Journal of KIIT. Vol. 16, No. 10, pp. 27-35, Oct. 31, 2018. pISSN 1598-8619, eISSN 2093-7571 27
http://dx.doi.org/10.14801/jkiit.2018.16.10.27

# DRAM의 성능 향상을 위한 Pre-Refresh 기법

조민신*[1], 한경호*[2], 장우영**

# A Pre-Refresh Technique for Improving DRAM Performance

Minshin Cho*[1], Kyong-Ho Han*[2], and Wooyoung Jang**

이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2017R1D1A1B03036353).

## 요 약

DRAM(Dynamic Random Access Memory) 지연 시간은 감지 증폭기(Sense-Amplifier)가 DRAM 셀(Cell)의 전하를 탐지하는 행 활성화(Row Activation) 시간에 좌우된다. 만약 DRAM 셀이 전하들로 완전히 충전되어 있다면, 감지 증폭기는 그 전하들을 신속하게 탐지할 수 있어서, 행 활성화 시간을 줄일 수 있을 것이다. 본 논문에서는 그러한 DRAM의 행 활성화 시간을 줄이기 위한 사전 리프레시(Pre-Refresh) 기법을 제안한다. 사전 리프레시 기술은 DRAM이 유휴 상태일 때, 접근될 것으로 예상되는 행의 모든 DRAM 셀들을 충전시킨다. 만약, 그 행의 어떤 DRAM 셀들이 곧 읽혀지거나 기록된다면, DRAM의 행 활성화 시간을 줄일 수 있다. 실험 결과에서, 사전 리프레시가 적용된 메모리 컨트롤러는 쿼드-코어 프로세서가 네 개의 PARSEC 벤치마크를 동시에 수행하는 경우, 5.49% 전력 소비를 감소시키며, 11.8%의 프로그램 실행 시간을 향상시킨다.

## Abstract

Dynamic random access memory (DRAM) latency depends on the row activation time required to detect the charge of a DRAM cell. If a DRAM cell is fully charged, the sense amplifier can rapidly detect charges, thus reduce the row activation time. In this paper, we propose a pre-refresh technique to reduce the row activation time of DRAM. The pre-refresh technique charges DRAM cells in a row that are expected to be accessed while DRAM is idle. If any DRAM cells in the row are read or written soon, the row activation time can be reduced. Experimental results show that, when the pre-refreshed quad-core processor performs PARSEC benchmarks, a memory system with the pre-refresh technique causes 5.49% less power consumption, and achieves 11.8% faster program execution time.

Keywords
DRAM, refresh, pre-refresh

* 단국대학교 전자전기공학과
- ORCID[1]: http://orcid.org/0000-0003-1709-7614
- ORCID[2]: http://orcid.org/0000-0002-0260-2107
** 단국대학교 전자전기공학과 교수(교신저자)
- ORCID: http://orcid.org/0000-0002-7262-0028

· Received: Jul. 23, 2018, Revised: Aug. 31, 2018, Accepted: Sep. 03, 2018
· Corresponding Author: Wooyoung Jang
Dept. of Electronics and Electrical Engineering, Dankook University
Yongin-si, Geonggi-do, Korea,
Tel.: +82-31-8005-3620, Email: wyjang@dankook.ac.kr

## Ⅰ. 서 론

반도체 기술의 발전에 따라, 최신 컴퓨팅 시스템은 하나의 다이(Die)에서 향상된 성능과 확장된 기능을 제공하기 위하여 멀티-코어 프로세서(Multi-Core Processor)를 내재한다[1]. 각 코어는 개별적인 캐시(Cache)를 가지고 있으며, LLC(Last-Level Cache)를 공유한다. 이러한 계층적인 캐시 구조로 프로그램 실행에 필요한 데이터와 명령들이 신속하게 각 코어로 제공된다. 그러나, 다수의 응용 프로그램들이 멀티-코어 프로세서에서 동시에 수행되는 경우, LLC의 용량이 충분하지 않다면, LLC에서 캐시 실패(Miss)가 자주 발생하게 되어, 주 메모리인 DRAM (Dynamic Random Access Memory)에 자주 접근하게 된다[2]. DRAM에 접근하는 동안, 그 프로세서는 데이터나 명령이 도착할 때까지 계속 정지한다. 따라서 DRAM의 지연시간을 최소화할 수 있다면, 프로그램의 실행시간은 크게 단축될 수 있을 것이다[3].

