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법은 컴퓨터 장치는 공격 대상인 애플리케이션을 구동하는 단계, 상기 컴퓨터 장치는 연장 디렉토리(extended directory, ED)에 오버플로우를 유발하는 주소들을 포함하는 제거 세트(eviction set)를 생성하는 단계, 상기 컴퓨터 장치는 새로운 트랜잭션을 시작하고, 상기 제거 세트에 포함된 각 주소에 접근하고 바쁜 대기 중 상기 트랜잭션이 폐기되면 레지스터에 저장된 값을 기준으로 폐기 원인을 체크하는 단계 및 상기 컴퓨터 장치는 상기 폐기 원인에 따라 부채널을 이용하여 유출 정보를 확인하는 단계를 포함한다.

대표도 - 도4

100



이 발명을 지원한 국가연구개발사업

과제고유번호 1711157895
 과제번호 2021R1A2C1011482
 부처명 과학기술정보통신부
 과제관리(전문)기관명 한국연구재단
 연구사업명 개인기초연구(과기정통부)
 연구과제명 고효율, SLO 보장 워크로드 콘솔리테이션을 위한 통합형 이중 서버 자원 관리 기법

연구 및 개발

기 여 율 40/100
 과제수행기관명 울산과학기술원
 연구기간 2022.03.01 ~ 2023.02.28

이 발명을 지원한 국가연구개발사업

과제고유번호 1711152885
 과제번호 2018-0-00769-005
 부처명 과학기술정보통신부
 과제관리(전문)기관명 정보통신기획평가원
 연구사업명 SW컴퓨팅산업원천기술개발
 연구과제명 인공지능 시스템을 위한 뉴로모픽 컴퓨팅 SW 플랫폼 기술 개발

기 여 율 30/100
 과제수행기관명 한국전자통신연구원
 연구기간 2022.01.01 ~ 2022.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호 1711152517
 과제번호 2021-0-01817-002
 부처명 과학기술정보통신부
 과제관리(전문)기관명 정보통신기획평가원
 연구사업명 정보통신방송혁신인재양성
 연구과제명 하이퍼-컴포저블 데이터센터를 위한 차세대 컴퓨팅 기술 개발

기 여 율 30/100
 과제수행기관명 울산과학기술원
 연구기간 2022.01.01 ~ 2022.12.31

명세서

청구범위

청구항 1

컴퓨터 장치는 공격 대상인 비포함 캐시 계층을 갖는 CPU를 이용하여 애플리케이션을 구동하는 단계;

상기 컴퓨터 장치는 연장 디렉토리(extended directory, ED)에 오버플로우를 유발하는 주소들을 포함하는 제거 세트(eviction set)를 생성하는 단계;

상기 컴퓨터 장치는 새로운 트랜잭션을 시작하고, 상기 제거 세트에 포함된 각 주소에 접근하고 바쁜 대기 중 상기 트랜잭션이 폐기되면 레지스터에 저장된 값을 기준으로 폐기 원인을 체크하는 단계; 및

상기 컴퓨터 장치는 상기 폐기 원인에 따라 부채널을 이용하여 유출 정보를 확인하는 단계를 포함하는 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법.

청구항 2

제1항에 있어서,

상기 제거 세트를 생성하는 단계는

상기 컴퓨터 장치가 동일한 LLC 세트에 매칭되는 후보 주소 세트를 입력받는 단계; 및

상기 컴퓨터 장치는 상기 제거 세트에 포함된 주소들의 개수가 LLC의 연관도(associativity)+1이 될 때까지 반복하여, 상기 후보 주소 세트에서 임의로 어느 하나의 주소를 제거하고 남은 주소들이 충돌하는지 확인하고, 상기 어느 하나의 주소가 충돌을 일으키지 않는 경우 상기 하나의 주소를 상기 제거 세트에 삽입하는 단계를 포함하는 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법.

청구항 3

제3항에 있어서,

상기 컴퓨터 장치는 특정 트랜잭션을 수행하면서 상기 남은 주소들 및 상기 남은 주소들 중 동일한 L2 캐시 세트에 매핑되는 주소들에 접근한 후 상기 특정 트랜잭션이 폐기되지 않는다면 상기 남은 주소들이 충돌하지 않는다고 판단하는 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법.

청구항 4

제1항에 있어서,

상기 컴퓨터 장치는 상기 폐기 원인이 용량 오버플로우(capacity overflow) 또는 트랜잭셔널 충돌 중 어느 하나라면, 상기 부채널을 이용하여 유출 정보를 확인하는 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법.

발명의 설명

기술 분야

[0001] 이하 설명하는 기술은 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 기법에 관한 것이다.

배경 기술

[0002] 최근 CPU는 효율적인 캐시 활용을 위하여 비포함 캐시(non-inclusive cache)를 이용하기 시작하였다. 비포함 캐시는 캐시 기반의 부채널 공격(side-channel attacks)에 강인하다.

선행기술문헌

비특허문헌

- [0003] (비특허문헌 0001) Mengjia Yan et al., Attack Directories, Not Caches: Side Channel Attacks in a Non-Inclusive World, 2019 IEEE Symposium on Security and Privacy

발명의 내용

해결하려는 과제

- [0004] CPU에 대한 공격 방법 연구는 CPU 개발 및 개량에 기여하고 있다. 종전 타이머(timer) 기반 캐시 부채널 공격은 고정밀 타이머가 필요하다.

- [0005] 이하 설명하는 기술은 고정밀 타이머를 사용하지 않고 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 기법을 제안하고자 한다. 또한 이하 설명하는 기술은 LLC(Last Level Cache) 슬라이스 매핑 함수가 메모리 객체들을 같은 비율로 슬라이스에 나누지 않는 경우에도 사용 가능한 제거-세트(eviction-set) 생성 기법도 제안하고자 한다.

