논문 2011-48CI-4-1

트랜잭셔널 메모리를 위한 효율적인 캐시 구조

(Efficient Cache Architecture for Transactional Memory)

최 동 민*, 김 승 훈*, 노 원 우**

(Dong Min Choi, Seung Hun Kim, and Won Woo Ro)

요 약

트랜잭셔널 메모리 시스템에서 오버플로우(overflow) 발생 시 이를 처리하기 위한 데이터의 기록은 그 복잡성으로 인해 전체 시스템 성능 저하의 주요 요인이 된다. 특히, 오버플로우 된 데이터가 일으킬 수 있는 충돌감지를 위해 캐시 일관성 프로토콜 상에 추가적인 상태 설정이 요구되며 이로 인해 트랜잭션간 커뮤니케이션에 지연이 발생한다. 이러한 문제점을 해결하기 위해 우리는 트랜잭셔널 메모리 시스템에서 오버플로우에 의해 발생하는 오버 헤드를 줄이기 위한 효율적인 캐시 구조를 연구하였다. 본 논문에서 제안하는 보조 캐시(supportive cache)는 1차 캐시와 동일한 교체 정책을 사용하며 병렬 룩업이 가능하도록 작동한다. 보조 캐시의 성능 평가를 위해 하드웨어 트랜잭셔널 메모리 시스템인 LogTM-SE를 사용하였으며 시뮬레이션 결과 평균적으로 37%의 성능 향상을 보였다.

Abstract

Traditional transactional memory systems are no longer able to guarantee the performance of diverse applications with overflowed transactions since there is the drawback that tracking the data for logging is difficult. Especially, this mechanism has a disadvantage of increasing communication delay for sustaining the state which is required to detect the conflict on the overflowed transactions from the first level cache in the transactional memory systems. To address this point, we have focused on the cache architecture of the systems to reduce the overhead caused by overflows and cache misses. In this paper, we present Supportive Cache which reduces additional overhead during transactions. Supportive Cache performs a parallel look-up with L1 private cache and uses the same replacement policy as L1 private cache. We evaluate the performance of the proposed design by comparing LogTM-SE with and without Supportive Cache. The simulation results show that our system improves the performance by 37% on average, compared to the original LogTM-SE which uses the same hardware resource.

Keywords : Transactional Memory, Parallel processing, Victim Cache, Supportive Cache

Ⅰ. 서 론

트랜잭셔널 메모리[1]는 다중 코어 시스템에서 많은 수의 쓰레드 간의 효율적인 병렬 처리를 지원한다. 이러한 트랜잭셔널 메모리의 이점은 dead lock 또는 live lock과 같은 lock 기반의 시스템에서 동기화로 인해 발생하는 문제들의 해결책을 제시하는 것이다. 배타적으로 공유 자원에 접근하는 lock 기반 시스템과 달리 트랜잭셔널 메모리 시스템은 일관성(consistency)과 원자성(atomicity) 그리고 격리(isolation)[2]의 특성을 유지하며 실행되어야 하며 이것을 가능하게 하기위해 3가지 주요 정책을 지켜야 한다. 이러한 정책에는 데이터 관리 정책, 충돌 감지 정책 그리고 충돌 해결 정책이 있으며 이를 구현하는 방식에 따라 하드웨어 트랜잭셔널 메모리(Haraware Transactional Memory; HTM)[3~5], 소프트웨어 트랜잭셔널 메모리(Software Transactional Memory; STM)[6~8], 하이브리드 트랜잭셔널 메모리(Hybrid Transactional Memory)[9~10] 형태로서 많은 선

* 학생회원, ** 정회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
※ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (2011-0005287)
접수일자: 2011년5월26일. 수정완료일: 2011년6월30일

행 연구가 진행된바 있다.

트랜잭셔널 메모리를 사용하는 복잡한 어플리케이션들은 프로그램 실행 중 잦은 충돌(conflict)을 일으키며 1차 캐시의 공간 부족으로 인한 오버플로우의 발생 빈도를 높인다. 특히, 트랜잭션 안에서 오버플로우 된 초기 데이터 값들을 기록하기 위해서 데이터 관리 정책은 다음 단계의 캐시 값을 추적해야 하는 복잡한 상황에 직면한다. 뿐만 아니라, 오버플로우 된 데이터가 일으킬 수 있는 충돌을 감지하기 위해 충돌감지 정책은 추가적인 상태를 캐시 일관성 프로토콜에 구현해야 한다. 이러한 복잡성으로 인하여 발생하는 지연은 성능의 저하에 큰 요인이 된다.