DRAM의 지연시간은 행 활성화 시간($t_{RCD}$), 열 접근 시간($t_{CL}$), 행 비활성화 시간($t_{RP}$))과 같은 타이밍 매개변수들로 결정된다[4]. 행을 구성하는 DRAM 셀들은 행 활성화 시간동안 전하들을 비트라인(Bit-Line)과 공유된다. 각 셀들은 충전 여부에 따라, 논리 1 혹은 0을 저장한다. 감지 증폭기(Sense-Amplifier)는 셀과 비트라인의 전하를 주입하거나 뽑아내고, 셀과 비트라인의 전압을 측정하여, 각 셀의 값을 탐지한다. 이러한 행 활성화 동작의 지연시간은 DRAM 셀의 전하량에 따라 변화될 수 있다. 예를 들어, 셀에 전하가 완전히 충전되어 있다면, 그 셀의 전하가 비트라인에 공유된 후, 전압이 더 높다. 따라서 감지 증폭기가 전하를 주입하는 시간이 짧아지게 되며, 기존의 $t_{RCD}$보다 빠른 시간에 데이터 전송을 시작할 수 있다[5][6]. 본 논문에서는 이런 사실에 근거하여, 사전 리프레시(Pre-Refresh) 기법을 제안한다. DRAM은 유휴 상태일 때, 사전 리프레시를 활용하여 접근될 것으로 예상되는 행의 셀들을 충전시킨다. 따라서 LLC가 DRAM에 요청들을 전송했을 때, DRAM은 더 짧은 지연시간으로 데이터나 명령을 제공할 수 있다.

그러나 사전 리프레시는 즉시 제공해야할 명령이나 데이터를 가진 DRAM 셀에 접근하는 것이 아니므로, 긴급한 요구 요청(Demand Request)과 충돌을 피해야 한다. 이를 위해서 제안된 사전 리프레시 기법은 ITCG(Idle Time Counter Group)과 HCRAC (Highly-Charged Row Address Cache)을 사용하여, 사전 리프레시 수행이 긴급한 요구 요청과 충돌하는 것을 최소화 한다.

본 논문은 다음과 같이 구성된다. 2장은 DRAM의 구조와 동작에 대해서 설명하며, 관련된 연구들을 기술한다. 3장에서는 제안하는 사전 리프레시 기법을 상세하게 설명한다. 4장에서는 구현방법과 실험 결과를 분석한다. 마지막으로 5장에서는 본 논문의 결론을 도출한다.

## Ⅱ. 관련 연구

DRAM은 단위 용량 당 비용이 SRAM보다 적기 때문에, 최신 컴퓨팅 시스템에서 주 메모리(Main Memory)로 사용된다[3]. 하지만, SRAM보다 지연시간이 큰 것이 단점이다. 따라서 이전 연구들에서 DRAM의 지연시간을 줄이기 위한 다양한 방법이 제안되었다. 먼저, 현재 사용되는 DRAM의 구조를 변경하는 방법들이 제안되었다. 이들 논문에서는 DRAM 구조 변경을 통해 비트라인의 길이를 변화시키며, 이를 통해 유동적인 $t_{RCD}$을 적용하였다[7]. 또한, $t_{RCD}$를 줄이기 위한 방법으로 셀의 초기 전하량을 고려하는 방법을 제안하였다[5][6].

DRAM에서 리프레시로 인한 성능 지연을 줄이는 방법들이 활발히 연구되었다. 리프레시 동안, DRAM이 각 뱅크의 요청을 수행하지 못하는 문제를 해결하기 위해, [8]은 JEDEC DDRx(Double Data Rate) 표준에서 제공하는 리프레시 지연을 활용하여, 자동-리프레시를 유연하게 시행하며 이를 통해 요구 요청의 수행이 지연되지 않도록 한다. [9]는 LPDDRx(Low Power DDRx)의 뱅크 단위 리프레시를 기반으로, DDRx에서도 하나의 뱅크가 리프레시되는 동안 DRAM 요청을 병렬적으로 수행하는 것을 제안했다. [10]에서는 리프레시의 매개변수 $t_{RFC}$를 줄이는 방법을 제안했다. DRAM의 내부 구조를 변경하여, 여러 개의 행이 병렬적으로 리프레시 되

어, 리프레시 지연시간을 줄이려 하였다.

리프레시 지연을 줄이기 위하여 행-단위 리프레시(Row-Level Refresh)들이 연구되고 있다. 몇몇 연구 [11], [12]에서는, 대부분의 DRAM 셀들은 64ms 이상 데이터 보유 시간을 가지고, 일부 셀들만 데이터 보유 시간이 짧다는 사실을 보여준다. 이를 근거로, 취약 셀(Weak Cell)이라 명명된 보유시간이 짧은 셀을 포함한 행만 집중적으로 리프레시하면, DRAM의 성능과 에너지 개선에 크게 기여할 수 있다는 점을 이용하였다. 이 연구들을 기반으로, [13]에서는 DRAM에서 사용하는 자동-리프레시 방법을 그대로 이용하면서, 취약 셀만 정확한 리프레시 주기를 지키고, 나머지는 시간이 더 지난 후, 리프레시 하는 방법을 제안하였다.

