ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 25, No. 12, pp. 628-633, 2019. 12
https://doi.org/10.5626/KTCP.2019.25.12.628

데이터 복구를 지원하는 비휘발성 메모리 기반 하이브리드 트랜잭셔널 메모리 기법

(Non-Volatile Memory Based Hybrid Transactional Memory Technique Supporting Data Recovery)

마 현 국 † 김 형 진 † 신 재 환 † 장 진 수 † 장 재 우 ††
(Hyeon-Guk Ma) (Hyeong-Jin Kim) (Jae-Hwan Shin) (Jin-Su Chang) (Jae-woo Chang)

요 약 트랜잭셔널 메모리는 프로그램의 명령어 집합을 트랜잭션 단위로 분할함으로써 높은 성능 및 유지 보수의 편리성을 제공한다. 그러나 시스템 실패가 발생할 경우, 수행된 데이터를 보호하지 못하는 문제점이 존재한다. 이를 해결하기 위해 NV-HTM이라 불리는 비휘발성 메모리 기반 트랜잭셔널 메모리 기법이 제안되었다. NV-HTM은 비휘발성 메모리상에서 시스템 실패가 발생해도 데이터 복구를 지원하는 하드웨어 트랜잭셔널 메모리(HTM)이다. 그러나 NV-HTM은 fallback 경로로 SGL을 사용하기 때문에 하드웨어 트랜잭셔널 메모리로 실패한 트랜잭션에 대해서는 병렬 처리를 지원하지 못한다. 따라서 본 논문에서는 비휘발성 메모리상에서 데이터 복구를 지원하며 fallback 경로와 HTM 경로 사이에 병렬 처리를 지원하는 비휘발성 메모리 기반 하이브리드 트랜잭셔널 메모리 기법(DHyTM)을 제안한다. 성능평가 결과, 제안하는 DHyTM이 STAMP 벤치마크에서 NV-HTM에 비해 평균 270%의 성능 향상이 있음을 보인다.

키워드: 병렬 프로그래밍, 트랜잭셔널 메모리, 비휘발성 메모리, 복구

Abstract Transactional memory provides high performance and ease of maintenance by dividing the instruction set of a program into transactional units. However, data is lost if system failure occurs. To solve this problem, a non-volatile memory-based transactional memory called NV-HTM has been proposed. NV-HTM is a hardware transactional memory (HTM) that supports data recovery on NVM(Non-Volatile Memory) even in the event of system failure. However, since NV-HTM uses SGL as a fallback path, it cannot support parallel processing for transactions which fail using hardware transactional memory. Thus, in this paper, we propose a NVM-based hybrid transactional memory (DHyTM) that provides data recovery on NVM and supports parallel processing between the fallback path and HTM path. It is shown from the performance evaluation that the proposed DHyTM has demonstrated an average performance improvement of 270% over NV-HTM in the STAMP benchmark.

Keywords: transactional memory, non-volatile memory, parallel programming, data recovery

· 이 논문은 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발 사업의 일환으로 수행하였음(IITP-2017-R0113-15-0005, 대규모 트랜잭션 처리와 실시간 복합 분석을 통합한 일체형 엔지니어링 기술 개발). 또한, 이 논문은 2019년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2019R1I1A3A01058375)
· 이 논문은 2019 한국컴퓨터종합학술대회에서 '비휘발성 메모리에서 복구를 지원하는 내구성의 하이브리드 트랜잭셔널 메모리 기법'의 제목으로 발표된 논문을 확장한 것임

† 비 회 원 : 전북대학교 컴퓨터공학과
akgusrnr123@naver.com
yeon_hui4@jbnu.ac.kr
djtm99@jbnu.ac.kr
lklil@jbnu.ac.kr
†† 종신회원 : 전북대학교 IT정보공학과 교수(Chonbuk Nat'l Univ.)
jwchang@jbnu.ac.kr
(Corresponding author임)

논문접수 : 2019년 9월 17일
(Received 17 September 2019)
심사완료 : 2019년 10월 24일
(Accepted 24 October 2019)

정보과학회 컴퓨팅의 실제 논문지 제25권 제12호(2019. 12)