따라서 오버플로우가 자주 발생하는 트랜잭셔널 메모리 시스템은 위에서 언급한 정책들을 어떻게 설정하느냐와 관계없이 성능의 저하를 일으킨다. 이를 해결하기 위해서는 정책들과는 독립적으로 오버플로우의 원인이 되는 요인들을 제거하기 위한 접근이 필요하다. 그러므로 우리는 트랜잭셔널 메모리 시스템의 성능 결정에 또 다른 중요한 요인이 되는 오버플로우를 개선하기 위한 캐시구조를 제안한다.

본 논문에서 제안한 보조 캐시는 희생 캐시(victim cache)[11~13]와 유사하다. 하지만 희생 캐시는 작은 크기의 완전 연관 버퍼와 고정된 선입선출(FIFO) 교체정책을 사용하는 반면 보조 캐시는 1차 캐시와 같은 크기, 같은 연관도, 같은 교체정책을 사용한다. 이는 보조 캐시가 하나의 버퍼로서의 역할 뿐만 아니라 pseudo 캐시의 역할도 수행한다는 것을 의미한다. 결과적으로 보조 캐시는 뛰어난 지역성 및 pseudo 연관성의 특성을 가지며 이로 인해 희생 캐시 보다 오버플로우를 줄이는 효과가 더 우수하다.

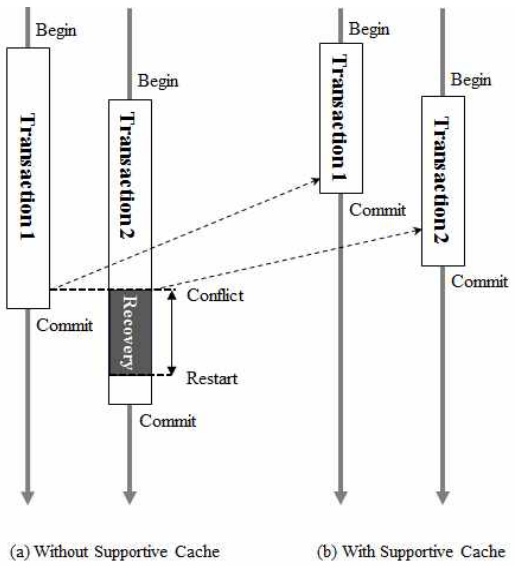

그림 1. 긴 트랜잭션에서 보조 캐시의 효과
Fig. 1. Effect of Supportive Cache in large transactions.

특히, 트랜잭셔널 메모리에서 사용되는 보조 캐시는 aborts 또는 stall에 의해 발생하는 메모리 사이클의 낭비를 줄일 뿐만 아니라 오버플로우 된 트랜잭셔널 데이터의 충돌 감지를 위해 추가된 sticky 상태를 유지하기 위한 커뮤니케이션 지연을 감소시킨다. 그러므로 전체 트랜잭션의 실행시간 및 충돌 발생 비율이 줄어들게 된다. 예를 들어, 그림 1-(a)에서 보는 것과 같이 트랜잭션 1 과 트랜잭션 2 사이에서 발생하는 충돌은 그림 1-(b)에서 보는 것과 같이 트랜잭션 실행시간이 감소함에 따라 제거되며 이로 인하여 트랜잭션 회복을 위해 소요되는 stall, abort, back-off와 같이 낭비되는 시간 또한 없어진다.

본 논문의 구성은 다음과 같다. 먼저, Ⅱ장에서 LogTM-SE[14]와 보조 캐시의 동작 및 구현에 대해 서술한다. Ⅲ장에서는 제안된 이론을 적용한 시뮬레이션 결과 및 분석에 대해 서술한다. 마지막으로 Ⅳ장에서는 결론에 대해 기술한다.

Ⅱ. 본 론

1. LogTM-SE

LogTM-SE[14]는 데이터 관리 정책을 위한 로그와 충돌 감지 정책을 위한 시그니처로 구성된다. 로그는 트랜잭션 구간에서 데이터가 변경되기 전의 초기 값들을 저장하는 공간으로 트랜잭션이 성공적으로 종료 될 경우 안에 있는 데이터를 제거하며 그렇지 않고 충돌이 발생할 경우 회복 메커니즘에 의해 데이터를 복원하는 동작을 수행하기 위해 사용된다. 시그니처는 load 명령어 수행 시 충돌을 감지하기 위한 read 시그니처와 store 명령어 수행 시 충돌을 감지하기 위한 write 시그니처가 있으며 MESI 디렉토리 프로토콜을 통한 시그니처 비교를 이용해 충돌을 감지한다.