최신 컴퓨팅 시스템에서, 프로세서와 DRAM의 동작 클럭 주파수의 차이는 크다. 따라서 프로세서가 DRAM을 접근할 때마다, DRAM 지연시간을 줄이는 것이 중요하다. 따라서 본 논문에서는 유동적인 $t_{RCD}$와 행-단위 리프레시를 기반으로 DRAM을 접근할 때, 지연시간을 최소로 하는 사전 리프레시 기법을 제안한다.

## III. 사전 리프레시 기법

3.1절에서는 행-단위 리프레시에 대해서 설명한다. 3.2절에서는 사전 리프레시 기법을 제안하고, 메모리 컨트롤러(Controller)의 스케쥴링(Scheduling) 정책과 이를 지원할 두 하드웨어 집단 HCRAC와 ITCG를 설명한다.

### 3.1 행-단위 리프레시

그림 1은 행-단위 리프레시의 기본적인 동작을 보여준다. ①에 나타난 취약-행의 셀은 부분적으로 충전된 상태(Partially Charged)이고, 그 외의 다른 행들은 충전(Fully Charged) 혹은 방전(Discharged) 상태이다. 메모리 컨트롤러는 이 행이 취약 행이라는 것을 확인하고, 행 활성화 명령을 전송한다. 셀에 저장된 전하가 비트라인에 공유되고, 감지 증폭기는 요구 요청(Demand Request)을 수행하는 것처럼 양쪽에 전하를 주입하여 전압을 증가시킨다(②). 전압이 기준 전하량(Threshold)에 도달했다면 읽기 명령을 통해 데이터 전송을 수행할 수 있지만, 행-단위 리프레시는 데이터 전송이 아닌 전하의 충전이 목적이므로 메모리 컨트롤러는 읽기 또는 쓰기 명령을 이 행으로 전송하지 않는다(③). 따라서 $t_{RAS}$ (Row Access Strobe)까지 감지증폭기는 계속 전하를 주입하여 셀의 전하를 완전히 충전시킨다(④). 여기서, $t_{RAS}$는 행 비활성화 명령이 행 활성화 명령 이후에 전송될 수 있는 최소 시간이다. 메모리 컨트롤러는 행 비활성화 명령을 전송해서 $t_{RP}$ 시간 동안 행을 비활성화 시킨다(⑤).

행-단위 리프레시는 하나의 목표 행의 셀을 충전시키기 위해 항상 $t_{RAS}$+$t_{RP}$ 사이클이 소모 된다. 하나의 자동-리프레시 명령이 전송되는 간격인 7.8us 마다 각 뱅크의 취약 행의 수가 한 번의 자동-리프레시 당 충전되는 행의 수보다 적다면, 행-단위 리프레시는 이득을 얻을 수 있지만, 반대의 경우에는 더 많은 지연 시간이 야기된다.

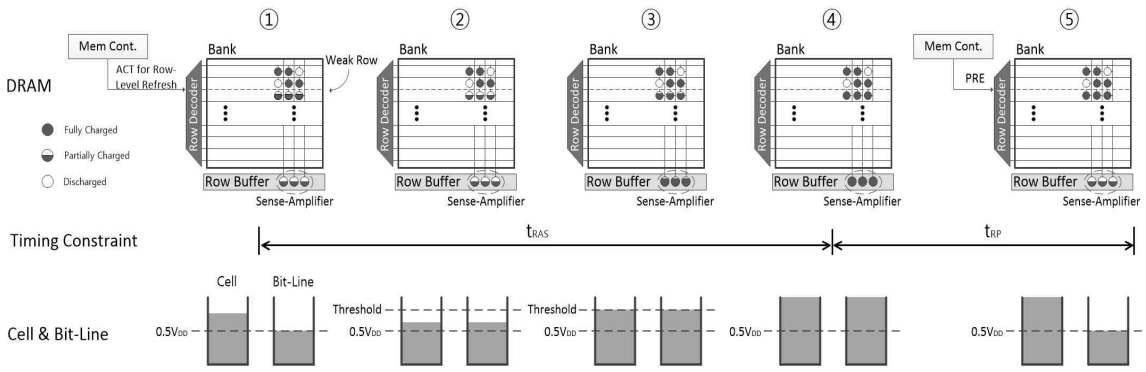



그림 1. 행-단위 리프레시의 동작 과정

Fig. 1. Row-level refresh operation

특히, 최근의 DRAM은 뱅크 당 행의 수가 늘어나고 있으며, 이와 비례해서 뱅크 내의 취약 행의 수도 증가할 수 있다. 따라서 행-단위 리프레시는 저성능과 고전력 소모를 유발하여, 최신 DRAM에 적용하는 것은 많은 제약이 있다.