1. 서 론

최근 멀티코어 프로세서가 널리 보급됨에 따라 동시성 제어를 통해 효율성을 높이는 병렬 프로그래밍 연구가 활발히 진행되었다. Herlihy가 제안한 트랜잭셔널 메모리(Transactional Memory: TM) 기술은 동시성 제어를 위해 데이터베이스의 트랜잭션 개념을 이용하여 일련의 명령어 집합을 하나의 트랜잭션으로 나누어 처리한다[1]. 트랜잭셔널 메모리의 구현 방식은 크게 소프트웨어 방식, 하드웨어 방식, 하이브리드 방식으로 분류된다. 트랜잭셔널 메모리는 전력 차단, OS 크래시(crash) 등과 같은 시스템 실패(system failure)가 발생할 경우, 데이터를 손실하여 시스템의 내구성(durability)을 지원하지 못하는 문제점이 존재한다.

이를 해결하기 위해 비휘발성 메모리(Non-volatile Memory: NVM) 기반 트랜잭셔널 메모리 기법이 제안되었다[2-5]. NVM은 전원이 공급되지 않아도 데이터를 유지하는 메모리로써, DRAM과 유사한 입출력 속도를 제공하며, byte 단위의 입출력를 지원한다. NVM을 하드웨어 트랜잭셔널 메모리(Hardware Transactional Memory: HTM)에 결합하여 ACID 특성을 제공한 대표적인 연구는 NV-HTM이다[4]. NV-HTM은 트랜잭션 상에서 로그를 유지하고, 시스템 실패 시에 로그를 이용하여 데이터 복구를 지원한다. 그러나 NV-HTM은 하드웨어 트랜잭셔널 메모리를 사용하기 때문에 크기가 큰 트랜잭션을 처리하지 못하며, SGL(single global lock)을 통해 처리하기 때문에 병렬성이 저하되는 단점이 존재한다. NV-HTM의 단점을 해결하기 위해, 본 논문에서는 비휘발성 메모리 상에서 데이터 복구를 지원하는 하이브리드 트랜잭셔널 메모리 기법을 제안한다. 제안하는 기법은 fallback 경로로 NOrecSTM을 사용하여 하드웨어 트랜잭셔널 메모리로 실패한 트랜잭션에 대해서도 병렬처리를 지원하고 HTM의 원자성(atomicity)과 NVM의 비휘발성 특성을 결합하여 시스템 내구성을 제공한다. 본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로써 NVM 기반의 트랜잭셔널 메모리 기법을 소개한다. 3장에서는 데이터 복구를 지원하는 비휘발성 메모리 기반 하이브리드 트랜잭셔널 메모리 기법을 제안한다. 4장에서는 제안하는 기법의 성능평가를 수행한다. 마지막으로, 5장에서는 결론 및 향후 연구 방향에 대해서 기술한다.

2. 관련연구

비휘발성 메모리와 트랜잭셔널 메모리를 결합하여 시스템 복구를 제공하려는 다양한 연구가 수행되었다. 첫째, Hillel Avni et al.이 제안한 PTM은 HTM에 NVM을 적용하여 시스템 복구(durability)를 제공하는 연구이다[2]. 그러나 HTM이 실패 시, 병렬성이 감소하는 문제점이 존재한다. 둘째, Hillel Anvi et al.이 제안한 PHyTM은 PHTM을 확장한 연구로써, 시스템 복구를 제공할 뿐 아니라 NVM 상에서 rwlock을 이용한 HTM과 STM의 동시성 제어를 수행한다[3]. HTM이 실패하여도 STM으로 수행하여 HTM과 동시에 수행 가능하다. 그러나 HTM의 캐시 간 일관성 프로토콜(cache coherence protocol)을 이용하지 않고 rwlock의 소프트웨어 방식을 사용함으로써 속도가 느리다. 셋째, Daniel Castro et al.이 제안한 NV-HTM은 HTM에 NVM 로깅을 통해 시스템 내구성을 보장한다[4]. 그러나 트랜잭션 크기가 증가하고, 동시에 수행하는 스레드 수가 증가하면 병렬성이 크게 감소하는 문제점이 존재한다. 넷째, Arpit Joshi et al.이 제안한 DHTM은 HTM을 활용하여 원자적 내구성을 제공하는 연구로써, L1 캐시 크기에서 LLC(Last Level Cache)까지 트랜잭션의 로깅을 지원하여 원자적 내구성을 제공한다[5]. 그러나 DHTM은 캐시 간 일관성 프로토콜을 수정하기 때문에 상용 HTM에서 시스템 복구를 지원하기 어렵다는 한계점이 있다.