특히, LogTM-SE의 충돌 정책은 오버플로우 된 데이터의 충돌 감지를 위해 sticky란 이름의 새로운 상태를 MESI 일관성 프로토콜에 추가하였다. Sticky 상태는 오버플로우 된 데이터가 마치 1차 캐시에 존재하는 것처럼 보이게 하며 프로세서가 해당 상태의 캐시 라인에 접근할 때 빠르게 충돌을 감지할 수 있도록 도와준

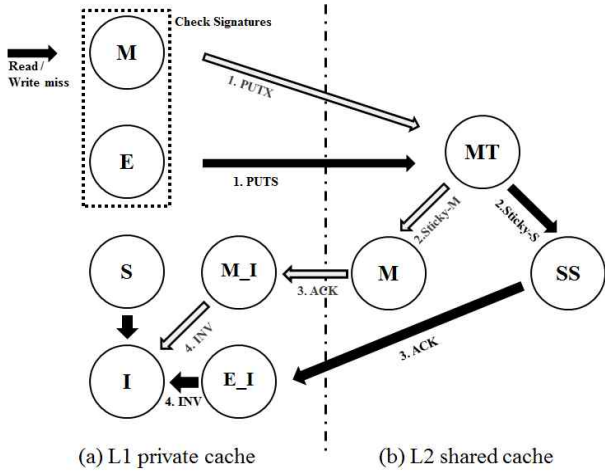

그림 2. LogTM-SE에서의 교체 상태 다이어그램
Fig. 2. Replacement states diagram in LogTM-SE.

다. Sticky 상태는 트랜잭션 안에서 다른 쓰레드에 의해 읽혀진 데이터에 대한 충돌 감지를 위한 sticky-S 상태와 다른 쓰레드에 의해 쓰인 데이터에 대한 충돌 감지를 위한 sticky-M 상태로 분류된다.

LogTM-SE에 적용된 일관성 프로토콜의 상태변화 다이어그램은 그림 2와 같다. 그림 2-(a)는 1차 캐시의 오리지널 상태(M :Modified, E :Exclusive, S :Shared, I :Invalid)와 과도 상태(M_I, E_I)를 보여주고 있다. M_I 상태는 1차 캐시에 있는 데이터가 M 상태에서 교체된 후에 2차 캐시로부터 ACK 신호를 기다리는 상태를 의미한다. 또한, E_I 상태는 E 상태에서 교체된 후 ACK 신호를 기다리는 상태를 의미한다. 그림 2-(b)에서는 MESI 프로토콜을 확장한 2차 캐시의 상태를 보여준다. MT 상태는 1차 캐시가 데이터를 변경하여 2차 캐시는 오래된 데이터를 가지고 있다는 것을 나타내는 상태이며 M 상태는 2차 캐시가 업데이트 되었으며 1차 캐시에는 해당 데이터가 없다는 것을 나타낸다. 그리고 SS 상태는 하나 이상의 1차 캐시에서 해당 데이터를 공유하고 있다는 것을 나타낸다.

그림 2의 상태 변화를 보면 다음과 같다. M 상태의 1차 캐시에서 오버플로우가 발생할 때 write 시그니처에 데이터의 주소가 있다면 추출되는 캐시 라인이 트랜잭션 상태에서 변경되었다는 것을 의미한다. 오버플로우가 트랜잭션 상태에서 발생하면 1차와 2차 캐시의 상태 변화는 밝은 화살표 방향으로 진행된다. 먼저, 트랜잭션 교체 이벤트는 1차와 2차 캐시의 데이터를 일치시키기 위해 PUTX 요청을 1차 캐시에서 2차 캐시로 보낸다. 그 다음 1차 캐시의 추출된 라인은 자신의 상태를 M 상태에서 M_I 상태로 변경하고 2차 캐시로부터 ACK 시그널을 기다린다. PUTX 요청을 받은 2차 캐시는 캐시 라인의 상태를 MT 상태에서 M 상태로 변경하고 sticky-M 상태를 디렉터리에 표시한 후 1차 캐시에게 ACK 시그널을 보낸다. 그 후 1차 캐시가 ACK 시그널을 받으면 자신의 상태를 M_I 상태에서 I 상태로 변경한다.