과제의 해결 수단

- [0006] 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법은 컴퓨터 장치는 공격 대상인 비포함 캐시 계층을 갖는 CPU를 이용하여 애플리케이션을 구동하는 단계, 상기 컴퓨터 장치는 연장 디렉토리(extended directory, ED)에 오버플로우를 유발하는 주소들을 포함하는 제거 세트(eviction set)를 생성하는 단계, 상기 컴퓨터 장치는 새로운 트랜잭션을 시작하고, 상기 제거 세트에 포함된 각 주소에 접근하고 바쁜 대기 중 상기 트랜잭션이 폐기되면 레지스터에 저장된 값을 기준으로 폐기 원인을 체크하는 단계 및 상기 컴퓨터 장치는 상기 폐기 원인에 따라 부채널을 이용하여 유출 정보를 확인하는 단계를 포함한다.

발명의 효과

- [0007] 이하 설명하는 기술은 비포함 캐시 계층을 갖는 CPU에 대하여 고정밀 타이머를 사용하지 않고, LLC 슬라이스 매핑 함수가 메모리 객체들을 언제나 같은 비율로 슬라이스에 나누지 않는 경우에도 적용 가능한 공격 기법을 제공한다. CPU 하드웨어 설계자는 해당 공격 기법을 고려한 CPU 설계 내지 개량에 이하 설명하는 기술을 활용할 수 있다.

도면의 간단한 설명

- [0008] 도 1은 제거 세트에서 주소의 개수를 조절하면서 실행되는 트랜잭션의 폐기율(abort rates)을 도시한다.
 도 2는 트랜잭션 폐기 후 트랜잭션 리더가 데이터에 접근한 대기 시간을 관찰한 결과이다.
 도 3은 평문에서 16개의 서로 다른 4자리 비트들에 대하여 공격자가 산출하는 상대적인 트랜잭션 폐기율을 나타낸 결과이다.
 도 4는 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법에 대한 예이다.

발명을 실시하기 위한 구체적인 내용

- [0009] 이하 설명하는 기술은 다양한 변경을 가할 수 있고 여러 가지 실시예를 가질 수 있는 바, 특정 실시예들을 도면에 예시하고 상세하게 설명하고자 한다. 그러나, 이는 이하 설명하는 기술을 특정한 실시 형태에 대해 한정하려는 것이 아니며, 이하 설명하는 기술의 사상 및 기술 범위에 포함되는 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다.

- [0010] 제1, 제2, A, B 등의 용어는 다양한 구성요소들을 설명하는데 사용될 수 있지만, 해당 구성요소들은 상기 용어들에 의해 한정되지는 않으며, 단지 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만 사용된다. 예를 들어, 이하 설명하는 기술의 권리 범위를 벗어나지 않으면서 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소도 제1 구성요소로 명명될 수 있다. 및/또는 이라는 용어는 복수의 관련된 기재된 항목들의 조합 또는 복수의 관련된 기재된 항목들 중의 어느 항목을 포함한다.

- [0011] 본 명세서에서 사용되는 용어에서 단수의 표현은 문맥상 명백하게 다르게 해석되지 않는 한 복수의 표현을 포함하는 것으로 이해되어야 하고, "포함한다" 등의 용어는 설명된 특징, 개수, 단계, 동작, 구성요소, 부분품 또는

이들을 조합한 것이 존재함을 의미하는 것이지, 하나 또는 그 이상의 다른 특징들이나 개수, 단계 동작 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 배제하지 않는 것으로 이해되어야 한다.

[0012] 도면에 대한 상세한 설명을 하기에 앞서, 본 명세서에서의 구성부들에 대한 구분은 각 구성부가 담당하는 주기능 별로 구분한 것에 불과함을 명확히 하고자 한다. 즉, 이하에서 설명할 2개 이상의 구성부가 하나의 구성부로 합쳐지거나 또는 하나의 구성부가 보다 세분화된 기능별로 2개 이상으로 분화되어 구비될 수도 있다. 그리고 이하에서 설명할 구성부 각각은 자신이 담당하는 주기능 이외에도 다른 구성부가 담당하는 기능 중 일부 또는 전부의 기능을 추가적으로 수행할 수도 있으며, 구성부 각각이 담당하는 주기능 중 일부 기능이 다른 구성부에 의해 전담되어 수행될 수도 있음은 물론이다.

[0013] 또, 방법 또는 동작 방법을 수행함에 있어서, 상기 방법을 이루는 각 과정들은 문맥상 명백하게 특정 순서를 기재하지 않은 이상 명기된 순서와 다르게 일어날 수 있다. 즉, 각 과정들은 명기된 순서와 동일하게 일어날 수도 있고 실질적으로 동시에 수행될 수도 있으며 반대의 순서대로 수행될 수도 있다.

[0015] 이하 설명하는 기술은 CPU에 대한 캐시 부채널 공격 기법에 대한 내용을 포함한다. 연구자는 인텔의 CPU를 대상으로 연구하였다. 따라서, 이하 설명이나 실험 내용은 인텔 CPU를 중심으로 설명한다. 다만, 이하 설명하는 기술이 반드시 특정 제품에 국한되는 것은 아니라고 할 것이다.

[0016] 최신 인텔 CPU는 3레벨의 캐시 계층 구조를 갖는다. 각 코어(core)는 프라이빗 L1 및 L2 캐시를 갖는다. L1에 저장된 모든 메모리 객체는 L2에도 저장되어야 한다(inclusive). LLC는 코어들에 공유된다. 각 코어는 LLC에서 슬라이스(slice)라는 동일 크기 영역을 갖는다. 한편, L1/L2는 LLC와 비포함 특성을 갖는다. 즉, L1/L2에 저장된 메모리 객체는 LLC에 저장되지 않을 수 있다.