기존 기법들은 비휘발성 메모리를 통한 시스템 내구성에 초점을 맞추어 HTM과 NVM을 결합한 연구를 수행하였다. 그러나 HTM만으로 수행할 수 없는 워크로드의 경우, fallback 경로로 SGL을 지원하거나 STM 방법을 사용하여 HTM의 캐시 간 일관성 프로토콜을 이용한 동시성 제어를 지원하지 못하는 단점이 존재한다.

3. 제안하는 비휘발성 메모리 기반 하이브리드 트랜잭셔널 메모리 기법

3.1 전체 시스템 구조

제안하는 비휘발성 메모리 상에서 복구를 지원하는 하이브리드 트랜잭셔널 메모리 기법은 그림 1과 같다. 제안하는 DHyTM은 크게 LiteHTM과 NOrecSTM[6]으로 구성된다. HTM은 빠른 트랜잭션 처리를 보장하지만 캐시 용량 문제, 데이터 충돌, OS 스케줄링에 의한 종료 등의 이유로 실패(abort) 횟수가 특정 임계값을 넘으면 트랜잭션을 처리할 수 없다고 판단하고 NOrec-STM으로 처리한다. NOrecSTM은 공유 메모리에 있는 Ownership 레코드인 메타데이터를 제거함으로써 빈번한 충돌 혹은 큰 용량의 트랜잭션에 대한 처리 성능을 높인 기법이다. NOrecSTM은 commit 단계에서 Global_seq_lock과 Local_seq_lock을 비교하여 commit이 있었는지 체크한다. 다른 스레드에서 commit을 수행한 사실을 발견하면, 자신이 사용한 데이터의 주소값에 접근하여 값이 변경되었는지 확인하여 abort를 수행한다. NOrecSTM은 메타데이터를 유지하지 않고, 간단한 확인절차를 통해 트랜잭션의 commit 여부를 확인함으로써 대부분의 트랜잭션에서 우수한 성능을 보인다.

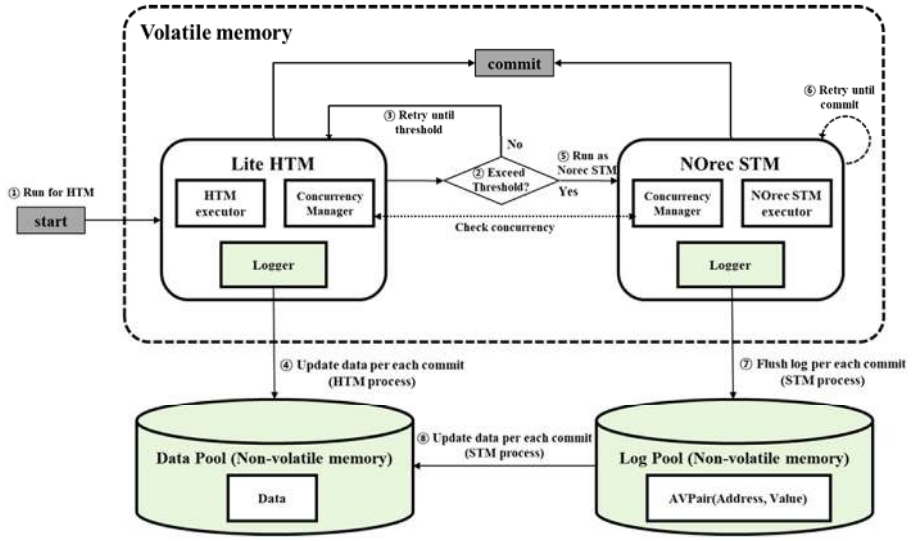

그림 1 제안하는 하이브리드 트랜잭셔널 메모리 기법
Fig. 1 Proposed Hybrid Transactional Memory Technique

