멀티코어 환경에서 효율적인 트랜잭션 처리를 위한 메모리 관리 기반 하이브리드 트랜잭셔널 메모리 기법*

장연우, 강문환, 장재우†
전북대학교 컴퓨터공학과
{kp7050, calvin, jwchang}@jbnu.ac.kr
†Corresponding author

Memory Management based Hybrid Transactional Memory Scheme for Efficiently Processing Transactions in Multi-core Environment

Yeon-Woo Jang, Moon-Hwan Kang, Jae-Woo Chang†
Dept of Computer Engineering, Chon-Buk University

요 약

최근 멀티코어 프로세서가 개발됨에 따라 병렬 프로그래밍은 멀티코어를 효과적으로 활용하기 위한 기법으로 그 중요성이 높아지고 있다. 트랜잭셔널 메모리는 처리 방식에 따라 HTM, STM, HyTM으로 구분되며, 최근 HTM 및 STM 결합한 HyTM 이 활발히 연구되고 있다. 그러나 기존의 HyTM 는 HTM과 STM의 동시성 제어를 위해 블룸필터를 사용하는 반면, 블룸필터의 자체적인 긍정 오류를 해결하지 못한다. 아울러, 트랜잭션 처리를 위한 메모리 할당/해제를 기존의 락 메커니즘을 사용하여 관리한다. 따라서 멀티코어 환경에서 스레드 수가 증가할수록 트랜잭션 처리 효율이 떨어진다. 본 논문에서는 멀티코어 환경에서 효율적인 트랜잭션 처리를 위한 메모리 관리 기반 하이브리드 트랜잭셔널 메모리 기법을 제안한다. 제안하는 기법은 트랜잭션 처리에 최적화된 블룸필터를 제공함으로써, 병렬적으로 동시에 수행되는 서로 다른 환경의 트랜잭션에 대해 일관성 있는 처리를 지원한다. 아울러, CPU 캐시 라인에 최적화된 메모리 기법을 통해, 메모리 할당량이 적은 트랜잭션은 로컬 캐시에 할당함으로써 트랜잭션의 빠른 처리를 지원한다.

1. 서론

전통적인 락 기법의 문제점을 해결하기 위해 개발된 트랜잭셔널 메모리(transactional memory; TM)는 병렬 프로그래밍의 패러다임을 바꾸었다. TM을 이용한 병렬 프로그래밍에서는 락을 사용하지 않으며 트랜잭션의 시작과 끝에 일련의 코드를 작성하는 것으로 스레드들 간의 데이터 동기화 문제를 해결한다.

TM은 구현하는 방법에 따라 크게 하드웨어 트랜잭셔널 메모리(Hardware TM;HTM), 소프트웨어 트랜잭셔널 메모리(Software TM;STM) 그리고 두 가지 방법을 결합한 하이브리드 트랜잭셔널 메모리(Hybrid TM;HyTM)로 분류된다. HTM 시스템은 TM의 주요 기능이 하드웨어로 구현되어 있기 때문에 효율적으로 설계된 락 기반의 병렬 프로그래밍과 동일한 정도의 성능을 달성할 수 있다. 하지만 HTM은 하드웨어를 이용하기 때문에 L1캐시 용량초과 문제 및 하드웨어를 제어하는 OS에 의한 성능 영향을 많이 받는다. 반면 STM 시스템은 컴파일러와 API 형태로 TM의 주요 기능들을 구현되어 있기 때문에 프로그래머가 트랜잭션 처리에 필요한 다양한 프로토콜을 추가로 구현할 수 있으며 처리된 트랜잭션의 결과를 피드백 할 수 있다. 하지만 STM은 자체적인 연산 오버헤드 및 트랜잭션 결과 피드백에 의한 오버헤드로 인해 HTM 시스템에 비해 낮은 성능을 보인다. 최근, HyTM 시스템은 HTM과 STM의 장점을 결합하여 구현한 TM으로 HTM으로 수행되지 못한 트랜잭션을 STM으로 수행함으로써 STM의 유연성과 HTM의 속도를 활용할 수 있는 장점이 존재한다. 그러나 기존 HyTM 는 HTM과 STM의 동시성 제어를 위해 블룸필터를 사용하는 반면, 블룸필터의 자체적인 긍정 오류를 해결하지 못한다. 아울러, 트랜잭션 처리를 위한 메모리 할당/해제를 기존의 락 메커니즘을 사용하여 관리한다. 따라서 멀티코어 환경에서 스레드 수가 증가할수록 트랜잭션 처리 효율이 떨어지는 문제점이 존재한다.