[0017] 디렉토리는 캐시 계층에 저장되는 캐시 라인들의 일관성 정보를 보유한다. 어느 하나의 코어가 프라이빗 캐시에 존재하지 않는 메모리 객체에 접근하고자 하면, 코어는 해당 캐시 라인에 대하여 위치 및 일관성 상태와 같은 상세한 정보를 디렉토리로부터 확인한다. 디렉토리는 물리적으로 LLC에 위치하고, 각 슬라이스의 디렉토리는 해당 슬라이스에 매핑된 주소들을 갖는 캐시 라인들의 일관성 정보를 추적하게 돕는다.

[0018] 디렉토리는 연장 디렉토리(extended directory, ED) 및 전통적 디렉토리(traditional directory, TD)가 있다. ED는 프라이빗 캐시에만 저장되는 메모리 객체들을 관리한다. TD는 LLC에 저장된 메모리 객체들을 관리한다. 각 LLC 슬라이스에 있는 ED 및 TD는 세트-연관 구조(set-associative structure)를 갖는다. 예컨대, ED 및 TD는 각각 배타적으로 10개의 웨이(way) 및 9개의 웨이를 갖고, 2개 웨이를 공유할 수 있다. ED에 있는 디렉토리 엔트리가 제거되면, 대응하는 캐시 라인에 대한 일관성 정보는 TD로 이동하고 해당 캐시 라인은 L2 캐시로부터 제거되어 LLC로 이동한다. 한편, TD에 있는 디렉토리 엔트리가 제거되면, 대응하는 캐시 라인은 전체 캐시 구조에서 제거된다.

[0019] 2개의 캐시 라인들이 동일한 LLC 슬라이스와 세트에 매핑되면, 그들의 일관성 정보는 동일한 ED 세트에 저장된다. 공격자가 해당 주소 세트에 접근하여 공격 대상의 타깃 캐시 라인에 대한 일관성 정보를 저장하는 해당 ED 세트에 충돌을 유발하면, 타깃 캐시 라인의 ED 엔트리는 ED 세트에서 오버플로우로 제거되고, 타깃 캐시 라인은 L2 캐시에서 제거된다.

[0020] 트랜잭셔널 메모리는 병행 제어(concurrency control)를 하여 병렬 프로그래밍을 단순하게 한다. 트랜잭셔널 메모리 시스템은 트랜잭션의 정확한 실행을 보장한다. 트랜잭션 메모리 시스템은 메모리 객체에 대한 읽기(read) 및 쓰기(write)에 대하여 각각 해당 메모리 객체의 주소를 수집하여 읽기-세트 및 쓰기-세트에 저장한다. 트랜잭셔널 메모리 시스템은 읽기-세트 및 쓰기-세트에서 오버플로우가 발생하는 경우 해당 트랜잭션을 폐기한다. 또한, 트랜잭셔널 메모리 시스템은 트랜잭션이 폐기되면 해당 트랜잭션의 모든 사이드 이펙트(side effect)를 롤백(rollback)한다.

[0021] 연구자는 TSX를 제공하는 최신의 인텔 CPU를 사용하여 연구하였다. 인텔 TSX 트랜잭션들은 아래 표 1과 같이 다양한 원인으로 폐기될 수 있다. 예컨대, 트랜잭션의 읽기-세트에서 오버플로우로 하나의 캐시 라인이 제거되면, 트랜잭션은 실제 읽기-세트 및/또는 쓰기-세트 충돌 없이 폐기된다. 또한, OS로부터의 인터럽트(interrupt) 또는 시스템 명령으로 수행중인 트랜잭션이 폐기될 수 있다.

표 1

Cause #	Description
1	Execution of certain instructions (e.g., CPUID)
2	System calls
3	OS interrupts
4	Deeply nested transactions
5	Page faults and access violations
6	R/W or W/W conflicts with other threads or processes
7	Write-set (i.e., L1 data cache) overflows
8	Read-set (i.e., LLC) overflows

[0023]

[0025]

CPU가 메모리 객체에 접근하는 경우 메모리 객체가 캐시에 저장된다. 메모리 객체가 이미 캐시에 저장된 경우, 메모리 객체는 상위 레벨 캐시로 이동한다. 즉, 메모리 객체의 캐시 저장 여부에 따라 공격 대상(victim)의 메모리 접근 여부를 확인할 수 있다. 캐시 부채널 공격은 다음과 같은 과정을 반복하면서 공격 대상이 수행하는 함수나 접근하는 데이터를 확인할 수 있다. 캐시 부채널 공격은 (i) 공격자가 제거-세트의 주소들에 접근하여 공격 대상의 메모리 객체를 특정 캐시 레벨에서 제거하고, (ii) 공격 대상의 메모리 접근을 기다린 후 (iii) 공격자가 타이머를 이용하여 접근 시간을 측정하여 메모리 객체가 어떤 캐시 레벨에 저장되었는지 확인한다.

[0026]

이하 연구자가 발견한 특성을 먼저 설명하고자 한다. 연구자가 실험한 환경은 아래 표 2와 같다.

