The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 2, pp.179-185, Apr. 30, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.2.179
JIIBC 2020-2-24

선택적 리프레시를 통한 DRAM 에너지 효율 향상 기법

Techniques to improve DRAM Energy Efficiency through Selective Refresh

김영웅*

Young-Ung Kim*

요 약 DRAM은 메인 메모리 시스템을 구성하는 주요한 요소로서 운영체제의 발전, 응용 프로그램의 복잡도와 용량의 증가에 맞추어 DRAM의 용량과 속도 역시 증가하는 추세이다. DRAM은 주기적으로 저장된 값을 읽은 후 다시 저장하는 리프레시 동작을 수행해야 하며, 이에 수반되는 성능 및 파워/에너지 오버헤드는 용량이 증가할수록 더 악화되는 특성을 내재하고 있다. 본 연구는 전하의 보존 시간이 가장 낮은 셀들에 대해서 블룸 필터를 사용하여 64ms, 128ms 이내에 리프레시를 수행해야 하는 로우들을 효율적으로 저장하여 선택적 리프레시를 수행하는 에너지 효율 향상 기법을 제안한다. 실험 결과에 따르면 제안 기법을 통하여 평균 5.5%의 성능 향상이 있었으며, 리프레시 에너지는 평균 76.4% 절감되었고, 평균 EDP는 10.3% 절감된 것으로 나타났다.

Abstract DRAM is a major component of the main memory system. As the operating system evolves and application complexity and capacity increases, the capacity and speed of DRAM are also increasing. DRAM should perform a refresh action of periodically reading and then re-storing stored values, and the accompanying performance and power/energy overhead embodies characteristics that worsen as capacity increases. This study proposes an energy efficiency improvement technique that efficiently stores the rows that need to be refreshed within 64ms and 128ms using the bloom filter for cells with the lowest retention time of electrons. The results of the experiment showed that the proposed technique resulted in an average 5.5% performance improvement, 76.4% reduction in average refresh energy, and 10.3% reduction in average EDP.

Key Words : DRAM, Memory Energy Performance, Refresh, Selective Refresh

Ⅰ. 서 론

DRAM의 셀을 구성하는 소자의 특성상 시간이 지남에 따라 서서히 방전되어 저장된 값이 손실되는 특성이 있다. 데이터 손실을 방지하기 위하여 DRAM은 주기적으로 저장된 값을 읽은 후 다시 쓰는 리프레시 과정이 필요하며, 이 과정은 성능과 파워/에너지의 오버헤드를 유발하는 주요 원인이 된다.

이러한 리프레시 오버헤드를 절감할 수 있는 많은 하드웨어 기법들이 이전 연구들에 의하여 제안되었다. 우선

*정회원, 한성대학교 컴퓨터공학부(교신저자)
접수일자 2020년 2월 3일, 수정완료 2020년 3월 5일
게재확정일자 2020년 4월 3일

Received: 3 February, 2020 / Revised: 5 March, 2020 / Accepted: 3 April, 2020
*Corresponding Author: yukim@hansung.ac.kr
Dept. of Computer Engineering, Hansung University, Korea

DRAM 디바이스를 수정하여 DRAM의 서로 다른 셀들에 대해 다른 주기로 리프레시를 수행하는 기법이 있지만[1][2], 이 기법들은 DRAM 다이(die) 상에서 최대 20% 가량의 공간 오버헤드를 유발하는 단점이 있어 실제로 적용하기 어렵다. 다른 기법으로는 메모리 컨트롤러를 수정하여 불필요한 리프레시를 피하거나[3], 리프레시 주기를 전체적으로 낮춘 후, 보존 에러(retention error)가 발생하면 error-correcting code를 사용하여 값을 복원하는 기법이 있다[4]. 그러나 이 기법 역시 심각한 공간 오버헤드의 영향을 받는다. 마지막으로 각 DRAM 로우(row)들의 최대 보존 시간을 사전에 검사한 후 이 로우들에 대해 불필요한 리프레시를 수행하지 않는 방법이 제안되었지만[5], 이 기법은 컨트롤러가 로우 단위로 리프레시 명령을 보내야 하며, 결과적으로 DRAM의 병렬성을 떨어뜨려 범용으로 적용되기 어렵다.