제안하는 하이브리드 트랜잭셔널 메모리 기법의 전체 수행과정은 다음과 같다. 트랜잭션은 먼저 LiteHTM으로 수행되며(①), LiteHTM으로 트랜잭션을 처리 중에 실패(abort)가 발생하면 재시도 횟수가 임계값을 초과하는지 확인하고(②) 초과하지 않았다면 재시도한다(③). LiteHTM이 실패하지 않고 트랜잭션 처리를 완료하면 HTM의 원자성 특성에 의하여 비휘발성 메모리의 데이터 저장소에 변경된 데이터가 반영된다(④). 만약 재시도 횟수를 모두 소진한 경우 NOrecSTM으로 처리하며(⑤), NOrecSTM의 경우 트랜잭션 처리에 성공할 때까지 재시도를 수행하고(⑥) 트랜잭션 처리에 성공한 경우 로그를 비휘발성 메모리의 로그 저장소에 flush한다(⑦). 마지막으로 비휘발성 메모리의 로그 저장소 정보를 바탕으로 데이터 변경을 수행하여 트랜잭션 처리를 완료한다(⑧).

3.2 트랜잭션 메모리 간 동시성 제어 알고리즘

동시성 제어 알고리즘은 멀티코어 환경에서 작업 병렬성을 극대화할 수 있는 중요한 기능이다. HTM과 STM은 서로 다른 특성을 지니고 있기 때문에, 이를 고려한 동시성 제어 기법이 필수적이다. LiteHTM의 동시성 제어 알고리즘은 그림 2와 같다. 첫째, LiteHTM path 인 경우 ① 트랜잭션 시작되면, LiteHTM을 이용하여 트랜잭션 처리를 시도한다. ② 트랜잭션이 abort되지 않고 commit하게 되면 Global TS를 2 증가시킨 후 NVM에 데이터가 변경되고 성공적으로 commit이 완료된다. ③ 만약 트랜잭션이 abort되면 retry 수를 1 감소시키고 재시도한다. 만약 retry가 0이 되면 더 이상 LiteHTM으로 처리할 수 없다고 판단하고 NOrecSTM으로 처리한다.

Algorithm 1. LiteHTM Algorithm

```
1  HTM_BEGIN(){
2      _xbegin();
3  }
4  abortHTM() {
5      _xabort()
6  }
7  sharedReadHTM(addr) {
8      return *addr
9  }
10 sharedWriteHTM(addr, value) {
11     if (isRO == 1) isRO = 0
12     *addr = value;
13 }
14 commitHTM() {
15     xend() // LiteHTM Commit
16     seqlock = seqlock+2
17 }
```

그림 2 LiteHTM 트랜잭셔널 메모리
Fig. 2 LiteHTM transactional memory

NOrecSTM의 동시성 제어 알고리즘은 그림 3과 같다. ① 트랜잭션이 시작되면, NOrecSTM을 이용하여 트랜잭션 처리를 시도한다. 이때, Global TS를 Local TS에 저장한다. ② NOrecSTM의 read 명령을 수행할 때, 로컬에 저장된 AVpair(address/value)와 실제 NVM 데이터 풀(pool)에 존재하는 address/value 를 비교하고 일치하지 않으면 abort를 수행한다. ③ NOrecSTM을 commit하기 전엔 validation을 수행한다. validation은 Global TS와 Local TS가 다른 경우 수행하고, 문제가

Algorithm 2. NOrecSTM Algorithm

```
1  TM_BEGIN(){
2      start_seq = seqlock
3      _xbegin();
4  }
5  abortSTM() {
6      resetLog();
7      goto 1;
8  }
9  sharedReadSTM(addr) {
10     if (seqlock != start_seq) {
11         if (readSetCoherent())
12             abortSTM()
13         seqlock = start_seq
14     }
15     return *addr
16 }
17 sharedWriteSTM(addr, value) {
18     LocalLog.push(addr, val)
19 }
20 commitSTM() {
21     if ((seqlock & 1)) {
22         if (readSetCoherent())
23             abortSTM()
24         seqlock = start_seq
25     }
26     LocalLog.flush()
27     CommitLog.flush()
28     seqlock = seqlock + 2
29 }
30 readSetCoherent(){
31     if(Local_readset != Global_readset)
32         abortSTM()
33 }
```

그림 3 NOrecSTM 트랜잭셔널 메모리
Fig. 3 NOrecSTM transactional memory

없으면 caslock을 획득한 후 데이터를 변경한다. caslock은 Global TS를 1 증가시킴으로써 획득한다. caslock을 획득하면 Log을 flush하고 데이터를 변경한다. 마지막으로 caslock을 해제하고 Global TS를 2 증가시킨 후 종료한다. cas_lock은 Global TS을 atomic하게 1 증가시킴으로써 획득하고 1 감소시킴으로써 해제한다. 이를 통해 두 개의 NOrecSTM이 동시에 데이터가 변경되는 것을 방지한다.