표 2

Component	Intel Xeon Gold 6142 (1st Gen.)	Intel Xeon Gold 6242 (2nd Gen.)	Intel Xeon Gold 5318H (3rd Gen.)
Frequency	2.6GHz	2.8GHz	2.5GHz
Core count	16	16	18
L1 I-cache	32KB	32KB	32KB
L1 D-cache	32KB	32KB	32KB
L2 cache	1MB, 16 ways	1MB, 16 ways	1MB, 16 ways
LLC	22MB, 11 ways, 16 slices	22MB, 11 ways, 16 slices	24.75MB, 11 ways, 18 slices
ED	12 ways	12 ways	12 ways
TD	11 ways	11 ways	11 ways

[0028]

[0029]

연구자는 인텔의 Xeon 프로세서 1세대, 2세대 및 3세대를 사용하는 서버 시스템을 대상으로 연구를 하였다. 평가에 사용된 모든 CPU들은 비포함 캐시 구조를 갖고, 인텔 TSX를 지원한다. 서버 시스템의 CPU governor는 모든 실험에 대하여 performance로 설정되었다.

[0030]

연구자는 실행 사이클(execution cycle), 트랜잭션 완료(commit) 및 폐기(abort)의 횟수를 수집하였다. 실행 사이클은 rdtsc 명령어를 사용하여 측정하였다. 실행 시간은 측정한 실행 사이클을 CPU frequency로 나누어 계산될 수 있다. 트랜잭션 완료 및 폐기 횟수는 프로그래밍을 통해 트랜잭션 완료 또는 폐기시에 카운트를 증가하여 수집하였다.

[0032]

부채널 공격은 공격할 타겟 캐시 라인에 대하여 제거 세트가 정의되어야 한다. 제거 세트는 타겟 캐시 라인의 일관성 정보가 저장된 ED 세트에 오버플로우를 유발하는 주소 세트이다.

[0033]

현재 하드웨어 성능 카운터를 이용하여 ED의 상태 변화를 직접 검출하는 방법은 없다. 반대로, LLC 상태 변화는 타이머 이용한 접근 시간(access latencies)을 측정하거나, 읽기-세트에서 오버플로우로 유발되는 인텔 TSX 트랜잭션 폐기가 있는지 점검하여 확인할 수 있다.

[0034]

종래 연구는 제거-세트 생성을 위하여 정확한 타이머를 필요로 하였다. 연구자는 인텔 TSX를 이용하여 정확한

타이머가 필요 없는 새로운 제거-세트 생성 제안하였다. 아래 표 3은 제거-세트 생성을 위한 함수 (generateEvictionSet)에 대한 알고리즘 1에 대한 예이다. 제거-세트 생성은 컴퓨터 장치가 수행한다고 설명한다. 컴퓨터 장치는 PC, 서버 등과 같은 데이터 처리 장치일 수 있다.

표 3

Algorithm 1 The generateEvictionSet function

```

1: procedure GENERATEEVICIONSET(candidateSet)
2:   evictionSets  $\leftarrow \emptyset$ 
3:   while candidateSet  $\neq \emptyset$ 
4:     tempSet  $\leftarrow$  candidateSet
5:     for testAddr in candidateSet
6:       tempSet.remove(testAddr)
7:       if checkConflict(tempSet) = false
8:         tempSet.insert(testAddr)
9:       end if
10:      if tempSet.size() =  $W_{LLC} + 1$ 
11:        break
12:      end if
13:    end for
14:    evictionSet  $\leftarrow$  tempSet
15:    tempSet  $\leftarrow$  tempSet.resize( $W_{LLC}$ )
16:    candidateSet  $\leftarrow$  candidateSet \ evictionSet
17:    testSet  $\leftarrow$  candidateSet
18:    for testAddr in testSet
19:      tempSet.insert(testAddr)
20:      if checkConflict(tempSet) = true
21:        evictionSet.insert(testAddr)
22:        candidateSet.remove(testAddr)
23:      end if
24:      tempSet.remove(testAddr)
25:    end for
26:    evictionSets.insert(evictionSet)
27:  end while
28:  return evictionSets
29: end procedure

```

[0036]

[0038] generateEvictionSet 함수는 candidateSet를 입력받는다. candidateSet는 동일한 LLC 세트에 매핑되는 주소들 세트를 포함한다. 이때 주소들 세트는 서로 다른 LLC 슬라이스에 대한 것일 수 있다. 연구자는 candidateSet의 크기를 $3 \times S \times W_{LLC}$ 으로 설정하였다. 여기서, W_{LLC} 는 LLC의 연관도(associativity)이고, S는 LLC 슬라이스 개수이다. 이 경우 candidateSet의 크기는 제거-세트가 충분한 개수의 주소들을 갖게 한다.

[0039] generateEvictionSet 함수는 evictionSet를 출력한다. evictionSet는 S개의 서브 세트로 구성된다. 각 서브 세트는 동일한 LLC 슬라이스에 매핑되는 주소들을 포함한다. generateEvictionSet 함수는 주소들의 서브 세트 S개를 생성하고, candidateSet의 주소들을 서브 세트 S에 삽입한다. 공격 대상의 타깃 캐시 라인과 동일한 LLC 슬라이스에 매핑되는 서브 세트는 타깃 캐시 라인을 위한 제거 세트가 된다.

[0040] generateEvictionSet 함수는 두 개의 단계(phase)로 동작한다.

[0041] 첫 번째 단계에서 컴퓨터 장치는 대응하는 LLC 슬라이스에서 타깃 LLC 세트에 대한 오버플로우를 위하여 주소들의 최소 개수(즉, $W_{LLC}+1$)를 갖는 서브 세트를 채운다(라인 4-13).

[0042] candidateSet의 각 testAddr에 대하여(라인 5), 컴퓨터 장치는 tempSet에서 testAddr을 임시적으로 제거한다(라인 6). 컴퓨터 장치는 알고리즘 2에 나타난 checkConflict 함수를 사용하여 tempSet에 남아 있는 주소가 서

로 충돌을 유발하는지 체크한다(라인 7). 알고리즘 2는 아래 표 4에 나타낸다. checkConflict의 출력이 참이면, tempSet가 적어도 하나의 LLC 세트에 오버플로우를 유발하는 주소 세트를 포함한다는 의미이다. 따라서, 컴퓨터 장치는 다음 반복 과정에서 testAddr의 삽입 없이 tempSet으로 돌아간다(라인 7-8 참고). tempSet에 있는 주소들이 서로 충돌을 일으키지 않는다면, 컴퓨터 장치는 testAddr을 tempSet에 삽입한다(라인 8).