이와 유사하게 E 상태의 1차 캐시에서 오버플로우가 발생할 때 read 시그니처에 데이터의 주소가 있다면 추출되는 캐시 라인은 트랜잭션 상태에서 읽혀진 라인이므로 다른 쓰레드에서 해당 라인의 데이터를 변경할 수 없다. 이와 같은 트랜잭션 상태의 변화는 어두운 화살표 방향으로 진행된다. 먼저, 트랜잭션 교체 이벤트는 PUTS 요청을 1차 캐시에서 2차 캐시로 보낸 후 자신의 상태를 E상태에서 E_I 상태로 변경한다. 그 다음 2차 캐시는 PUTS 요청을 받은 후 자신의 상태를 MT 상태에서 SS 상태로 변경하고 sticky-S 상태를 디렉터리에 표시한 후 ACK 시그널을 1차 캐시로 보낸다. ACK 시그널을 받은 1차 캐시는 자신의 상태를 I 상태로 변경한다. 위와 같이 sticky 상태를 위해 추가적인 상태 및 시그널이 존재하며 이것은 캐시 사이의 커뮤니케이션 지연을 야기한다.

2. 보조 캐시(supportive cache)의 동작 및 구현

LogTM-SE와 함께 동작하는 보조 캐시는 1차 캐시와 같은 연관도 및 교체 정책을 사용한다. 보조 캐시를 사용할 경우 1차 캐시의 교체 이벤트로 인해 발생하는 PUTS 혹은 PUTX 요청은 2차 캐시의 상태를 변경하는 대신 보조 캐시의 상태를 변경하게 된다. 해당 요청을 받은 보조 캐시는 1차 캐시의 상태를 그대로 승계한 후 1차 캐시의 상태를 I 상태로 변경한다. 따라서 2차 캐시는 자신의 상태를 그대로 유지하게 되며 교체 이벤트로 인한 1차 캐시의 M_I 혹은 E_I 상태 와 2차 캐시의 M 혹은 SS 상태로의 전이는 발생하지 않는다.

이와 같이 보조 캐시는 1차 캐시와 같은 역할을 수행하는 추가적인 캐시이다. 하지만 두 개의 1차 캐시와 함께 하는 시스템과는 다르게 동작한다. 먼저 보조 캐시는 1차 캐시가 두 개로 나눠진 형태로 구성되며 같은 세트의 데이터를 저장한다. 즉 캐시의 세트가 증가되는 것이 아니라 오버플로우된 데이터를 저장하는 공간이 증가하며 이는 캐시의 way수를 증가시키는 효과가 있다. 따라서 트랜잭션에서 사용되는 보조 캐시의 동작은

크게 교체 동작, load와 store 동작, 그리고 commit과 abort 동작에 영향을 미친다.

가. 교체 동작 (Replacement Operation)

그림 3-(a)는 LogTM-SE에서 추출된 캐시 라인이 어떻게 교체되는지를 보여준다. 트랜잭션 상태에서 특정 주소가 store 동작을 수행하기 위해 캐시에 접근했을 때 1차 캐시의 모든 라인이 이용가능하지 않다면 L1_replacment_XACT 이벤트가 발생하고 해당 이벤트는 오버플로우 비트를 설정함과 동시에 L1_PUTX 이벤트를 발생시킨다. 이 이벤트는 1차 캐시에서 2차 캐시로 메시지를 보내기 위해 발생하는 이벤트이다. 이벤트를 받은 후 LRU 교체 정책에 의해 선택된 1차 캐시 라인의 데이터는 라이트-백 동작을 통해 2차 캐시에 저장된다. 2차 캐시가 업데이트 된 후 디렉터리는 추출된 캐시 라인의 상태를 M 상태에서 sticky-M 상태로 변경한다. 디렉터리의 업데이트가 완료된 후 2차 캐시는 라이트-백 동작과 sticky 상태 변경 작업이 완료 되었다는 것을 알리기 위해 WB_ACK 시그널을 1차 캐시로 보낸다.

그림 3-(b)는 같은 상황에서 보조 캐시가 어떻게 동작하는지를 나타낸다. 트랜잭션 상태에서 특정 주소가 store 동작을 수행하기 위해 캐시에 접근했을 때 캐시에 여유 공간이 없다면 기존의 L1_replacment_XACT

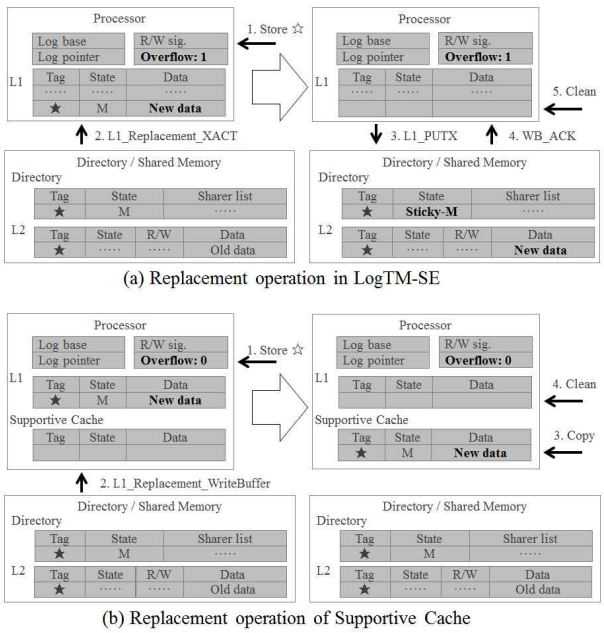

그림 3. LogTM-SE에서의 보조 캐시 교체 동작
Fig. 3. Supportive cache replacement operation.