3.3 NVM 기반 로깅 알고리즘 및 복구 알고리즘

제안하는 비휘발성 메모리 기반 하이브리드 트랜잭셔널 메모리 기법은 LiteHTM과 NOrecSTM을 사용하여 트랜잭션 처리 효율을 높인다. 그러나 각 TM별로 메모리에서 데이터가 변경되는 구조가 다르기 때문에, TM에 맞는 로깅 기법을 적용해야 한다. 첫째, LiteHTM에서 실행되는 트랜잭션의 경우 트랜잭션 블록 내에서 읽기/ 쓰기가 진행되었던 데이터는 CPU 내의 캐시에서만 작업이 진행되는 샌드박싱의 특징을 지닌다. 따라서 트랜잭션 실행 도중 발생되는 로그는 캐시에 존재하기 때문에, LiteHTM에서 commit이 완료된 경우 데이터 변경이 이미 NVM에 반영된 상태이므로 데이터 복구를 수행할 필요가 없다. LiteHTM은 원자성을 제공하기 때문에 트랜잭션이 commit 되는 도중의 시스템 실패는 발생하지 않는다. 따라서 제안하는 DHyTM은 PHyTM[3] 마찬가지로 LiteHTM 경로인 경우 로그 플러시를 수행하지 않는다.

둘째, NOrecSTM은 쓰기 명령어를 수행할 때, 해당 로그를 로컬 데이터에 먼저 저장한다. NOrecSTM은 commit 단계에 들어가면 로그를 NVM에 플러시한다. 이 때, commit log를 0로 세팅하고 플러시를 수행함으로써, 로그가 commit이 완료를 판단할 수 있다. 로그 플러시가 완료된 후, NVM에 저장된 로그를 바탕으로 NVM 에 변경된 데이터를 반영한다. 데이터가 반영되면 commit log를 1로 세팅하여 플러시를 수행라고 commit이 완료되었음을 표시한다. 즉, commit log가 0인 경우, 로그는 플러시 되었으나 commit이 완료되지 않았으므로 복구를 수행한다. commit 로그가 1인 경우, 로그도 플러시 되었고 commit도 완료가 되었으므로 복구할 필요가 없다.

4. 성능평가

4.1 성능평가 환경

성능평가를 위한 환경은 표 1과 같다. 비휘발성 메모리는 아직 충분히 상용화 단계에 진입하지 못했기 때문에, 기존 연구인 NV-HTM과 같이 Emulated 환경을 사용한다.[4] 비휘발성 메모리는 DRAM과 쓰기 성능이 다르기 때문에, 100ns의 지연을 추가하여 성능평가를 수행하였다. 성능평가에 사용된 벤치마크는 STAMP의 HTM 패치 버전을 사용하였다[7]. 총 8개의 워크로드로 구성되며 각각의 특성은 표 2와 같다. 8개의 워크로드 중 bayes는 비결정적(non-deterministic)특성을 가지고 있기 때문에 성능평가 대상에서 제외시켰다. 성능 평가를 위해 제안하는 DHyTM을 기존 연구 중 우수한 기법으로 알려진 NV-HTM[4]과 비교하였다. 성능 비교를 위해 각 워크로드 별 Throughput을 10회 측정하여 평균을 사용하였다.

표 1 성능평가 환경
Table 1 Performance environment

Category	Peformance
CPU	Intel Xeon Processor E5-2630v3 10-Core
Memory	64GB
OS	Linux ubuntu 18.04.1 LTS
Compiler	gcc, g++
NVM	Emulated NVM

표 2 STAMP 벤치마크 워크로드에 대한 특성
Table 2 Characteristics for STAMP Benchmark Workloads

Application	Tx Length	R/W Set	Tx Time	Contention
bayes	Long	Large	High	High
genome	Medium	Medium	High	Low
intruder	Short	Medium	Medium	High
kmeans	Short	Small	Low	Low
labyrinth	Long	Large	High	High
ssca2	Short	Small	Low	Low
vacation	Medium	Medium	High	Medium
yada	Long	Large	High	Medium