본 논문은 리프레시의 수행 횟수를 효과적으로 줄이면서도 하드웨어의 복잡도를 크게 증가시키지 않는 범용으로 사용 가능한 선택적 리프레시 기법을 제안한다. 현재의 JEDEC DRAM 설계 표준에 의하면 85℃이하의 온도에서 각 로우에 대해 64㎳마다(85℃ 이상에서는 32㎳마다) 리프레시를 수행하도록 규정하고 있지만, 측정 결과에 의하면 약 0.1%의 DRAM 셀들을 제외하고는 대부분의 셀들이 256㎳ 동안(85℃ 이상에서는 128㎳) 저장된 값을 보존하는 것으로 나타났다[5][6]. 따라서 DRAM의 보존 시간에 따라 선택적으로 리프레시를 수행한다면 리프레시로 인한 오버헤드를 효과적으로 절감할 수 있다.

DRAM의 복잡도를 증가시키지 않기 위하여 제안 기법은 기존 DRAM과 같이 셀이 아닌 로우 단위의 리프레시를 가정하였다. 따라서 약한 DRAM 셀을 포함한 로우의 위치를 저장한 후, 저장된 로우들만 자주 리프레시를 수행해주고 나머지 로우들은 더 긴 주기로 리프레시를 수행한다. 하지만 만약 약 1000여개의 로우들을 테이블 또는 리스트 형태로 관리하면 큰 공간 및 에너지 오버헤드가 발생한다. 따라서 제안 기법에서는 블룸 필터 구조를 사용하며, 각 DRAM 뱅크 (bank)마다 두 개의 필터를 사용하여 64㎳의 리프레시 주기를 가진 로우들과 128㎳의 주기를 가진 로우들을 저장하고, 이외의 로우들은 256㎳ 마다 리프레시를 수행한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 DRAM 기반 메인 메모리 구조를 간략히 소개하고, 제 3장에서는 블룸 필터 및 제안 기법에 대해 기술한다. 제 4장에서는 시뮬레이션을 통한 제안 기법의 평가하고, 마지막으로 5장에서 결론을 통해 본 연구의 기술적 기여도를 정리한다.

II. 배 경

1. DRAM 계층구조

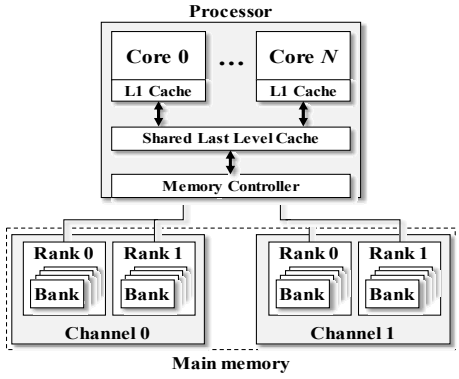


그림 1. DRAM 계층구조
Fig. 1. Hierarchical structure of DRAM

현대의 DRAM 메모리는 그림 1과 같이 계층적으로 구성된다. 프로세서 내부에 SRAM으로 구성된 Last-Level Cache(LLC)에서 사용되는 캐시 데이터의 hit/miss 여부에 따라 프로세서 내부의 메모리 컨트롤러는 데이터를 DRAM에서 가져오거나 DRAM으로 저장한다. 메모리 컨트롤러는 DRAM의 각 채널(channel)로 명령을 보내는데, 이 채널은 서로 병렬적으로 동작할 수 있도록 독립적인 커맨드, 주소, 그리고 버스를 가지고 있다. 하나의 채널은 하나 이상의 랭크(rank)를 가지고 있으며, 각 랭크는 독립적인 DRAM 디바이스를 가지고 있다. 또한 채널 내 모든 랭크는 병렬적으로 동작을 수행한다. 각 랭크는 다수의 뱅크를 가지고 있으며, 뱅크 내에는 DRAM 셀들이 2차원 배열 형태로 존재한다. 각각의 뱅크 역시 병렬적으로 동작 가능하지만 실제로는 채널 및 랭크로 인하여 병렬 동작이 일부 제한되는 경우가 있다.