본 논문에서는 멀티코어 환경에서 효율적인 트랜잭션 처리를 위한 메모리 관리 기반 하이브리드 트랜잭셔널 메모리 기법을 제안한다. 제안하는 기법은 HTM 및 가장 뛰

*이 논문은 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음(IITP-2016-R0113-15-0005). 또한 이 논문은 2016년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임 (NRF-2016H1C1A1065816)

어난 성능을 보이는 STM 기법인 NorecSTM[1] 간의 동시성 제어를 위해 효율적인 블룸 필터를 제공함으로써, 병렬적으로 동시에 수행되는 서로 다른 환경의 트랜잭션에 대해 일관성 있는 처리를 지원한다. 아울러, CPU 캐시 라인에 최적화된 메모리 기법을 통해, 메모리 할당량이 적은 트랜잭션은 로컬 캐시에 할당함으로써 트랜잭션의 빠른 처리를 지원한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 기술하고, 3장에서는 제안하는 메모리 관리자 기반인 하이브리드 트랜잭셔널 메모리 기법을 기술한다. 4장에서는 제안하는 기법의 우수성을 검증하기 위해 성능 평가를 수행한다. 5장에서는 결론 및 향후 연구에 대해 기술한다.

2. 관련 연구

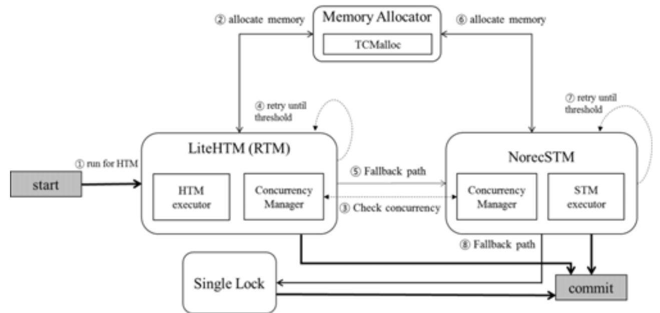
하이브리드 트랜잭셔널 메모리(HyTM)기법은 크기가 작은 트랜잭션은 HTM으로 처리하며 HTM으로 처리될 수 없는 큰 트랜잭션의 경우에는 fallback-path로 STM 기법을 사용하여 유연성을 확대한 기법이다. L. Dalessandro et al.의 연구[2]는 가장 뛰어난 성능을 보이는 NOrecSTM[1] 및 HTM을 결합한 Hybrid NOrec를 제안하였다. 해당 연구는 HTM과 STM을 병렬적으로 수행하며 서로 다른 기법의 동시성 제어를 위해 스냅샷 기반의 락 프리 기법을 사용하였다. HTM이 STM에 비해 처리 속도가 매우 빠르기 때문에 병렬 수행에 있어 일관성 제어가 매우 어렵다는 문제가 발생한다. 해당 연구는 이를 위해 동시성 제어에 필요한 모든 메타데이터를 STM에서 유지하여 HTM에 불필요한 연산을 수행하지 않도록 하였다. 또한 HTM에 처리 우선순위를 두어 HTM과 STM간에 충돌 발생 시, HTM을 commit 시키고 STM을 강제로 중단시킨다.

그러나 해당 연구는 모든 트랜잭션을 Single Global Lock 없이 HTM 및 STM으로 처리하기 때문에 지속적으로 충돌이 발생하는 경우에 starvation 현상이 발생할 수 있다. 또한 동시성 제어에 블룸필터를 사용하는 반면, 블룸필터가 갖는 긍정 오류를 해결하지 못한다. 아울러, 해당 연구는 트랜잭션에 최적화된 메모리 관리 기법을 사용하지 않기 때문에, 멀티 코어 환경에서 스레드의 수가 증가할수록 트랜잭션 처리 효율이 떨어지는 문제점이 발생한다.