## 3.2 사전 리프레시

사전 리프레시 기법을 활용하기 위해서는 프로세서의 하드웨어 프리페처의 지원이 필요하다. 본 논문은 선행 연구에서 제안된 명령 프리페처(Instruction Prefetcher)[16]를 수정하여 활용한다. 수정된 프리페처는 예측된 명령의 주소가 PC(Program Counter)와 PC보다 큰 가장 가까운 분기 명령(Branch Instruction)의 주소 사이이면, 일반적인 명령 프리페처와 동일하게 프리페치 요청을 하고, 그렇지 않으면, 사전 리프레시를 요청한다. 사전 리프레시 요청을 다른 요청들과 구분하는 방법으로 프로세서에서 생성되는 모든 요청에 대해 사전 리프레시 플래그(Pre-Refresh Flag)라 불리는 하나의 비트가 추가된다. 프리페처는 사전 리프레시를 요청할 때마다, 사전 리프레시 플래그를 1로 변경한다.

그림 2는 제안하는 사전 리프레시 기법을 위한 메모리 컨트롤러이다. 그 메모리 컨트롤러는 사전 리프레시 요청을 받아 그것의 뱅크와 행 주소를 사전 리프레시 버퍼(PRB, Pre-Refresh Buffer)에 저장한다. 일반적으로, 메모리 컨트롤러는 LLC로부터 요구 요청을 받고, 목표 뱅크로 전송되기 전까지 저장하는 요청 버퍼(Request Buffer)를 가지고 있다. 사전 리프레시 기법은 이 요청 버퍼 앞에 사전 리프레시 디코더(Decoder)를 추가하여, 요청들이 나타내는 사전 리프레시 플래그를 확인한다. 만약 어떤 요청의 사전 리프레시 플래그가 1이라면, 이것은 사전 리프레시 요청이므로, PRB에 따로 저장된다. 반대로, 어떤 요청의 사전 리프레시 플래그가 0이면, 이것은 요구 요청으로 요청 버퍼로 전송한다. 이 때, 메모리 컨트롤러는 PRB에 접근하여, 그 요구 요청과 같은 뱅크와 행 주소를 가지는 사전 리프레시 요청이 저장되어 있다면, PRB에서 그 항목을 폐기한다.

PRB에 저장된 요청들은 대응하는 목표 뱅크가 유휴상태가 되고, 그 뱅크의 요청 버퍼가 비어있다면, 해당 뱅크의 요청 버퍼로 삽입될 수 있다. 하지만, 무조건적인 사전 리프레시 요청의 요청 버퍼로의 삽입은 요구 요청과 충돌하여, 메모리 시스템의 성능이 저하될 수 있다. 따라서, 메모리 컨트롤러가 요청 버퍼에 사전 리프레시 요청을 이동할지를 결정할 기준이 필요하다. 이것을 위하여, 그림 2에서 모든 PRB에 저장된 사전 리프레시 요청들은 ITCG (Idle Time Counter Group)를 참조한다. ITCG는 다음과 같은 카운터들로 구성된다.

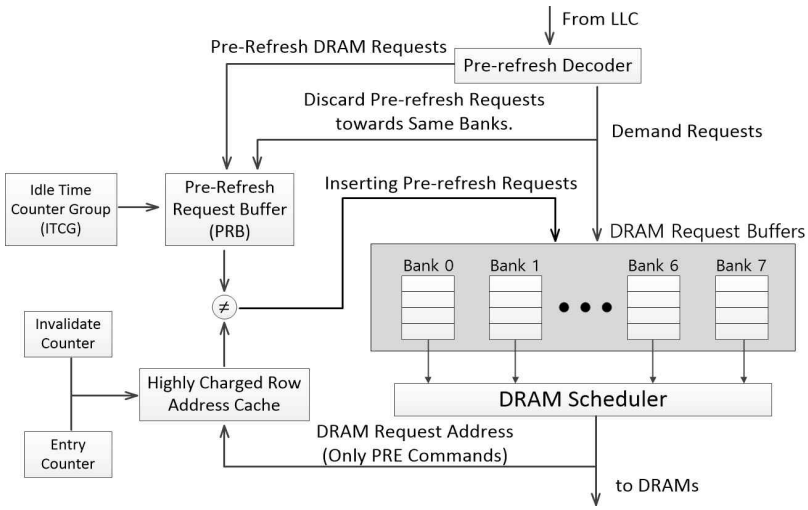



그림 2. 사전 리프레시 기법을 제공하는 메모리 컨트롤러
Fig. 2. Proposed memory controller supporting a pre-refresh technique

1) ICC(Idle Cycle Counter): ICC는 시간 윈도우(Time Window)[5]에서 각 뱅크의 유휴 시간을 측정한다. 어떤 뱅크가 요청을 처리하고 비활성화 되었다면, ICC는 그 뱅크에 다음 활성화 명령이 도달할 때까지 사이클의 수를 측정한다.