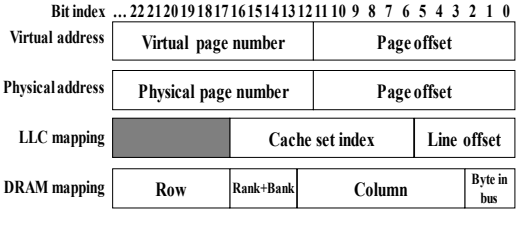


그림 2. 프로세서, 캐시, DRAM의 주소 Mapping 방식
Fig. 2. Address Mapping for Processor, Cache and DRAM

그림 2는 Intel i7-2600 프로세서 아키텍처 상에서 주소 값이 각 계층 구조에 따라 어떻게 매핑되는지를 나타낸다. 먼저 프로세서 내의 각 코어(core) 및 어플리케이션은 가상 주소(virtual address)를 사용하여 실제 메모리 및 저장소의 구조와 크기에 상관없이 독립적인 논리 주소를 통해 메모리 접근을 수행한다. 이 논리 주소가 가리키고 있는 메모리의 위치를 찾기 위하여 페이지 테이블(page table) 및 TLB(translation look-aside buffer)를 참조하여 물리 주소를 얻는다. 이 물리 주소 체계는 LLC에서 동일하게 사용되지만, LLC는 실제 메모리의 일부만을 반영하는 작은 크기이므로 일부 상위 비트들은 캐시의 위치를 결정하는 값이 아닌 태그(tag)로 사용된다. 이 주소가 DRAM으로 내려오게 되면 DRAM은 이 주소 값의 비트 위치에 따라 채널, 랭크, 뱅크, 로우, 칼럼을 결정하게 된다.

2. DRAM 내부 구조 및 리프레시

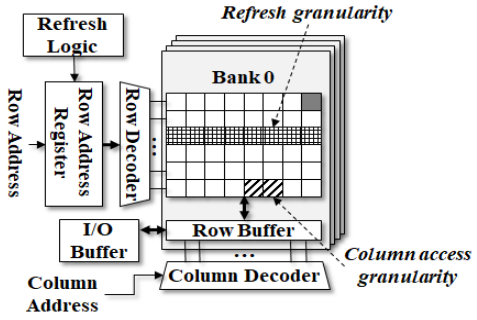


그림 3. DRAM Bank 구조
Fig. 3. DRAM Bank Architecture

그림 3은 DRAM 뱅크 내부의 구조를 나타낸다. 하나의 뱅크는 DRAM 셀들을 여러 개의 배열 형태로 포함하고 있으며 이 배열 하나가 로우가 된다. 로우는 뱅크 내에서 접근되는 데이터의 최소 크기이다. 따라서 DRAM의 (채널, 랭크, 뱅크, 로우)의 주소 값이 주어지면 이를 통해 뱅크 내의 로우의 값이 읽혀지게 되고, 이 값은 로우 버퍼(row buffer)로 전송된다. 이때 칼럼 주소를 통해 로우 데이터 중 64-byte만 I/O 버퍼를 통해 전달된다.

그림 4는 리프레시를 수행하는 리프레시 로직 구조를 나타낸다. 리프레시는 리프레시 컨트롤러에 의하여 매 64㎳마다(또는 32㎳마다) 발생된다. 리프레시를 수행하는 방법은 여러 가지가 있지만, 가장 간단한 구조는 일단 한번 리프레시가 발생하면 뱅크 내 첫 번째 로우부터 마지막 로우까지 연속적으로 리프레시를 수행하는 것이다. 이 때 컨트롤러는 로우와 칼럼의 접근 타이밍을 결정하는 시그널인 row address strobe (RAS) 및 column address strobe(CAS)의 값을 통해 하나의 로우에 대한 리프레시가 끝난 후 바로 다음 리프레시를 수행한다. 리프레시 카운터(refresh counter)는 하나의 로우에 대한 리프레시가 수행될 때 마다 1씩 증가하며 동시에 리프레시를 수행할 로우 주소로 사용된다.