이벤트 대신 L1_replacment_WriteBuffer 이벤트가 발생한다. 이벤트 발생 후 오버플로우 비트를 설정함과 동시에 PUTX_WRITEBUFFER 요청이 생성되며 해당 요청은 1차 캐시의 데이터와 상태정보를 보조 캐시에 복사하는 동작을 수행한다. 복사 동작이 완료 되면 1차 캐시의 추출된 캐시라인의 사용이 가능해 진다. 결과적으로 보조 캐시는 L1_PUTX 이벤트와 WB_ACK 시그널로 인해 발생하는 1차 캐시와 2차 캐시의 접근을 줄일 수 있다.

나. Load/Store 동작 (Load/Store Operation)

만약 트랜잭션 상태에서 load 혹은 store 동작 중에 1차 캐시에서 캐시 미스가 발생하면 프로세서는 GETS 혹은 GETX 메시지를 디렉터리로 보내 해당 데이터를 요청하고 디렉터리는 FWD_GETS 혹은 FWD_GETX 메시지를 다른 프로세서들에게 보내 해당 데이터의 사용 유무를 확인한다. FWD 메시지를 받은 프로세서는 데이터의 유무를 판별하고 데이터가 존재하면 시그니처를 이용하여 충돌 감지를 확인한다. 만약 충돌이 존재하면 back-off와 같은 충돌 해결 정책에 따라 회복 메커니즘이 수행된다. 반면 다른 프로세스들 모두 데이터가 존재하지 않거나 충돌이 발생하지 않으면 디렉터리는 데이터를 요청한 프로세서에게 해당 데이터의 사용이 가능하다는 것을 알린다.

보조 캐시를 사용할 경우 load 혹은 store 동작은 pseudo 연관 캐시를 사용하는 것과 유사하게 동작한다. 1차 캐시에서 캐시 미스가 발생하면 프로세서는 디렉터리로 FWD 메시지를 보내는 대신 보조 캐시에 데이터가 존재하는지를 판별한다. 보조 캐시에서 캐시 미스가 발생 할 경우 보조 캐시가 디렉터리에 FWD 메시지를 보내게 된다. 따라서 1차 캐시에서 추출된 데이터를 가지고 있는 보조 캐시는 지역성의 특성을 높여주어 캐시 미스 율을 감소시키는 효과가 있다.

다. Commit/Abort 동작 (Commit/Abort Operation)

LogTM-SE의 경우 트랜잭션 상태에서 새로운 값을 캐시에 바로 업데이트하기 때문에 commit시에 빠르게 수행된다. 하지만 오버플로우가 많이 발생한 트랜잭션 상태에서 abort가 발생할 경우 충돌 해결 정책은 회복 메커니즘을 통해 sticky상태에 있는 캐시 라인들을 확인하기 위해 2차 캐시에 접근해야한다. 이와 같은 경우 보조 캐시는 sticky상태를 제거하는 효과가 있기 때문

에 2차 캐시의 접근을 상당부분 줄여 준다. 따라서 abort가 발생할 경우 보조 캐시의 사용은 성능 향상에 많은 도움을 준다.

라. 보조 캐시의 구현

보조 캐시는 GEMS[15]에서 MESI 일관성 프로토콜을 나타내기 위한 스크립트 언어인 SLICC (Specification Language for Implementing Cache Coherence)를 이용하여 구현하였다. SLICC는 캐시 메모리와 디렉터리 컴포넌트로 구성되어 있으며 각 컴포넌트들은 상태 머신, 이벤트, 액션을 포함한다. 각 상태 머신은 메시지가 도착했을 때 해당 상태에 의해 생성되어 지는 이벤트를 이용하여 통신을 한다. 보조 캐시 구현을 위해 SLICC에 보조 캐시의 컴포넌트 들을 추가하였으며 1차 캐시, 2차 캐시 및 디렉터리와 모두 통신이 가능하도록 이벤트를 추가하였다.