2) RAC(Row Activation Counter): RAC는 시간 윈도우에서 각 뱅크가 몇 번 활성화가 되었는지를 수집한다.

3) AICC(Average Idle Cycle Counter): AICC는 ICC를 RAC으로 나눈 값으로, 시간 윈도우의 평균 유휴 시간을 의미한다.

메모리 컨트롤러는 AICC를 참조하여, 다음 윈도우에서 PRB에 있는 사전 리프레시 요청이 요청 버퍼로 전송되는 것을 제한한다. 즉, AICC가 사전 리프레시 수행에 필요한 최소 지연 시간 ($t_{RAS}+t_{RP}$)보다 큰 경우에만 PRB에서 사전 리프레시 요청이 요청 버퍼로 전송된다. 그렇지 않으면, PRB에 저정된 사전 리프레시 요청은 다음 윈도우까지 대기하거나, 같은 뱅크와 행을 접근하는 요구 요청이 요청 버퍼에 저장될 때 폐기된다.

사전 리프레시로 활성화된 행은 비활성화 명령을 수행하지 않을 수 있다. 즉, 메모리 컨트롤러는 충전하고 있는 행의 뱅크와 같은 뱅크의 요청 버퍼에 저장된 요구 요청이 없으면, 사전 리프레시를 위한 비활성화 명령을 수행하지 않는다. LLC에서 다음 요구 요청이 도착하고, 그 요청의 목표 뱅크와 행이 사전 리프레시의 활성화 중인 행과 일치한다면, 그 요구 요청은 활성화 지연 시간 없이, 데이터 전송을 시작할 수 있게 된다.

제안된 ITCG를 활용하여 PRB의 사전 리프레시 요청을 요청 버퍼로 제한적으로 이동시켜 DRAM의 성능 향상을 기대할 수 있다. 하지만, 완전히 충전된 행을 다시 충전할 필요가 없고, 충전량에 따라 행 활성화 시간을 다르게 적용해야하기 때문에, 어떤 행이 얼마나 충전이 방전되었는지를 고려해야 한다. 우리는 HCRAC[6]를 활용하여 제안된 사전 리프레시 기법의 성능을 극대화한다.

HCRAC는 작은 캐시로 뱅크와 행 주소를 저장하고, 각 캐시 블록마다 카운터가 장치되어 있다. 어떤 행이 비활성화될 때, 그 뱅크와 행 주소가 HCRAC의 캐시 블록에 저장되고, 그 캐시 블록의 카운터가 동작된다. 메모리 컨트롤러는 요구 버퍼에 저장된 요청이 수행될 때, 그 요청의 뱅크와 행 주소가 HCRAC의 블록들과 일치하는지 확인한다. 만약 어떤 HCRAC의 블록과도 불일치(Miss)이면, 메모리 컨트롤러는 그 요구 요청에 일반적인 행 활성화 시간을 적용하고, 만약 일치(Hit)이면, 단축된 행 활성화 시간을 적용된다. 그 단축된 행 활성화 시간은 카운터의 값을 참조하여 결정될 수 있다. 반도체 공정에 따라 DRAM 셀의 방전량이 다르므로, 카운터의 값과 취약 셀의 최대 방전량을 측정하여 단축된 행 활성화 시간을 설정한다. 일반적으로 카운터의 값이 작을수록 충전된 셀들의 방전량이 크지 않으므로 행 활성화 시간을 더 단축시킬 수 있다. 반대로, 카운터의 값이 증가할수록 충전된 셀의 방전량도 증가하므로 행 활성화 시간을 단축시키기 어려울 것이다.

사전 리프레시 기법에서는, 메모리 컨트롤러가 사전 리프레시 요청을 요청 버퍼로 삽입하기 전에 항상 HCRAC의 캐시 블록과 일치하는지 확인한다. 만약 일치하는 항목이 있다면, 이 요청의 목표 뱅크와 행은 최근에 충전되어 있으므로 다시 충전시킬 필요가 없다. 따라서 제안된 메모리 컨트롤러는 이 사전 리프레시 요청을 요구 버퍼로 삽입하지 않고, 폐기한다.

## Ⅳ. 실험 결과

제안된 사전 리프레시를 활용한 DRAM의 성능 평가를 위해 MARSSx86[14]와 DRAMSim2[15] 시뮬레이터를 이용하였다. MARSSx86는 x86기반 프로세서, DRAMSim2는 1600㎒ DDR3 SDRAM을 모델링 할 수 있다. 두 시뮬레이터들은 사이클 레벨 정확도를 추구한다. 표 1은 실험에 시용된 시뮬레이터들의 설정을 보여준다. 실험에서 구현된 HCRAC는 672B 캐시로서 128개의 42bit 캐시 블록으로 구성된다. HCRAC에 저장된 뱅크의 행으로 요구 요청이 수행되면, 충전된 셀에 의하여 $t_{RCD}$와 $t_{RAS}$는 각각 4사이클과 8사이클을 절약될 수 있다.