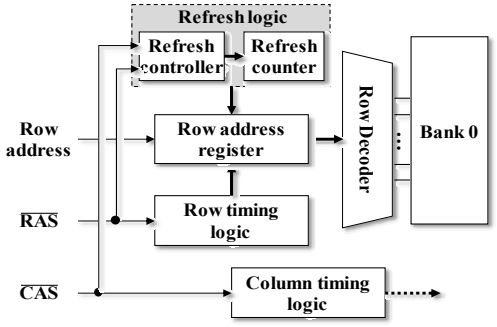


그림 4. DRAM 리프레시 로직 구조
Fig. 4. DRAM refresh logic structure

III. 제안 기법

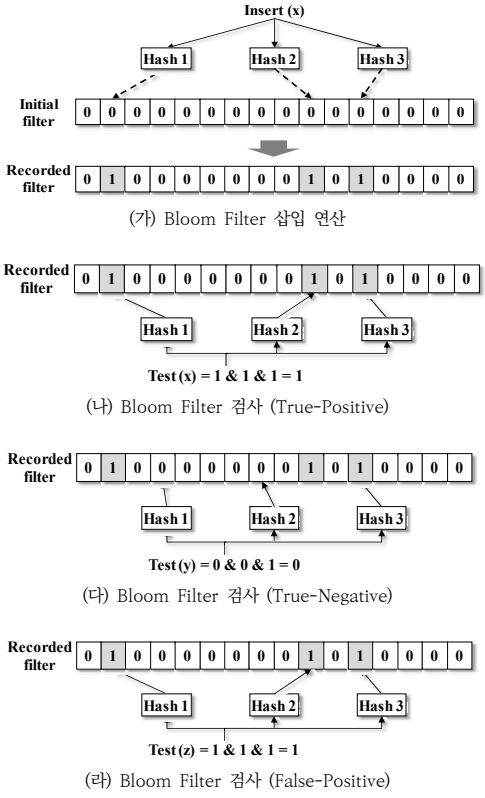
1. DRAM 셀 보존 시간

표 1에서 보는 바와 같이 99.9%의 셀들은 약 네 배의 시간인 256㎳이상 값을 보존할 수 있는 것으로 나타난다[6]. 이러한 현상은 DRAM 셀을 제조할 때 발생하는 공정 변동(process variation)에 기인한다. 따라서 1000여개의 약한 셀들에 대해서만 64㎳ 또는 128㎳ 주기로 리프레시를 수행하고 나머지 셀들에 대해서는 256㎳마다 수행하면 리프레시로 인하여 발생하는 오버헤드를 크게 절감할 수 있다. 만약 약한 셀들이 존재하는 로우의 주소를 테이블이나 리스트 형태로 저장한다면 64-bit 크기의 엔트리가 1000개나 필요하게 되므로 많은 공간 오버헤드가 발생하게 되며, 더욱이 매 리프레시마다 엔트리들을 전부 비교해야 하므로 성능 손실도 커지게 될 것이다. 따라서 제안 기법에서는 블룸 필터를 사용하여 약한 셀을 포함하고 있는 로우의 위치를 기록한다.