[0043] 컴퓨터 장치는 tempSet의 크기가 $W_{LLC}+1$ 가 될때까지 루프를 반복한다. 컴퓨터 장치는 $W_{LLC}+1$ 개의 주소들을 동일한 LLC 슬라이스에 매핑하고, 대응하는 LLC 세트에 오버플로우를 일으킬 수 있는 최소 개수의 주소들 설정한다.

[0044] 컴퓨터 장치는 evictionSets의 엔트리에서 이 주소들을 저장한다(라인 11). evictionSets의 각 엔트리는 하나의 세트이고, 타깃 캐시 라인으로 특정 LLC 슬라이스에서 동일한 LLC 세트에 오버플로우를 유발할 수 있는 주소들을 포함한다.

[0045] 첫 번째 단계 후에, candidateSet는 여전히 주소들을 포함하고 있을 수 있다. 해당 주소들은 첫 번째 단계에서 발견된 $W_{LLC}+1$ 개의 주소들로서 동일한 LLC 슬라이스에 매핑되지만 대응하는 evictionSets의 엔트리에 삽입되지 않은 것이다.

[0046] 두 번째 단계에서 generateEvictionSet 함수는 남은 각 주소(즉, testAddr)에 대하여 반복하면서 testAddr이 첫 번째 단계에서 발견된 $W_{LLC}+1$ 개의 주소들과 충돌하는지 체크한다(라인 18-25). testAddr이 첫 번째 단계에서 발견된 주소와 충돌한다면, testAddr이 동일한 LLC 슬라이스에 또한 매핑되고, evictionSets의 대응하는 엔트리에 주소들로 설정된다는 의미이다. 따라서, 컴퓨터 장치는 evictionSets의 대응하는 엔트리에 testAddr을 삽입한다(라인 18).

[0047] 첫 번째 단계 및 두 번째 단계 후에 컴퓨터 장치는 LLC 슬라이스에 매핑된 candidateSet의 모든 주소들을 추출한다. generateEvictionSet 함수가 완료되면, evictionSets의 각 엔트리는 대응하는 LLC 슬라이스에 속하는 candidateSet의 모든 주소를 담게 된다.

표 4

Algorithm 2 The checkConflict function	
1:	procedure CHECKCONFLICT(testAddrSet)
2:	if beginTransaction() = STARTED
3:	for addr in testAddrSet
4:	access(addr)
5:	end for
6:	for addr in conflictSetL2
7:	access(addr)
8:	end for
9:	endTransaction()
10:	return false
11:	else ▷ Fallback execution path
12:	return true
13:	end if
14:	end procedure

[0049]

[0051] 알고리즘 2는 checkConflict 함수를 나타낸다. checkConflict 함수는 인텔 TSX를 사용하되 정확한 타이머를 필요로 하는 종래 기술과 다르다. checkConflict 함수는 주소들 세트를 포함하는 testAddrSet를 입력 파라미터로 입력받는다. checkConflict 함수는 testAddrSet에 포함된 주소들이 서로 충돌한다면 참(true)를 출력(리턴)한다. 반대의 경우라면 checkConflict 함수는 거짓(false)을 출력한다.

[0052] checkConflict 함수는 트랜잭션을 시작한다(라인 2). 트랜잭션이 폐기된다면, checkConflict 함수는 참(true)을 출력한다(라인 11-13). 트랜잭션내에서 checkConflict 함수는 testAddrSet에서 모든 주소에 접근한다. 트랜잭션이 아직 폐기 되기 전이라면, checkConflict 함수는 conflictSetL2에 모든 주소들에 접근한다.

conflictSetL2는 testAddrSet의 주소들과 동일한 L2 캐시 세트에 매핑되는 주소 세트를 포함한다. 이는 testAddrSet의 주소들에 연관된 캐시 라인들이 L2 캐시에 저장되지 않게 한다. testAddrSet 및 conflictSetL2의 모든 주소들에 접근한 후 트랜잭션이 폐기되지 않는다면, testAddrSet의 주소들은 충돌하지 않는다는 의미이다. 이 경우 checkConflict 함수는 거짓(false)을 출력한다.