Ⅲ. 실 험 결 과

본 절에서는 시뮬레이션의 결과 및 분석에 대해 서술한다. 시뮬레이션은 위스콘신에서 제공하는 GEMS[15]와 SIMICS[16]를 통해 수행하였으며 사용한 코어의 수는 16개이다. 보조 캐시의 성능을 평가하기위해 트랜잭셔널 메모리의 성능 평가에 많이 사용되는 STAMP[17] 벤치마크를 이용하였다.

오버플로우는 캐시 세트의 부족(capacity overflow)으로 발생하는 것 이외에도 스레싱(thrashing)을 야기하는 연관성(associativity)의 부족으로도 발생한다. LogTM-SE와 보조 캐시는 기본적으로 4-way 연관 캐시를 사용한다. 그러므로 우리는 증가된 캐시의 연관성과 보조 캐시를 비교하기 위해 4-way 연관 캐시와 8-way 연관 캐시로 구성된 LogTM-SE를 기반으로 보조 캐시의 성능을 평가하였다. 두 캐시의 크기를 같게 하기 위해 8-way 연관 캐시는 4-way 연관 캐시의 절반의 세트를 가지도록 구성하였다.

그림 4는 STAMP의 정규화 된 라이트-백을 나타낸다. 트랜잭션 구간에서 오버플로우가 발생할 경우 캐시의 상태가 M이나 E일 경우에만 sticky 상태를 유지하게 된다. 즉, 라이트-백이 일어날 경우에만 오버플로우 된 트랜잭션을 추적해야하고 그렇지 않고 캐시의 상태가 S 상태일 경우에는 해당 라인의 데이터를 비활성화 시키는 동작만 하게 된다. 그림에서 보듯이 bayes, intruder, labyrinth와 같이 트랜잭션 구간이 많은 부분을 차지하는 어플리케이션에서 라이트-백이 크게 줄어드는 것을 확인할 수 있다.

그림 5는 트랜잭션 구간에서의 일반화된 시간의 분배를 나타낸다. 트랜잭션 구간의 실행 시간은 총 7가지로 분류된다.(stall, barrier, back-off, aborted, aborting, good transaction, non transaction) 또한 그래프의 L은 LogTM-SE를 의미하고 8Way_L은 8-way 연관 캐시로 구성된 LogTM-SE를 의미하며 SC는 보조 캐시를 의미한다. 트랜잭션의 실행시간 중에 stall, back-off, aborted, aborting 사이클은 트랜잭션 충돌로 인해 야기되는 낭비되는 사이클이며 good transaction 사이클은 낭비되는 것을 제외한 트랜잭션 수행 사이클을 의미한다. 그리고 non transaction 사이클은 트랜잭션 구간 밖에서 실행되는 시간을 의미한다.

또한 우리는 특정 사이클이 공통적으로 큰 비중을 차지하는 어플리케이션 별로 분류하였다. Bayes와 intruder는 다른 어플리케이션에 비해 back-off 사이클이 큰 비중을 차지하며 전체 사이클중 위에서 언급한 낭비되는 사이클의 비중이 72%를 차지한다. Genome과 vacation은 다른 어플리케이션에 비해 good transaction 사이클이 큰 비중을 차지하며 낭비되는 사이클의 비중

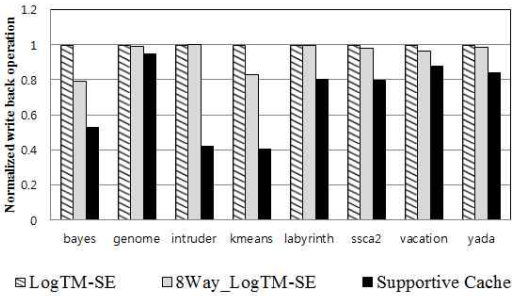

그림 4. STAMP의 라이트-백 카운트
Fig. 4. Write back count of applications.

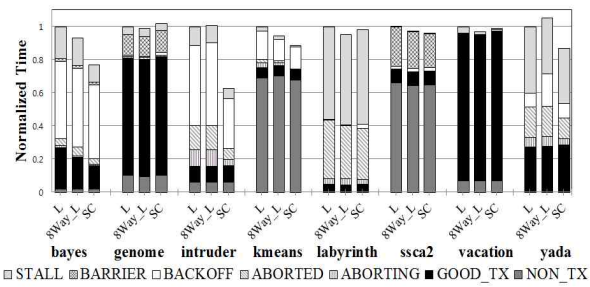

그림 5. STAMP의 시간 분배 분석
Fig. 5. Normalized distribution time of applications.