표 1. DRAM 셀 보존 시간 분포
Table 1. DRAM cell retention time distribution

Retention time	Number of rows
64ms-128ms	28
128ms-256ms	978
256ms 〈	523282

2. 블룸 필터



(가) Bloom Filter 삽입 연산

(나) Bloom Filter 검사 (True-Positive)

(다) Bloom Filter 검사 (True-Negative)

(라) Bloom Filter 검사 (False-Positive)

그림 5. Bloom Filter 설정 및 검사
Fig. 5. Bloom Filter Setting and Inspection

블룸 필터는 원소가 집합에 속하는지 여부를 검사하는데 사용되는 확률적 자료구조이다[7]. 블룸 필터의 특징은 원소가 집합에 속한다고 판별되었으나 실제로는 집합에 속하지 않는 false-positive 오류는 발생할 수 있으나, 반대로 집합에 속하지 않는다고 판별되었는데 실제로는 집합에 속하는 true-negative 오류는 발생하지 않는다는 점이다. 블룸 필터는 보존 시간이 긴 DRAM 로우를 자주 리프레시하는 것은 문제가 되지 않으나, 보존 시간이 짧은 로우를 리프레시 하지 않는 것은 문제가 되기 때문에 제안 기법에서 사용하기에 적합하다.

그림 5는 하나의 블룸 필터와 특정 DRAM 주소 x, y, 그리고 z를 통한 예시를 나타낸다. 또한 오직 주소 x만 약한 셀을 포함하고 있다고 가정한다. 그림 5(가)와 같이 최초의 필터(initial filter) 값은 전부 0이라고 가정한다. 이 상태에서 DRAM 주소 x에 대하여 서로 다른 세 개의 해시함수(hash function)를 수행하여 얻은 값을 필터에 기록한다(recorded filter). 공정 변동으로 인한 약한 셀의 위치는 무작위 결과에 가까우므로 해시 함수로는 xor-shift pseudo random number generator[5][8]를 사용하였다.

그림 5(나)는 기록된 필터를 통하여 DRAM 주소 x가 약한 셀을 포함하고 있는지를 검사하는 과정이다. 이 예제에서 이전과 동일한 세 개의 해시 함수에 대하여 주소 x를 적용시킨 후, 그 결과를 기록된 필터의 값과 AND 연산하면 값은 true가 나오게 된다(true-positive). 따라서 주소 x에 해당하는 로우는 자주 리프레시를 수행해야 하는 것으로 판단할 수 있다. 그림 5(다)는 DRAM 주소 y에 대하여 블룸 필터를 적용한 결과를 나타낸다. 주소 y에 세 해시 함수를 수행한 결과와 필터의 비교 결과가 false 이므로 주소 y에 해당하는 로우는 보존 시간이 긴 셀들로 구성된 로우라고 판단할 수 있다(true-negative). 그림 5(라) 역시 주소 z에 대하여 블룸 필터 검사를 수행하는 과정을 나타낸다. 비록 주소 z에 해당하는 로우 내에 약한 셀이 존재하지 않음에도 불구하고 세 개의 해시 함수 결과가 필터의 값과 일치할 수 있는데, 이는 해시의 특성상 서로 다른 입력 값이 동일한 해시 값을 도출하는 해시 충돌(hash collision)을 발생시킬 수 있기 때문이다. 따라서 이 경우 주소 z는 실제로는 약한 셀을 포함하고 있는 로우의 주소가 아니지만, 약한 셀이 존재한다고 판별되는 false-positive가 발생한다. 그럼에도 불구하고 잘못된 판별로 인하여 자주 리프레시를 하는 것은 기능상 문제가 되지 않는다.

3. 제안 리프레시 아키텍처

그림 6은 블룸 필터를 사용하여 선택적 리프레시를 수행하는 제안 아키텍처를 나타낸다. 기존 리프레시 로직에 대하여 필터링 로직(filtering logic)이 추가되었다. 이 필터링 로직은 각 뱅크마다 하나씩 존재하며, 하나의 필터링 로직 내에는 두 개의 블룸 필터가 존재한다. 블룸 필터들은 각각 64ms마다 리프레시를 수행해야 할 로우의 위치(bloom-64)와 128ms마다 리프레시를 수행할 로우의 위치(bloom-128)를 따로 가지고 있다. 리프레시 컨