- [0054] 이하 설명하는 기술은 인텔의 트랜잭셔널 메모리(Transactional memory) 관련 명령어 집합인 인텔 TSX와 디렉토리들(coherence directories) 사이의 알려지지 않은 상호작용에 기반한다. 연구자는 트랜잭션 동안 접근한 캐시 라인에 연관된 ED 엔트리가 제거되는 경우, LLC에 상기 접근한 캐시 라인에 저장되어 있는 경우라도 해당 트랜잭션은 폐기된다는 것을 발견하였다.
- [0055] 연구자는 동시에 2개의 프로세스에서 2개의 단순한 애플리케이션을 실행하면서 실험 결과를 수집하였다.
- [0056] 트랜잭셔널 리더(TR)는 8바이트 데이터를 읽고, 트랜잭션내에 더미 연산을 수행한다. 트랜잭셔널 리더가 8바이트 데이터를 읽고난 후 EV(evictor)가 TR 프로세스가 접근한 8바이트 데이터에 대응하는 제거 세트에서의 설정 가능한 개수(N)의 주소들을 읽는다.
- [0057] 도 1은 제거 세트에서 주소의 개수를 조절하면서 실행되는 트랜잭션의 폐기율(abort rates)을 도시한다. 도 1은 제거 세트의 주소 개수 $N = 0 \sim 16$ 개로 다양하게 설정하면서 실험을 수행한 결과이다. 연구자는 설정된 주소 개수 각각에 대하여 전술한 실험(제거 세트 생성)을 100,000번을 반복하였다.
- [0058] 먼저, N(주소 개수)이 11 이하인 경우 트랜잭션 폐기율은 낮았다. ED 및 LLC의 연관도가 12 및 11이기 때문에, N이 11보다 낮은 경우 트랜잭션 폐기율이 낮게 관찰되었다. 한편, N이 11보다 큰 경우 트랜잭션 폐기율이 거의 100%가 되었다. 이와 같은 트랜잭션 폐기율 증가는 ED 세트의 오버플로우로 유발된 것으로 추정된다.
- [0059] 높은 트랜잭션 폐기율의 원인을 파악하기 위하여 연구자는 트랜잭션 폐기 후에 TR 프로세스가 다시 8바이트 데이터에 접근하며 접근 대기 시간(access latency)을 관찰하였다. 도 2는 트랜잭션 폐기 후 트랜잭션 리더가 데이터에 접근한 대기 시간을 관찰한 결과이다. 아래 표 5는 메모리 계층 구조의 각 구성에서의 접근 대기 시간을 나타낸다. 연구자는 인텔 MLC(Memory Latency Checker)를 사용하여 캐시 접근 대기 시간을 측정하였고, rdtsc 명령어 기반하여 구현된 마이크로벤치마크(microbenchmark)를 사용하여 메모리 접근 대기 시간을 측정하였다.

표 5

Component	Latency (Cycles)
L1 cache	4
L2 cache	14
Last-level cache (LLC)	55
DRAM	264

- [0061]
- [0063] 도 2를 살펴보면, 트랜잭션 폐기 후 8 바이트 데이터에 접근하는 대기 시간은 LLC 접근 대기 시간과 유사하다. 이는 8 바이트 데이터를 위한 폐기된 캐시 라인이 LLC에 저장된다는 것을 의미한다. 사실 EV에서 접근한 다수의 데이터는 L2의 연관도가 충분히 높기 때문에(즉 16) L2 캐시에 유지될 가능성이 높다. EV가 접근한 데이터는 8 바이트 데이터를 저장한 LLC 세트에 적은 영향을 주기 때문에, 트랜잭션 폐기 후에 8 바이트 데이터는 여전히 LLC (트랜잭셔널 읽기-세트를 포함하는)에 남는다.
- [0064] L2 캐시 오버플로우가 트랜잭셔널 폐기의 원인이 아니라는 것을 확인하기 위하여, 연구자는 서로 다른 ED 세트이지만 동일한 L2 캐시 세트에 속하는 N개의 주소들에 접근하는 단순한 애플리케이션을 실행하였다. L2 캐시 폐기가 트랜잭셔널 폐기의 원인이라면, N이 L2 캐시 연관도(즉, 16)보다 커지면 트랜잭셔널 폐기율은 높아질 것이다. 그러나 N이 312로 설정되어도 트랜잭셔널 폐기율은 여전히 낮은 상태를 보인다. 따라서, 실험 결과를 해석하면 L2 캐시 폐기는 트랜잭셔널 폐기의 원인은 아니라고 할 것이다.
- [0065] 결국, N이 11보다 높아질 때 트랜잭셔널 폐기율이 높아지는 원인은 ED 세트 오버플로우라고 해석된다. 실험 결

과를 정리하면, 트랜잭션에서 접근한 캐시 라인의 ED 엔트리가 폐기되면 LLC에 여전히 해당 캐시 라인이 있다고 하더라도 트랜잭션 폐기가 발생한다는 것이다.

[0066] 상기 실험 결과와 같은 인텔 TSX 트랜잭션과 디렉토리 사이의 알려지지 않은 상호 관계에 기반하여 연구자는 인텔 TSX를 사용하여 비포함 캐시 구조에 적용가능한 새로운 부채널 공격 기법을 제안한다.

[0068] 아래 표 6의 알고리즘 3은 연구자가 제안하는 부채널 공격 과정에 해당한다.

표 6

Algorithm 3 Pseudocode for the DPRIME+DABORT attack

```

1: procedure DPRIMEDABORTATTACK(evictionSet)
2:   while continueToAttack() = true
3:     if beginTransaction() = STARTED
4:       for addr in evictionSet
5:         access(addr)
6:       end for
7:       busyWait()
8:       endTransaction()
9:     else ▷ Fallback execution path
10:      status ← getTransactionAbortStatus()
11:      if status = CONFLICT or status = CAPACITY
12:        inferVictimActivity()
13:      end if
14:    end if
15:  end while
16: end procedure

```

[0069]

[0070] 공격자는 해당 CPU 공격을 위한 컴퓨터 장치를 의미한다. 공격자는 PC, 서버 등과 같은 데이터 처리 및 연산이 가능한 장치에 해당한다. 표 6에서 연구자가 제안한 공격 과정을 수행하는 함수를 "DPRIME+DABORT attack"라고 표현하였다. DPRIME+DABORT attack()는 알고리즘 1에서 산출되는 evictionSet를 입력으로 받는다(라인 1). 즉, 공격자는 사전에 공격 대상에 대한 evictionSet를 산출한다.

[0071] 공격자는 beginTransaction 함수로 새로운 트랜잭션을 시작하여 공격을 시작한다(라인 3). 공격자는 제거 세트에 있는 모든 주소들에 접근한다(라인 4-5). 제거 세트는 W_{ED} 개의 주소를 포함하고, W_{ED} 는 ED의 연관도를 의미한다.