3. 제안하는 메모리 관리 기반 하이브리드 트랜잭셔널 메모리 기법

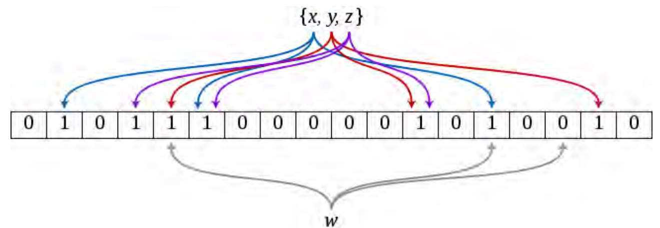
3.1 제안하는 하이브리드 트랜잭셔널 메모리 기법의 전체 구조

제안하는 기법의 전체 흐름은 <그림 1>과 같다. ① 워크로드부터 트랜잭션이 시작되면, HTM executor를 통해 HTM 시작을 준비한다. ② Memory Allocator 컴포넌트를 통해 필요한 메모리를 할당받는다. ③ 만약 STM 기법을 통해 트랜잭션이 수행 중이라면 Concurrency Manager를 통해 HTM과 STM의 동시성 제어를 수행한다. ④ 트랜잭션 처리가 실패되면 정의된 재시도 횟수만큼 HTM을 재시작한다. ⑤ HTM으로 처리될 수 없는 경우, fallback 경로를 이용하여 STM 기법으로 수행한다. ⑥ Memory Allocator 컴포넌트를 통해 필요한 메모리를 할당받는다. ⑦ 트랜잭션 처리가 실패되면 정의된 재시도 횟수만큼 STM을 재시작한다. ⑧ 만약 TM으로 처리될 수 없는 경우, 안전성이 보장된 single global lock으로 재수행한다.



(그림 1) 제안하는 하이브리드 트랜잭셔널 메모리 기법의 전체 흐름

3.2 제안하는 블룸필터 기법



(그림 2) 블룸 필터 구조

블룸 필터(Bloom Filter)는 원소가 집합에 속하는지 여부를 검사하는데 사용되는 확률적 자료 구조이며 이는 <그림 2>와 같다. 블룸 필터는 결과의 신뢰성에 치명적인 영향을 미치는 부정 오류(false negative)가 발생되지 않는다. 하지만 긍정 오류(false positive)가 발생할 가능성이 존재하기 때문에, 트랜잭션 처리 효율을 증대하기 위해서는 블룸 필터의 긍정 오류를 줄이는 것이 중요하다. 이를 해결하기 위해 제안하는 기법은 긍정 확률을 줄이는 식(1)을 적용하여 조절한다. 식(1)에서 k는 해시 함수의 개수이고, n은 집합의 크기, m은 bloom filter의 bit 사이즈이다. 예를 들어, 10,000개의 트랜잭션 연산이 존재한다고 가정하고, 1%의 error rate를 요구할 경우 필터의 크기는 <식 2>과 같이 계산된다.

$$False\ Positive\ rate \approx \left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-\frac{kn}{m}}\right)^{k}$$

$$Optimal\ of\ hash\ functions\ (k) \approx \ln 2\frac{m}{n} \approx 0.7\frac{m}{n}$$

---- (식 1)

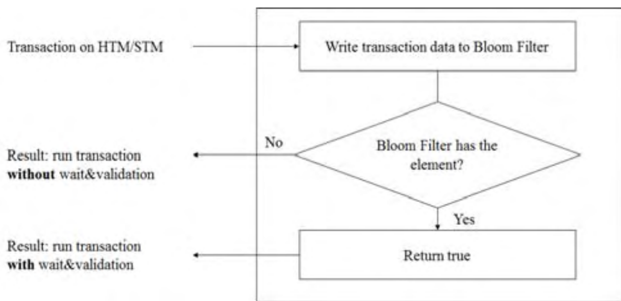
$$10{,}000\,operations, 1\%\,error\ rate, bit\ size = 10$$

$$m = 10{,}000x\,10bits = 12kb\ of\ memory$$

$$k = 0.7\times\frac{120{,}000}{10{,}000} = 7\ hash\ functions$$

----- (식 2)