트롤러와 리프레시 카운터는 기존의 구조와 같이 매 64㎳마다 리프레시 수행을 시도한다. 그러나 실제 리프레시의 실행은 네 번의 사이클에 따라 다르게 처리된다. 첫 번째 64㎳에는(첫 번째 사이클, 64㎳) 리프레시 카운터로부터 전달된 주소를 bloom-64와 비교한 후 해당되는 주소에 대해서만 리프레시를 수행한다. 만약 블룸 필터를 통한 필터링 결과는 MUX의 컨트롤 신호로 사용되게 되며, 만약 해당 주소가 필터에 속하지 않는다면 필터링 로직은 disable 신호를 발생시켜 리프레시를 수행하지 않는다. 그 다음 64㎳에서는(두 번째 사이클, 128㎳) bloom-64, bloom-128 중 하나에 속하는 주소에 대해서만 리프레시를 수행한다. 그 다음 64㎳ (세 번째 사이클, 192㎳)에는 bloom-64에 속한 주소에 대해서만 리프레시를 수행한다. 마지막인 그 다음 64㎳에서는 (네 번째 사이클, 256㎳) 필터링 결과와 상관없이 모든 주소에 대하여 리프레시를 수행한다. 이와 같은 방법을 사용하면 전체 메모리 주소에 대해서 네 번 중 세 번의 불필요한 리프레시를 효과적으로 제거할 수 있다.

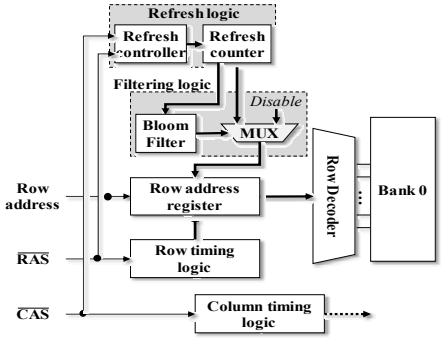


그림 6. 제안 리프레시 아키텍처
Fig. 6. Proposed Refresh Architecture

IV. 실험 결과

1. 실험 방법

제안 기법의 효능을 검증하기 위해 Instumentation 기반 시뮬레이터인 Snipersim[9] 및 cycle-accurate 메모리 시뮬레이터인 USIMM[10]을 사용하였다. Snipersim을 통하여 SPEC 2006[11] 벤치마크들 중 메모리 중심의 벤치마크 조합에 대해 트레이스를 생성하였고, 이를 USIMM의 입력으로 사용하여 제안 기법의 효율성을 평가하였다. 메모리 용량은 4GB이며 채널, 랭크, 그리고 뱅크의 수는 각각 1개, 2개, 그리고 8개이다. 또한 하나의 뱅크는 32K개의 로우를 포함하고 있다. 표 2는 이러한 시뮬레이터의 전체 구성을 나타낸다.

표 2. 시뮬레이터 설정
Table 2. Simulator Settings

Configuration	Value
Number of cores	4
Processor clock speed	3.2GHz
Processor ROB size	128
Processor fetch/retire width	4
Last level cache (shared)	64MB, 8-way, 64B lines
Memory size	4GB
Memory bus speed	800MHz
Banks, Ranks, Channels	8, 2, 1
Rows per bank	32K
Cache lines per row	128
Page allocation policy	Random