싱글-/쿼드-코어 프로세서는 하나의 PARSEC[17]

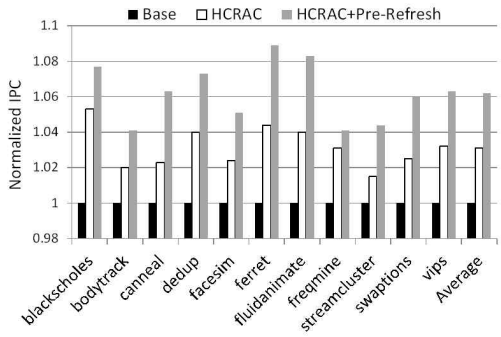
벤치마크 또는 네 개의 PARSEC 벤치마크들을 임의로 혼합하여 실행하였다. 그 프로세서들은 일반적인 메모리 컨트롤러와, HCRAC만 적용한 메모리 컨트롤러, HCRAC와 사전 리프레시를 모두 적용한 메모리 컨트롤러에 의해 지원된다. 실험은 1억 사이클 동안 수행되었다.

표 1. 프로세서와 메모리의 설정
Table 1. Configuration of processor and memory

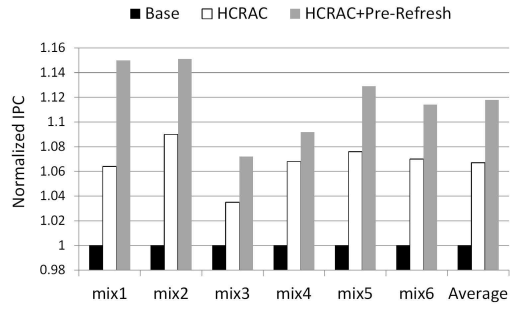
| | |
|---|---|
| Processor (MARSSX86) | 3.0GHz, quad-core, out-of-order, 4-issue per core |
| L1 I- and D-Cache | 16KB, 64B block size, 4-way associativity, write-back, pseudo LRU |
| Shared L2 Cache | 512KB, 64B block size, 4-way associativity, write-back, pseudo LRU |
| Memory Controller | address mapping(Chan:Row:Rank:Bank:Colum) FR-FCFS, 64-entry DRAM request buffer, open-page policy |
| DRAM (DRAMSim2) | 1600Mhz 11-11-11 DDR3 SDRAM, 1 channel, 2 ranks, 8 banks per rank, 32K rows, 2K columns |
| HCRAC [6] | 128-entry (672 bytes), 2-way associativity, LRU replacement policy, 1ms caching duration, $t_{RCD}$/$t_{RAS}$ reduction 4/8 cycles |

## 4.1 성능 비교

그림 3(a)는 벤치마크들이 단독으로 실행된 경우의 정규화된 IPC(Instruction Per Cycle)을 보여준다. HCRAC만 적용된 경우, 이전의 요구 요청된 뱅크의 행에 대해서만 행 활성화 시간을 단축할 수 있다. 따라서 그것은 일반적인 메모리 컨트롤러보다 평균 3.11%의 IPC 향상을 보여준다. 반면에, 사전 리프레시 기법은 곧 접근될 행의 셀까지 전하를 미리 충전시킨다. 또한, 충전이 완료된 후, 그 뱅크의 다른 행에 대한 요구 요청이 없다면, 활성화 상태를 유지시킨다. 만약 그 뱅크의 행을 접근하는 요구 요청이 전송되면, DRAM은 $t_{RCD}$ 시간이 없이 그 요구 요청의 데이터를 전송할 수 있다. 따라서 실험 결과에서는 HCRAC+사전 리프레시를 적용하였을 때, 일반적인 메모리 컨트롤러보다 평균 6.23%의 IPC 향상을 보여준다.




(a) 싱글 벤치마크 실행
(a) Executing single benchmark




(b) 네개의 벤치마크 실행
(b) Executing four benchmarks
그림 3. 성능 비교
Fig. 3. Performance comparison

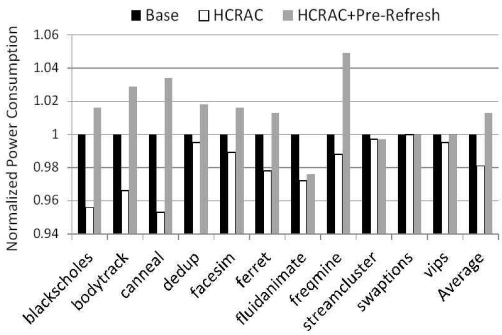
그림 3(b)는 네 개의 벤치마크들이 동시에 수행된 경우의 정규화 된 IPC 결과를 보여준다. 여러 개의 벤치마크가 동시에 수행되면, LLC에서 캐시 실패가 증가하여, DRAM이 자주 접근된다. 실험 결과에서 HCRAC만 적용한 경우는 일반적인 메모리 컨트롤러보다 평균 4.24%의 IPC 향상을 보인다. 또한, HCRAC+사전 리프레시를 사용한 경우에는 평균적으로 11.8%의 IPC 향상을 보여준다. 따라서 DRAM 접근이 많은 응용 프로그램들의 경우에, 제안된 사전 리프레시 기법은 더 향상된 성능을 보여줄 수 있다.