블룸 필터 기반 동시성 제어 흐름은 <그림 3>과 같다. 첫째, HTM 혹은 STM에서 처리되는 모든 트랜잭션의 데이터를 블룸 필터에 기록한다. 둘째, 블룸필터를 통해 병렬적으로 수행되고 있는 트랜잭션 사이에 충돌이 발생하는 지 검사한다. 셋째, 충돌이 없는 경우 대기 및 검증 단계를 생략하고 commit을 수행한다. 만약 충돌이 발생한 경우, 충돌한 트랜잭션을 대기상태로 두고, 검증 단계를 수행해서 commit 여부를 결정한다.

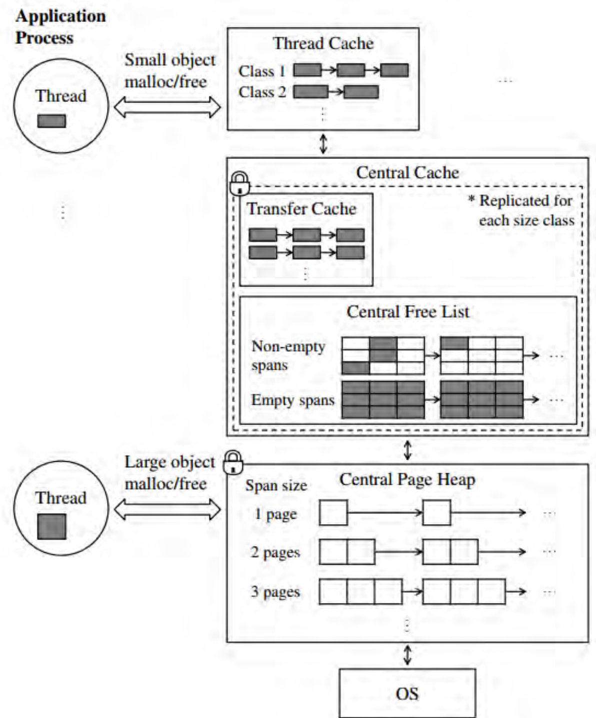


(그림 3) 블룸 필터 기반 동시성 제어 흐름

3.3 제안하는 메모리 관리자 기법

기존의 단일 코어 환경에서 메모리 관리자는 공유 메모리에 하나의 메모리 풀을 사용하여 메모리를 할당한다. 반면, 멀티 코어 환경에서는 여러 스레드에서 하나의 메모리 풀을 사용하기 때문에 데이터 충돌을 방지하기 위해 락을 사용한다. 하지만 데이터 충돌 외에도 스레드 대기시간, 스레드 우선순위 등 다양한 문제가 발생하기 때문에 최근에는 멀티 코어 환경에서의 효율적인 메모리 관리 기술에 대한 연구[3]가 활발히 진행되고 있다. 제안하는 기법은 메모리 블록의 크기가 커질수록 우수한 성능을 보이는 TCMalloc 기반 메모리 관리자[4]를 사용한다.

TCMalloc은 멀티 코어 환경에서 효과적으로 메모리 풀을 관리하는 Lock-free 기반 메모리 관리자이며, 오브젝트의 크기에 따라 중앙 캐시와 로컬 캐시로 나누어 할당 및 해제를 수행한다. TCMalloc의 전체 구조는 <그림 4>와 같다. 32K 이하의 작은 오브젝트를 처리하기 위해 각 스레드가 사용할 로컬 캐시를 할당한다. 로컬 캐시는 Cental Page Heap에 비해 작은 영역으로 할당되고 지역적으로 동작하기 때문에 효과적으로 동시성 제어를 수행할 수 있다. 반면 32K 이상의 큰 오브젝트인 경우 free list들의 배열로 이루어진 Central Page Heap에서 직접 관리한다. Cental Page Heap은 256개의 연속된 페이지로 이루어져 있으며, 연속된 페이지들은 각각 같은 크기를 가진 일련의 작은 오브젝트들로 나누어진다. 예를 들어 한 페이지(4K)는 각각 128바이트의 32개의 객체로 나뉠 수 있다.