은 작다. Kmeans와 ssca2는 non transaction 사이클이 큰 비중을 차지하며 낭비되는 사이클의 비중은 25%정도이다. Labyrinth와 yada는 stall과 aborted의 사이클이 큰 비중을 차지하며 낭비되는 사이클의 비중은 80%를 차지한다.

보조 캐시를 사용하는 경우 낭비되는 사이클이 크게 줄어드는 것을 확인하였으며 전체 어플리케이션에서 평균적으로 35%가 개선되는 것을 알 수 있다. 또한 good transaction과 non transaction의 사이클이 큰 비중을 차지하는 어플리케이션은 트랜잭션 구간에서 실행할 때 충돌이 많이 발생하지 않는 다는 것을 알 수 있다. 결과적으로 낭비 사이클의 비중이 큰 bayes와 intruder가 보조 캐시의 사용으로 트랜잭션의 성능을 크게 향상 시킬 수 있다는 것을 알 수 있다.

그림 6은 트랜잭션 구간에서의 abort 카운트를 나타낸다. Abort 카운트의 감소는 보조 캐시의 사용으로 트랜잭션의 충돌 횟수가 감소된다는 것을 의미한다. 그림 4 와 그림 5에서 확인하였듯이 보조 캐시는 트랜잭션 구간에서의 오버플로우를 효율적으로 감소시키고 이로 인하여 충돌 횟수를 줄이며 그에 따르는 낭비되는 사이클을 줄일 수 있다는 것을 확인하였다.

그림 7은 STAMP의 정규화 된 전체 실행 시간을 나타낸다. 보조 캐시의 사용으로 bayes는 2배 이상의 실행 시간 단축을 확인 할 수 있다. 또한 intruder는 42%의 실행 시간 단축을 가져온다. Genome의 경우는 트랜잭션 구간에서의 충돌이 많이 발생하지 않기 때문에 보조 캐시의 사용으로 효과를 볼 수 없다는 것도 알 수 있다. 전체적으로 모든 어플리케이션의 실행시간이 단축되며 보조 캐시의 사용으로 평균적으로 37%의 성능 향상을 얻을 수 있다.

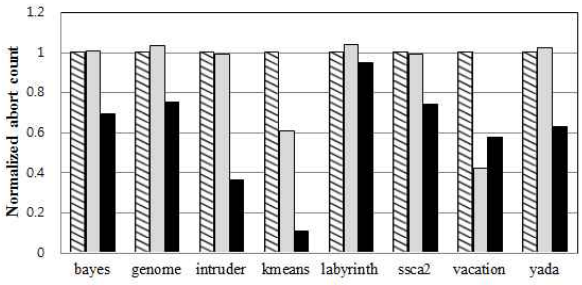

그림 6. STAMP의 abort 카운트
Fig. 6. Abort count of applications.

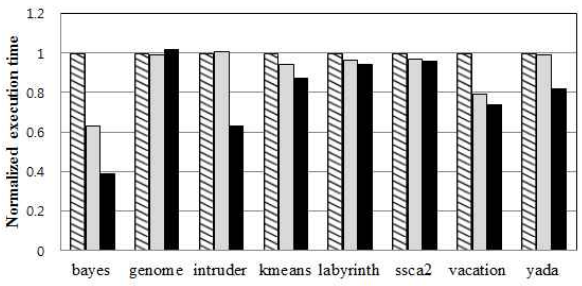

그림 7. STAMP의 전체 실행시간 분석
Fig. 7. Normalized overall execution time of applications.

Ⅳ. 결 론

본 연구를 통해 트랜잭셔널 메모리에서 오버플로우가 성능에 미치는 영향을 파악할 수 있었다. 트랜잭션 구간에서의 오버플로우는 충돌 감지를 위해 추가적인 상태가 필요하고 이 상태로 인해 발생하는 낭비되는 사이클은 성능에 좋지 않은 영향을 준다는 것도 확인하였다. 따라서 트랜잭셔널 메모리를 위한 보조 캐시의 사용은 오버플로우 발생 빈도를 줄여 트랜잭션의 빠른 실행을 가능하게 한다. 보조 캐시는 희생 캐시의 장점뿐만 아니라 pseudo 캐시의 장점도 가지고 있어 높은 지역성 및 연관성을 보인다. 연구 결과를 통해 알 수 있듯 보조 캐시의 사용으로 오버플로우의 발생 빈도를 줄이고 이에 따라 충돌 횟수와 낭비되는 사이클을 줄어드는 것을 확인하였다.

참 고 문 헌

[1] M. Herlihy and J. Moss. Transactional memory: Architectural support for lock-free data structures. In Proceedings of the 20th Annual International Symposium on Computer Architecture, pages 289 - 300, May 1993.