2. 성능 평가

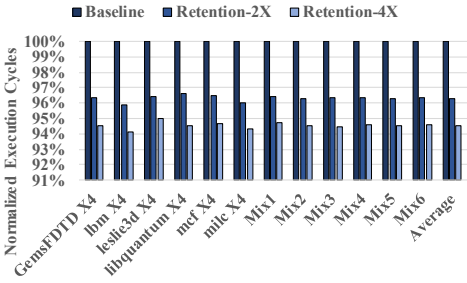


그림 7. 정량화된 수행 사이클
Fig. 7. Normalized Execution Cycles

그림 7은 기존의 리프레시 방식과 제안 기법을 적용한 방식 간의 성능 차이인 정량화된 수행 사이클을 나타낸다. Retention-2X는 64㎳의 두 배인 128㎳마다 전체 리프레시를 수행하는 구조를 나타내며, Retention-4X는 네 배인 256㎳마다 전체 리프레시를 수행하는 구조를 나타낸다. 참고로 Retention-2X의 경우 아키텍처 상에서 64㎳마다 리프레시를 수행할 주소를 포함하는 블룸 필터 하나만 존재한다. 벤치마크의 종류와 조합 방식에 상관없이 대부분의 경우 비슷한 성능이 향상되었다. Retention-2X의 경우 평균 3.7%, Retention-4X의 경우 평균 5.5%의 성능이 각각 향상되었다. 이는 프로세서로부터 요청된 DRAM 읽기/쓰기 동작이 리프레시 동작 중에는 지연되던 현상의 해소로 인하여 평균 서비스 latency가 감소한 것에 기인한다.

3. 리프레시 파워/에너지 평가

그림 8은 기존 구조 및 제안 기법에 따른 평균 리프레시 파워 소모를 나타낸다. 비록 제안 구조가 일부 로우들에 대하여 주기적으로 리프레시를 수행한다고 하더라도 이들의 수는 전체 로우들 대비 0.1% 수준이기 때문에 평균 파워 소모는 큰 폭으로 감소하였다. Retention-2X에서는 약 50%, Retention-4X에서는 약 75% 가량 감소하였다. 참고로 블룸 필터는 수 비트의 SRAM 셀만으로 표현 가능하며, 비교 연산은 비트 당 하나의 comparator만 소모되므로 이에 대한 파워 소모는 무시할 만한 수준이라고 가정하였다.

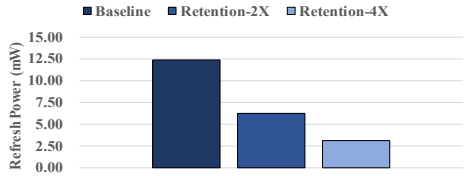


그림 8. 평균 리프레시 파워
Fig. 8. Average Refresh Power

그림 9는 벤치마크 별 리프레시 에너지 소모를 나타낸다. 소모 파워의 감소와 함께 수행 속도가 증가하였기 때문에 에너지 소모는 소폭의 추가 감소가 발생하였다. Retention-2X의 경우 약 51.8%가 감소하였으며, Retention-4X의 경우 76.4% 가량 감소하였다.

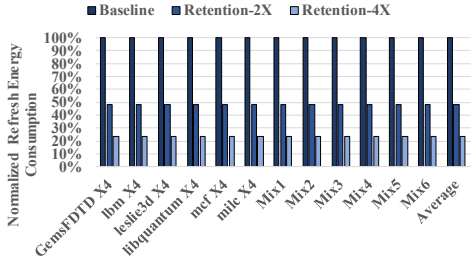


그림 9. 정량화된 에너지 소모
Fig. 9. Normalized Refresh Energy Consumption

4. EDP 평가

그림 10은 제안 기법이 적용된 시스템에 대한 정량화된 Energy-Delay Product (EDP)를 나타낸다. 산출에는 메인 메모리의 파워 소모 및 예상 프로세서 소모 파워, 그리고 기타 주변 장치들로 인한 파워 소모가 포함되었다. Retention-2X의 경우 약 7%의 향상이 존재하였으며, Retention-4X의 경우 약 10.3%의 EDP 향상이 발생하였다.

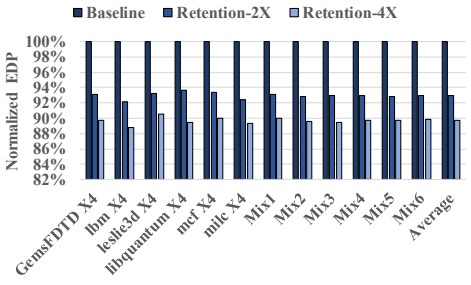


그림 10. 정량화된 EDP
Fig. 10. Normalized EDP

V. 결 론

DRAM은 현대의 메인 메모리 시스템을 구성하는 주요한 요소이지만 주기적으로 저장된 값을 읽은 후 다시 저장하는 리프레시 동작으로 인한 성능 및 파워/에너지 오버헤드가 존재한다. DRAM의 리프레시 주기는 전하의 보존 시간이 가장 낮은 약한 셀에 의해 결정되지만, 선행 연구결과들에 따르면 대부분의 셀들이 훨씬 더 오랜 시간 동안 일정 수준 이상의 전하량을 보존할 수 있는 것으로 나타났다.