(그림 4) TCMalloc의 전체 구조

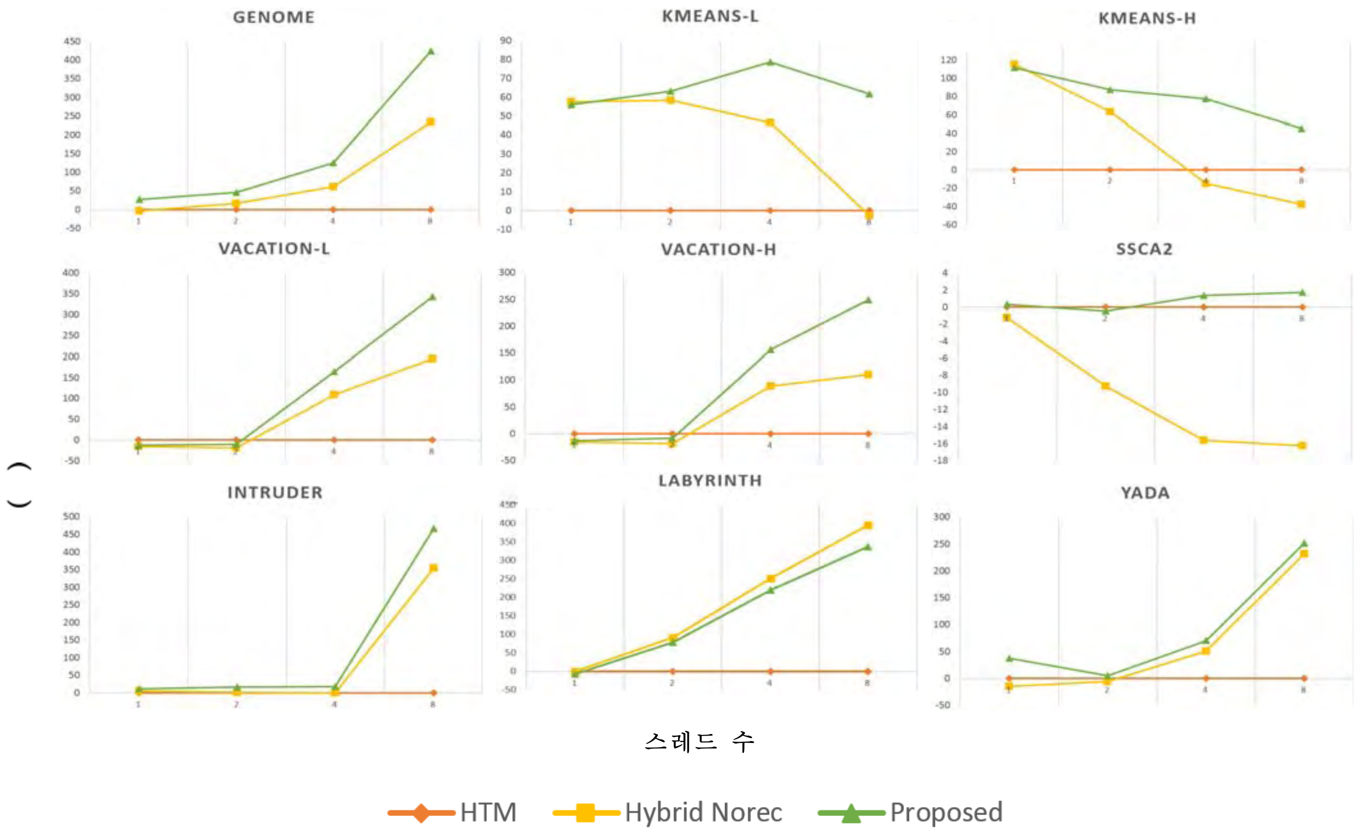
4. 성능평가

본 절에서는 제안하는 기법의 성능을 검증하기 위해 제안하는 기법의 성능평가를 수행한다. 성능평가 대상은 HTM, Hybrid NORec이다. 성능평가는 STAMP(Stanford Transactional Applications for Multi-Processor) 벤치마크[5]를 통해, 워크로드에 대한 정량적인 처리 시간 및 HTM을 기준으로 한 throughput에 대한 비교를 수행한다. 실험에 사용된 환경은 <표 1>과 같다.

<그림 5>는 STAMP 벤치마크에서 각 워크로드에 따른 HTM 대비 트랜잭션 처리 성능 시간을 측정한 결과이다. 특히, 8 스레드를 기준으로 제안하는 기법(Proposed)이 전체적인 워크로드에서 HTM에 비해 약 240%, Hybrid Norec에 기법에 비해 약 45%의 성능 향상을 보인

<표 1> 실험 환경

항목	성능
CPU	Intel Haswell Xeon E3-1220v3 processor 3.10 GHz
RAM	32GB(8GB×4EA)
OS	Linux Ubuntu 12.04 LTS
Compiler	gcc
Hyperthreading	enabled



(그림 5) 스레드 수 변화에 따른 HTM 대비 트랜잭션 처리속도

다. 이는 제안하는 기법이 멀티코어 환경에서 기존 연구보다 최적의 트랜잭션 처리 성능을 보임을 알 수 있다. SSCA2 벤치마크는 작은 읽기/쓰기 데이터 셋으로 이루어져 있기 때문에 스레드 개수가 적은 경우 HTM으로 처리되는 것이 더 좋은 성능을 나타낸다. 하지만 스레드가 높아질수록 제안하는 기법이 더 좋은 성능을 나타냄을 알 수 있다. 또한 KMEANS 벤치마크의 경우 스레드 개수가 많을수록 Hybrid Norec 기법은 메모리 충돌에 의해 성능이 하락하는 반면 제안하는 기법은 하락 폭이 낮다. 이는 제안하는 기법이 성능 저하를 일으키는 긍정 오류를 해결하고, 우수한 성능을 지닌 메모리 관리자를 이용하여 필요한 데이터를 효율적으로 할당/해제를 수행하기 때문이다.

5. 결론