## 4.2 전력 소모 비교

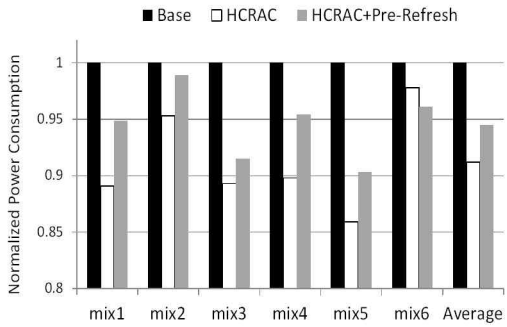
그림 4(a)는 하나의 벤치마크가 수행될 때, 정규화된 전력 소모를 보여준다. HCRAC만을 적용한 메모리 컨트롤러는 이미 충전된 행에 접근할 경우에 $t_{RAS}$가 감소되어 행이 이른 시간에 비활성화 된다. 따라서 실험에서 그것은 일반적인 메모리 컨트롤러

를 적용한 경우보다 평균 1.9%의 전력 소모가 감소되었다. 사전 리프레시 요청은 곧 접근될 것으로 예상되는 행의 DRAM 셀을 충전시키므로, 전력 소모가 더 많이 발생한다. 따라서 HCRAC+사전 리프레시는 일반적인 메모리 컨트롤러를 적용한 경우보다 평균 1.3%의 추가적인 전력 소모가 발생한다.

그림 4(b)는 네 개의 벤치마크가 동시에 수행될 때 정규화 된 전력 소모를 보여준다. HCRAC만 적용하면, 일반적인 메모리 컨트롤러를 적용한 경우보다 평균 8.8%의 전력 소모가 감소된다. HCRAC+사전 리프레시를 적용하였을 때, 일반적인 메모리 컨트롤러를 적용한 경우보다 평균적으로 5.5%의 전력 소모가 감소된다. 그 이유는 HCRAC를 적용할 때는 최근에 접근된 행에 대해서만 활성화 시간을 감소시키기 때문에 전력에 큰 이득을 보지만, 사전 리프레시와 같이 적용하였을 때에는 사전 리프레시 요청에 의해 불필요한 셀 충전에 필요한 추가적인 전력 소모가 발생할 수 있기 때문이다.




(a) 싱글 벤치마크 실행
(a) Executing single benchmark




(b) 네개의 벤치마크 실행
(b) Executing four benchmarks
그림 4. 전력 소모 비교
Fig. 4. Power consumption comparison

따라서 두 방법들을 같이 활용하였을 때는 HCRAC만 적용할 때보다 전력 소모의 이득이 감소하지만 일반적인 메모리 컨트롤러를 사용한 경우와 비교하였을 때보다 전력 소모에서 이득이 발생한다.

## Ⅴ. 결론 및 향후 과제

주 메모리로 사용되는 DRAM의 지연시간을 줄이는 것은 프로세서의 성능을 크게 향상시킬 수 있다. 본 논문에서는 DRAM의 지연시간을 줄이는 방법으로 사전 리프레시 기법을 제안하였다. 사전 리프레시 기법은 행-단위 리프레시를 기반으로 하고, 곧 접근될 것으로 예상되는 행의 셀들을 미리 충전시킨다. 충전된 셀들은 요구 요청으로 접근될 때 행 활성화 시간을 크게 감소시켜 DRAM의 지연시간을 향상시킨다. 셀들의 충전을 위해 추가적인 전력 소모를 요구하지만, 충전된 셀들에 다시 접근하는 것을 회피하여 전력 소모를 최소화 한다. 사전 리프레시 기법은 DRAM에 접근이 많을수록, 성능과 전력 소모가 크게 개선된다. 또한, 적은 하드웨어 비용이 요구되며, DRAM 구조를 변경하지 않아도 된다. 향후 연구로서, 사전 리프레시를 최적화 된 프로세서의 하드웨어 프리페처를 고려하고 있다.

## References

[1] Shen, John Paul and Mikko H. Lipasti, "Modern processor design: fundamentals of superscalar processors", Waveland Press, 2013.