4.2 성능평가 결과

그림 4는 제안하는 DHyTM과 기존 NV-HTM의 워크로드별 Throughput을 스레드가 증가함에 따라서 나타낸다. 제안하는 DHyTM이 기존 기법에 비해 전체 스레드 기준 평균 약 270% 우수한 성능을 보인다. 제안하는 DHyTM이 Labyrinth와 Ssca2를 제외한 모든 벤치마크에서 기존 기법보다 약 100%~260% 성능 향상을 보이며, 특히 Labyrinth의 경우 fallback 경로로 수행되는 다수의 트랜잭션이 병렬 처리되기 때문에 약 780% 성능 향상을 보인다.

표 3 스레드 별 평균 성능
Table 3 Average performance by thread

Thread	1	2	4	6	8	10
Performance	219%	230%	244%	283%	303%	384%

스레드 별 평균 성능은 표 3과 같다. 1 스레드의 경우, 제안하는 DHyTM이 기존 기법에 비해 약 220% 성능 향상을 보이고, 10 스레드의 경우 약 380% 성능 향상을 보인다. 그 이유는 스레드가 높아질수록 HTM의 abort가 증가하여 fallback 경로로 수행되는 트랜잭션의 비율이 높아지며, 이때 기존 NV-HTM은 fallback 경로로 수행되는 트랜잭션이 SGL에 의해 직렬 처리됨으로써 성능이 저하되기 때문이다. 반면 제안하는 DHyTM은 fallback 경로의 수행되는 트랜잭션의 수가 많아도 HTM과 병렬적으로 처리되기 때문에 우수한 성능을 보인다.

5. 결론 및 향후 연구

효율적인 동시성 제어를 위해 고안된 트랜잭셔널 메모리 기법은 시스템 실패에 대해 데이터를 보호하지 못하기 때문에, 비휘발성 메모리상에서 병렬 처리를 지원하는 HTM이 제안되었다. 그러나 대표적인 기존 기법인 NV-HTM은 fallback 경로로 SGL을 사용하기 때문에

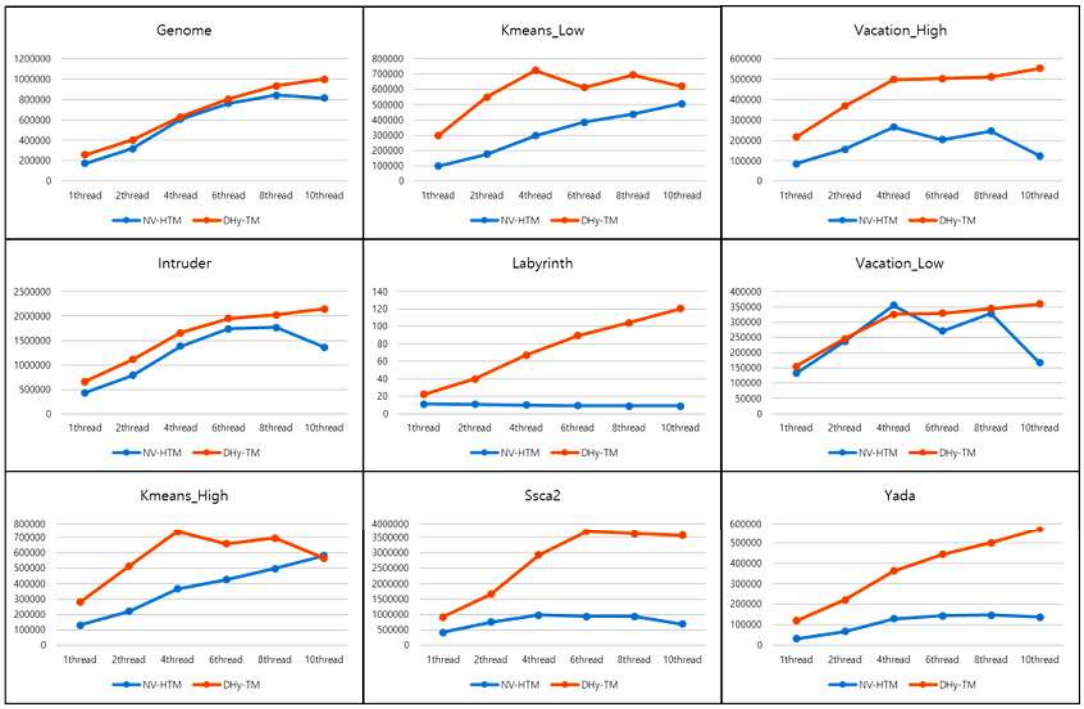

그림 4 멀티코어 환경에서 각 워크로드 Throughtput 성능평가
Fig. 4 Performance Evaluation of Each Workload Throughput in Multicore Environments