[0072] 제거 세트의 모든 주소들에 접근한 후 공격자는 트랜잭션이 폐기될 때까지 바쁜 대기(busy waiting)를 수행한다(라인 6-7). 트랜잭션이 폐기되면, 공격자는 비특권 모드(non-privileged mode)에서 접근 가능한 %eax 레지스터에 저장된 값을 읽어 폐기의 원인을 체크한다(라인 9-14).

[0073] 폐기의 원인이 용량 오버플로우(capacity overflow) 또는 트랜잭셔널 충돌 중 어느 하나라면, 공격 대상이 타깃 캐시 라인에 접근할때 ED 세트 오버플로우가 발생함을 나타낸다. 이는 ED 및 인텔 TSX 사이의 상호 작용을 통해 공격 대상과 공격자 사이의 부채널을 구성한다. 공격자는 유출된 정보를 저장한다. 필요하다면, 공격자는 다시 공격을 수행한다.

[0074] 폐기의 원인이 OS 컨텍스트 스위치(context switch)라면, 공격 대상은 타깃 캐시 라인에 접근하지 않았다는 의미이다. 필요한 경우, 공격자는 전술한 과정을 반복한다.

[0075] 알고리즘 3에서 DPRIME+DABORT attack은 인텔 TSX에서 검출될 수 있는 하드웨어 상태 변화를 이용하기 때문에 정확한 타이머가 필요하지 않다. 또한 DPRIME+DABORT attack은 종래 타이머 기반의 부채널 공격과 달리 반복적으로 하드웨어 상태 변화를 체크하지 않아도 된다. 이는 공격 대상의 행동으로 ED 세트 오버플로우가 발생할 때 인텔 TSX 트랜잭션이 폐기를 보장하기 때문이다.

- [0077] 연구자는 부채널 공격 기법(DPRIME+DABORT attack)의 성능을 검증하였다. 연구자는 AES(Advanced Encryption Standard)의 OpenSSL의 T-table 구현을 사용한 애플리케이션을 공격하였다.
- [0078] AES에서 첫 번째 T-table의 엔트리에 접근하기 위한 인덱스는 비밀키의 앞쪽 바이트와 평문의 앞쪽 바이트의 XOR 연산으로 연산된다. 비밀키의 처음 4자리 비트들과 평문의 처음 4자리 비트들이 매칭되면, OpenSSL의 AES는 첫 번째 T-table(즉, Te_0)의 첫 번째 캐시 라인(즉, $L0_{Te_0}$)에 접근하는 방식으로 구현된다.
- [0079] 공격자는 DPRIME+DABORT attack을 통해 비밀키의 처음 4개 비트들을 정확하게 추론하였다. 공격자는 $L0_{Te_0}$ 와 충돌하는 제거 세트를 생성하였다. 공격자는 반복적으로 OpenSSL의 AES에 기반한 암호화 애플리케이션을 구동하는 공격 대상에 암호화 요청을 전송한다. 각 평문에 대하여 공격자는 DPRIME+DABORT attack을 이용한 부채널을 통해 공격 대상이 $L0_{Te_0}$ 에 접근하는지 체크한다. 공격자가 공격 대상이 $L0_{Te_0}$ 에 접근하게 하는 평문을 발견하면, 공격자는 비밀키의 처음 4자리 비트들을 추론할 수 있다.
- [0080] AES에 대한 DPRIME+DABORT attack 공격의 유효성은 다음과 같은 실험으로 평가하였다. 공격 대상은 공격자를 포함하는 다른 프로세스로부터 암호화 요청을 수행하는 암호화 애플리케이션을 구동한다. 공격 대상은 비밀키의 처음 4자리 비트를 "1000"으로 사용하였고, 공격자는 평문에 대하여 4자리 비트의 가능한 각 값에 대하여 100,000번 암호화 요청을 전달하였다. 도 3은 평문에서 16개의 서로 다른 4자리 비트들에 대하여 공격자가 산출하는 상대적인 트랜잭션 폐기율을 나타낸 결과이다. 도 3을 살펴보면, 공격자는 평문과 비밀키에서 4자리 비트들이 매칭될 때 가장 높은 트랜잭션 폐기율을 관찰하여 비밀키의 처음 4자리 비트들을 정확하게 추론할 수 있다.
- [0082] 도 4는 비포함 캐시 계층을 갖는 CPU에 대한 부채널 공격 방법(100)에 대한 예이다. 도 4는 전술한 부채널 공격 방법의 주요 내용을 나타내는 순서도의 예이다. 부채널 공격 방법은 전술한 알고리즘 3에서 설명한 과정과 같다.
- [0083] 컴퓨터 장치(공격자)는 비포함 캐시 계층을 갖는 CPU를 포함하는 시스템을 대상으로 공격을 수행하며, 해당 시스템에서 특정 애플리케이션을 구동한다(110). 여기서 공격 대상인 시스템은 특정 프로세서를 갖는 서버 등일 수 있다.
- [0084] 컴퓨터 장치는 ED에 오버플로우를 유발하는 주소들을 포함하는 제거 세트를 생성한다(120). 제거 세트 생성 과정은 알고리즘 1에서 설명한 바와 같다. 간략하게 설명하면 컴퓨터 장치가 동일한 LLC 세트에 매칭되는 후보 주소 세트를 입력받는다. 컴퓨터 장치는 제거 세트에 포함된 주소들의 개수가 LLC의 연관도(associativity)+1이 될 때까지 반복하여, 후보 주소 세트에서 임의로 어느 하나의 주소를 제거하고 남은 주소들이 충돌하는지 확인하고, 어느 하나의 주소가 충돌을 일으키지 않는 경우 하나의 주소를 상기 제거 세트에 삽입한다. 한편, 컴퓨터 장치는 특정 트랜잭션을 수행하면서 남은 주소들 및 상기 주소들 중 동일한 L2 캐시 세트에 매핑되는 주소들에 접근한 후 상기 특정 트랜잭션이 폐기되지 않는다면 상기 남은 주소들이 충돌하지 않는다고 판단할 수 있다.
- [0085] 컴퓨터 장치는 생성한 제거 세트를 입력으로 이용하여 부채널 공격 과정을 수행한다. 컴퓨터 장치는 새로운 트랜잭션을 시작하고, 제거 세트에 포함된 각 주소에 접근하고 바쁜 대기 중 해당 트랜잭션이 폐기되면 레지스터에 저장된 값을 기준으로 폐기 원인을 체크한다(130).
- [0086] 컴퓨터 장치는 폐기 원인에 따라 부채널을 이용하여 유출된 정보를 확인한다(140). 컴퓨터 장치는 폐기 원인이 용량 오버플로우 또는 트랜잭셔널 충돌 중 어느 하나라면, 부채널을 이용하여 유출 정보를 확인할 수 있다.
- [0088] 또한, 상술한 바와 같은 CPU 공격 방법 내지 캐시 부채널 공격 방법은 컴퓨터에서 실행될 수 있는 실행가능한 알고리즘을 포함하는 프로그램(또는 어플리케이션)으로 구현될 수 있다. 상기 프로그램은 일시적 또는 비일시적 판독 가능 매체(non-transitory computer readable medium)에 저장되어 제공될 수 있다.
- [0089] 비일시적 판독 가능 매체란 레지스터, 캐쉬, 메모리 등과 같이 짧은 순간 동안 데이터를 저장하는 매체가 아니라 반영구적으로 데이터를 저장하며, 기기에 의해 판독(reading)이 가능한 매체를 의미한다. 구체적으로는, 상술한 다양한 어플리케이션 또는 프로그램들은 CD, DVD, 하드 디스크, 블루레이 디스크, USB, 메모리카드, ROM(read-only memory), PROM(programmable read only memory), EPROM(Erasable PROM, EPROM) 또는 EEPROM(Electrically EPROM) 또는 플래시 메모리 등과 같은 비일시적 판독 가능 매체에 저장되어 제공될 수 있다.