[2] Tae-jin Park and Woo-young Jang, "Large-Scale Last-Level Cache Design Based on Parallel TLC STT-MRAM", JKIIT, Vol. 15, No. 12, pp. 77-89, Dec. 2017.

[3] Jacob, Bruce, Spencer Ng, and David Wang. "Memory systems: cache, DRAM, disk", Morgan Kaufmann, 2010.

[4] JEDEC DDR3 standard: https://www.jedec.org/standards-documents/docs/jesd-79-3d. [accessed: Aug. 12. 2018]

[5] Wongyu Shin, Jeongmin Yang, Jungwhan Choi,

and Lee-Sup Kim, "NUAT: A non-uniform access time memory controller", High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on. IEEE, pp. 464-465, Feb. 2014.

[6] Hassan, Hasan, et al., "ChargeCache: Reducing DRAM latency by exploiting row access locality", 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 581-593, Mar. 2016.

[7] Lee, Donghyuk, et al., "Tiered-latency DRAM: A low latency and low cost DRAM architecture", 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), pp. 615-626, Feb. 2013.

[8] Jeffrey Stuecheli, Dimitris Kaseridis, Hillery C.Hunter, and Lizy K. John, "Elastic refresh: Techniques to mitigate refresh penalties in high density memory", Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 375-384, Dec. 2010.

[9] Chang, Kevin Kai-Wei, et al., "Improving DRAM performance by parallelizing refreshes with accesses", 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 356-367, Feb. 2014.

[10] Kalyan, T. Venkata, Kasha Ravi, and Madhu Mutyam, "Scattered refresh: An alternative refresh mechanism to reduce refresh cycle time", 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 598-603, Jan. 2014.

[11] Liu, Jamie, et al. "RAIDR: Retention-aware intelligent DRAM refresh", ACM SIGARCH Computer Architecture News - ISCA '12, Vol. 40. No. 3, pp. 1-12, Jun. 2012.

[12] Venkatesan, Ravi K., Stephen Herr, and Eric Rotenberg, "Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM", High-Performance Computer Architecture, 2006. The Twelfth International Symposium on. IEEE, pp. 157-167, Mar. 2006.

[13] Bhati, Ishwar, et al., "Flexible auto-refresh: enabling scalable and energy-efficient DRAM refresh reductions", ACM SIGARCH Computer Architecture News. Vol. 43. No. 3. pp. 235-246, Jun. 2015.

[14] Patel, Avadh, Furat Afram, and Kanad Ghose, "Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors", 1st International Qemu Users' Forum. 2011.

[15] Rosenfeld, Paul, Elliott Cooper-Balis, and Bruce Jacob, "DRAMSim2: A cycle accurate memory system simulator", IEEE Computer Architecture Letters, Vol. 10, No. 1, pp. 16-19, Apr. 2011.

[16] Chen, I-Cheng K., Chih-Chieh Lee, and Trevor N. Mudge, "Instruction prefetching using branch prediction information", ICCD '97 Proceedings of the 1997 International Conference on Computer Design, pp. 593-603, Oct. 1997.

[17] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li, "The PARSEC benchmark suite: Characterization and architectural implications", Proceedings of the 17th international conference on Parallel architectures and compilation techniques, pp. 72-81, Oct. 2008.

## 저자소개

조 민 신 (Minshin Cho)

2016년 2월 : 단국대학교 전자전기공학부 (전문공학사)
2017년 ~ 현재 : 단국대학교 전자공학과 (공학석사)
관심분야 : 컴퓨터 구조, 임베디드 시스템, SoC

한 경 호 (Kyong-Ho Han)

1982년 : 서울대학교 (공학사)
1984년 : 서울대학교 전자공학과(공학석사)
1992년 : 텍사스 A&M 대학교 전자공학과(공학박사)
1984년 ~ 1985년 : 삼성 HP사업부 연구원
1985년 ~ 1987년 : 한국통신 품질보증단 전임연구원
1992년 ~ 1993년 : 한국전자통신연구소(ETRI) CDMA 단말기 개발실 선임연구원
1993년 ~ 현재 : 단국대학교 전자전기공학부 교수
관심분야 : 드론, 자율주행, ITS, F/A System, 임베디드 프로세서 응용

장 우 영 (Wooyoung Jang)

1998년 2월 : 경희대학교 전파공학과(공학사)
2000년 2월 : 연세대학교 전기컴퓨터공학과(공학석사)
2011년 5월 : 텍사스 주립 대학교 (Univ. of Texas at Austin) 전기컴퓨터공학과(공학박사)
2000년 ~ 2012년 : 삼성전자 System LSI 사업부 SoC 개발실 책임연구원
2013년 ~ 현재 : 단국대학교 전자전기공학부 조교수
관심분야 : 컴퓨터 구조, 설계 자동화, 머신러닝