따라서 본 연구에서는 블룸 필터를 사용하여 약한 셀들에 대해서만 선택적으로 자주 리프레시를 수행하고 이외의 셀들에 대해서는 더 긴 주기에 따라 리프레시를 수행하는 에너지 효율 향상 기법을 제안하였다. 실험 결과 제안 기법을 통하여 평균 5.5%의 성능 향상이 있었으며, 리프레시 에너지는 평균 76.4% 절감되었고, 평균 EDP는 10.3% 절감된 것으로 나타났다.

References

[1] Kim, Joohee, and Marios C. Papaefthymiou, "Dynamic memory design for low data-retention power", International Workshop on Power and Timing Modeling, Optimization and Simulation. Springer, Berlin, Heidelberg, pp. 207-216, Sep. 2000.
DOI: https://doi.org/10.1007/3-540-45373-3_22

[2] Kim, Joohee, and Marios C. Papaefthymiou, "Block-based multiperiod dynamic memory design for low data-retention power", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 11, No. 6, pp. 1006-1018, Dec 2003.
DOI: https://doi.org/10.1109/TVLSI.2003.817524

[3] Ghosh, Mrinmoy, and Hsien-Hsin S. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs", Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 134-145, Dec 2007.
DOI: https://doi.org/10.1109/MICRO.2007.4408251

[4] Emma, Philip G., William R. Reohr, and Mesut Meterelliyoz, "Rethinking refresh: Increasing availability and reducing power in DRAM for cache applications", IEEE Micro, Vol. 28, No. 6, pp. 47-56, Dec 2008.
DOI: https://doi.org/10.1109/MM.2008.93

[5] Liu, Jamie, et al., "RAIDR: Retention-aware intelligent DRAM refresh" ACM SIGARCH Computer Architecture News. Vol. 40, No. 3, Jun 2012.
DOI: https://doi.org/10.1145/2366231.2337161

[6] Xusheng Zhan, Yungang Bao, Ninghui Sun, "DearDRAM: Discard Weak Rows for Reducing DRAM's Refresh Overhead", 12th Conference on Advanced Computer Architecture, pp. 109-114, Aug 2018.
DOI: https://doi.org/10.1007/978-981-13-2423-9_9

[7] Burton H. Bloom, "Space/time trade-offs in hash coding with allowable errors", The Journal of Communications of the ACM, Vol. 13, No. 7, pp. 422-426, Jul 1970.

[8] Marsaglia, George. "Xorshift rngs", The Journal of Statistical Software, Vol. 8, No. 14, pp. 1-6, 2003.
DOI: https://doi.org/10.18637/jss.v008.i14

[9] Carlson, Trevor E., Wim Heirmant, and Lieven Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation", Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1-12, Nov 2011.
DOI: https://doi.org/10.1145/2063384.2063454

[10] Chatterjee, Niladrish, et al., "Usimm: the utah simulated memory module", University of Utah, Tech. Rep pp. 1-24, 2012.

[11] Standard Performance Evaluation Corporation, "SPEC CPU2006", 2006.
http://www.spec.org/cpu2006/

저 자 소 개

김 영 웅(정회원)

- 1993년 KAIST 전산학과 박사
- 1984 ~ 1997년 KT 통신망연구소
- 1997년 ~ 현재 한성대학교 컴퓨터공학부 교수
- 주관심분야 : 시스템 신뢰도, 소프트웨어 설계, 임베디드 시스템

※ 본 연구는 한성대학교 교내연구비 지원 과제임.