[2] L. Hammond, V. Wong, M. Chen, B. Carlstrom, J. Davis, B. Hertzberg, M. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun. Transactional memory coherence and consistency. In Proceedings of the 31st International Symposium on Computer Architecture, pages 102 - 113, June 2004.

[3] K. Moore, J. Bobba, M. Moravan, M. Hill, and D. Wood. Logtm; log-based transactional memory. In Proceedings of the 12th International Symposium on High-Performance Computer Architecture, pages 254 - 265, February 2006.

[4] M. Lupon, G. Magklis, and A. Gonzalez. Fastm;

a log-based hardware transactional memory with fast abort recovery. In Proceedings of the 18th International Conference on Parallel Architecture and Compilation Techniques, pages 293 - 302, September 2009.

[5] 김승훈, 김선우, 노원우, "집중 충돌 병렬 처리를 위한 효율적인 다중 코어 트랜잭셔널 메모리," 전자공학회논문지, 제48권 CI편, 제1호, 72~79쪽, 2011년 1월.

[6] V. J. Marathe, W. N. S. III, and M. L. Scott. Adaptive software transactional memory. In Proceedings of the 19th International Symposium on Distributed Computing, pages 354 - 368, 2005.

[7] B. Saha, A.-R. Adl-Tabatabai, R. L. Hudson, C. C. Minh, and B. Hertzberg. Mcrtstm; a high performance software transactional memory system for a multi-core runtime. In Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 187 - 197, 2006.

[8] M. F. Spear, L. Dalessandro, V. J. Marathe, and M. L. Scott. A comprehensive strategy for contention management in software transactional memory. In Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 141 - 150, February 2009.

[9] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum. Hybrid transactional memory. In Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 336 - 346, October 2006.

[10] A. Shriraman, M. F. Spear, H. Hossain, S. Dwarkadas, and M. L. Scott. An integrated hardware-software approach to flexible transactional memory. In Proceedings of the 34th International Symposium on Computer Architecture, pages 104 - 115, 2007.

[11] R. I. Bahar, D. Grunwald, and B. Calder. A comparison of software code reordering and victim buffers, March 1999.

[12] B. M. Beckmann, M. R. Marty, and D. A. Wood. Asr: Adaptive selective replication for cmp caches. In Proceedings of the 39th Annual International Symposium on Microarchitecture, pages 443 - 454, December 2006.

[13] M. Qureshi. Adaptive spill-receive for robust high-performance caching in cmps. In Proceedings of the 15th International Symposium on High-Performance Computer Architecture, pages 45 - 54, February 2009.

[14] L. Yen, J. Bobba, M. Marty, K. Moore, H. Volos, M. Hill, M. Swift, and D. Wood. Logtm-se; decoupling hardware transactional memory from caches. In Proceedings of the 13th International Symposium on High-Performance Computer Architecture, pages 261 - 272, February 2007.

[15] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. ACM SIGARCH Computer Architecture News, 33(4):92 - 99, November 2005.

[16] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. IEEE Computer, 35(2):50 - 58, February 2002.

[17] C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. Stamp; stanford transactional applications for multi-processing. In Proceedings of the IEEE International Symposium on Workload Characterization, pages 35 - 46, September 2008.

저 자 소 개

최 동 민(학생회원)
2004년 연세대학교 의용전자 공학과 졸업 (학사)
2004년 7월 삼성전자 무선사업부 6그룹 S/W Lab1.
2010년～현재 삼성전자 무선사업부 Global GSM S/W 개발그룹 선임연구원
2010년 연세대학교 전기전자공학과 입학 (석사과정)
<주관심분야 : 컴퓨터 시스템, 트랜잭셔널 메모리 등>

노 원 우(정회원)
1996년 연세대학교 전기공학과 졸업(학사)
1999년 University of Southern California 졸업(석사)
2004년 University of Southern California 졸업(공학박사)
2003년～2004년 Apple Computer Inc. 인턴 연구원
2004년～2007년 California State University 전기 및 컴퓨터공학과 조교수
2006년～2007년 ARM Inc. 소프트웨어 엔지니어
2007년～현재 연세대학교 전기전자공학과 조교수
<주관심분야 : 고성능 마이크로프로세서 디자인, 컴파일러 최적화, 임베디드 시스템 디자인 등>

김 승 훈(학생회원)
2009년 연세대학교 전기전자 공학과 졸업 (학사)
2011년 연세대학교 전기전자 공학과 졸업 (석사)
2011년 연세대학교 전기전자 공학과 입학 (박사과정)
<주관심분야 : 컴퓨터 시스템, 트랜잭셔널 메모리 등>