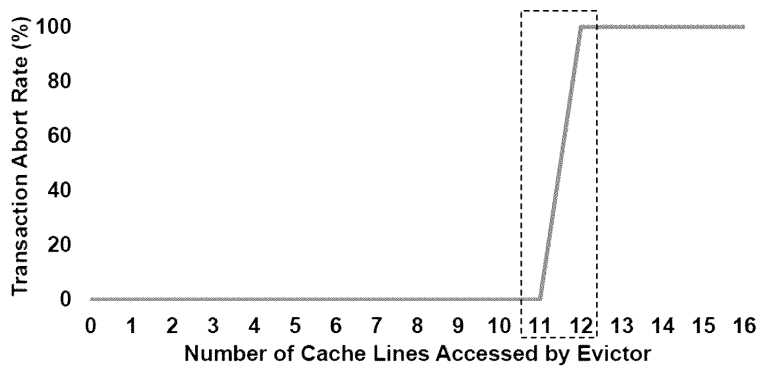
다.

[0090] 일시적 판독 가능 매체는 스태틱 램(Static RAM, SRAM), 다이내믹 램(Dynamic RAM, DRAM), 싱크로너스 디램(Synchronous DRAM, SDRAM), 2배속 SDRAM(Double Data Rate SDRAM, DDR SDRAM), 증강형 SDRAM(Enhanced SDRAM, ESDRAM), 동기화 DRAM(Synclink DRAM, SLD RAM) 및 직접 램버스 램(Direct Rambus RAM, DRRAM) 과 같은 다양한 RAM을 의미한다.

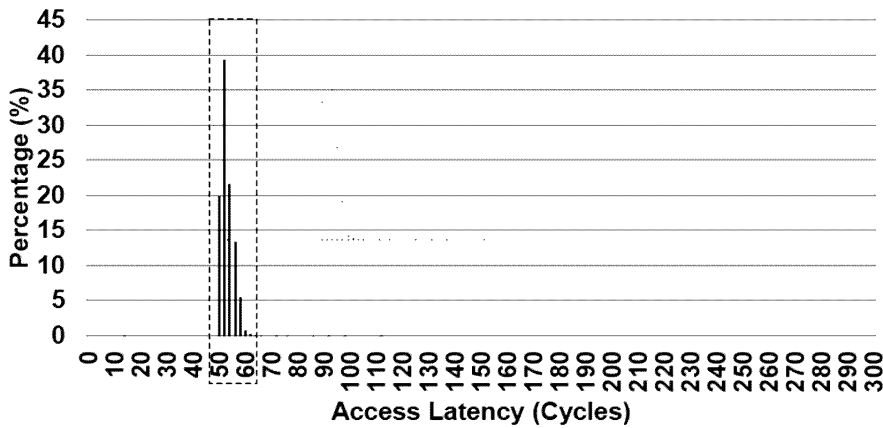
[0091] 본 실시예 및 본 명세서에 첨부된 도면은 전술한 기술에 포함되는 기술적 사상의 일부를 명확하게 나타내고 있는 것에 불과하며, 전술한 기술의 명세서 및 도면에 포함된 기술적 사상의 범위 내에서 당업자가 용이하게 유추할 수 있는 변형 예와 구체적인 실시례는 모두 전술한 기술의 권리범위에 포함되는 것이 자명하다고 할 것이다.

도면

도면1



도면2



도면3



도면4

100