본 논문에서는 멀티코어 환경에서 효율적인 트랜잭션 처리를 위한 메모리 관리 기반 하이브리드 트랜잭셔널 메모리 기법을 제안하였다. 제안하는 기법은 블룸필터가 지닌 긍정 오류를 최소화하여 성능을 개선하였다. 아울러 하이브리드 트랜잭셔널 메모리 기법의 성능 극대화를 위해, 멀티코어 환경에서 트랜잭션에 최적화된 메모리 관리자를 제공한다. STAMP 벤치마크 성능평가 결과, 8스레드에서 HTM 대비 약 240%, NOrec Hybrid 대비 약 45% 성능향상을 보였다. 향후 연구로는 본 논문에서 제안한 하이브리드 트랜잭셔널 메모리 기법을 긴 트랜잭션에서도 우수한 성능을 나타내도록 확장하는 것이다.

참고문헌

[1] L. Dalessandro, M. F. Spear, and M. L. Scott. NOrec: Streamlining STM by Abolishing Ownership Records. In 15th ACM Symp. on Principles and Practice of Parallel Programming, 2010.

[2] Dalessandro, Luke, et al. "Hybrid norec: A case study in the effectiveness of best effort hardware transactional memory." ACM SIGARCH Computer Architecture News pp 39-52, 2011.

[3] Yun, Heechul, et al. "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms." IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014.

[4] Sanjay Ghemawat, Paul Menage, "TCMalloc: Thread-Caching Malloc", http://goog-perftools.sourceforge.net/doc/tcmalloc.html

[5] Minh, Chi Cao, et al. "STAMP: Stanford transactional applications for multi-processing." Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on. IEEE, 2008.