HTM의 병렬성을 저해시킨다. 따라서 본 논문에서는 비휘발성 메모리 상에서 데이터 복구를 지원하는 하이브리드 트랜잭셔널 메모리 기법(DHyTM)을 제안하였다. 제안하는 DHyTM은 로그 기반 복구를 지원할 뿐만 아니라 HTM의 병렬성을 증대시키는 NOrecSTM 기법을 fall-back 경로로 사용함으로써 기존 기법에 비해 약 260%의 성능 향상을 보였다. 향후 연구는 비휘발성 메모리상에서 트랜잭션 특징에 따른 재시도 최적화를 수행하여 보다 효율적인 트랜잭션 처리 기술을 연구하는 것이다.

References

[1] M. Herlihy and J. E. B. Moss, "Transactional Memory: Architectural Support for Lock-Free Data Structures," *Proc. 20th Annual Int. Symp. on Computer Architecture*, pp. 289-300, 1993.

[2] Z. Wang, H. Yi, R. Liu, M. Dong, and H. Chen, "Persistent transactional memory," *IEEE Computer Architecture Letters*, Vol. 14, No. 1, pp. 58-61, 2015.

[3] H. Avni, T. Brown, "PHyTM: Persistent Hybrid Transactional Memory," *VLDB Endowment*, Vol. 10, No. 4, pp. 409-420, 2016.

[4] D. Castro, P. Romano and J. Barreto, "Hardware Transactional Memory Meets Memory Persistency," *IEEE International Parallel and Distributed Processing Symposium*, IEEE, pp. 368-377, 2018.

[5] A. Joshi, V. Nagrarajan, M. Cintra and S. Vigals, "DHTM: Durable Hardware Transactional Memory," *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture*, pp. 452-465, 2018.

[6] Dalessandro, Luke, Michael F. Spear, and Michael L. Scott, "NOrec: streamlining STM by abolishing ownership records," *ACM Sigplan Notices*, pp. 67-78, 2010.

[7] T. Nakaike, R. Odaira, M. Gaudet, M. M. Michael and H. Tomari, "Quantitative comparison of Hardware Transactional Memory for Blue Gene/Q, zEnterprise EC12, Intel Core, and POWER8," *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture*, pp. 144-157, 2015.

마 현 국

2018년 전북대학교 IT정보공학과 졸업(학사). 2018년~현재 전북대학교 컴퓨터공학과 석사과정 재학중. 관심분야는 트랜잭셔널 메모리, 데이터 마이닝, 암호화 질의 처리

김 형 진

2015년 전북대학교 IT정보공학과 졸업(학사). 2017년 전북대학교 컴퓨터공학과 졸업(석사). 2017년~현재 전북대학교 컴퓨터공학과 박사과정 재학중. 관심분야는 트랜잭셔널 메모리, 데이터 마이닝, 암호화 질의 처리

신 재 환

2018년 전북대학교 IT정보공학과 졸업(학사). 2018년~현재 전북대학교 컴퓨터공학과 석사과정 재학중. 관심분야는 트랜잭셔널 메모리, 데이터 마이닝, 암호화 질의 처리

장 진 수

2018년 전북대학교 IT정보공학과 졸업(학사). 2018년~현재 전북대학교 컴퓨터공학과 석사과정 재학중. 관심분야는 트랜잭셔널 메모리, 데이터 마이닝, 암호화 질의 처리

장 재 우

1984년 서울대학교 전자계산기공학과 졸업(학사). 1986년 한국과학기술원 전산학과 졸업(석사, 박사). 1996년~1997년 Univ. of Minnesota, Visiting Scholar. 2003년~2004년 Penn State Univ, Visiting Scholar. 1991년~현재 전북대학교 IT정보공학과 교수. 관심분야는 공간 데이터베이스, 하부 저장구조, 센서 네트워크, 데이터베이스 아웃소싱, 빅데이터