JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 4, pp. 1-9, April 2020
https://doi.org/10.9708/jksci.2020.25.04.001

Dynamic Rank Subsetting with Data Compression

Seokin Hong*

*Professor, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea

[Abstract]

In this paper, we propose Dynamic Rank Subsetting (DRAS) technique that enhances the energy-efficiency and the performance of memory system through the data compression. The goal of this technique is to enable a partial chip access by storing data in a compressed format within a subset of DRAM chips. To this end, a memory rank is dynamically configured to two independent sub-ranks. When writing a data block, it is compressed with a data compression algorithm and stored in one of the two sub-ranks. To service a memory request for the compressed data, only a sub-rank is accessed, whereas, for a memory request for the uncompressed data, two sub-ranks are accessed as done in the conventional memory systems. Since DRAS technique requires minimal hardware modification, it can be used in the conventional memory systems with low hardware overheads. Through experimental evaluation with a memory simulator, we show that the proposed technique improves the performance of the memory system by 12% on average and reduces the power consumption of memory system by 24% on average.

▸ **Key words**: Memory System, DRAM, Data Compression, Performance, Power

[요 약]

본 논문에서는 데이터 압축을 통해 메모리 시스템의 에너지 효율 및 성능을 향상시키는 동적 랭크 서브세팅 기법 (Dynamic Rank Subsetting, DRAS)을 제안한다. DRAS 기법은 하나의 메모리 랭크 (Rank)를 두 개의 서브랭크 (Sub-rank)로 동작되도록 하여, 데이터가 절반 크기로 압축될 경우 압축된 데이터를 하나의 서브랭크에만 저장한다. 이를 통해 DRAS 기법은 압축된 데이터에 대한 읽기 및 쓰기 동작의 메모리 대역폭을 두 배로 높일 수 있고, 동적 전력 소모도 절반으로 감소시킬 수 있다. 만약 데이터가 절반 크기로 압축되지 않는다면 기존 메모리 시스템에서와 같이 데이터를 두 서브랭크에 저장한다. 따라서 DRAS 기법은 데이터가 압축되지 않는 경우에 대해서는 기존 메모리 시스템 수준의 메모리 대역폭과 전력 효율성을 보장한다. 메모리 시뮬레이터를 사용한 실험 평가를 통해 DRAS 기법이 컴퓨터 시스템 성능을 평균 12% 향상시키고 메모리 시스템의 전력소모를 평균 24% 감소시킬 수 있음을 보인다.

▸ **주제어**: 메모리 시스템, DRAM, 데이터 압축, 성능, 파워 소모

• First Author: Seokin Hong, Corresponding Author: Seokin Hong
*Seokin Hong (seokin@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
• Received: 2020. 03. 23, Revised: 2020. 04. 06, Accepted: 2020. 04. 16.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

오늘날의 컴퓨팅 시스템에서 고속 및 대용량 메모리에 대한 수요가 증가함에 따라 메모리 시스템은 서버 및 데이터 센터 에너지 소모의 상당 부분을 차지한다 [1, 2]. 메모리 시스템은 일반적으로 비트 당 비용(Bit per Cost)를 최소화하도록 설계되며, 이는 높은 에너지 소비와 같은 높은 운영비용을 초래할 수 있다.

각 DIMM(Dual Inline Memory Modules)은 여러 개의 메모리 칩으로 구성된다. 단일 칩의 I/O 버스 폭(4, 8 또는 16 bits)은 제한되어 있기 때문에, 여러 칩으로 하나의 랭크(Rank)를 형성하여 64-bit I/O 버스를 구성한다. 이때, 하나의 랭크를 구성하는 여러 메모리 칩은 하나의 디바이스처럼 동작하게 된다. 메모리와 CPU 간 데이터 전송 단위인 블록 (Block)의 크기는 64 Byte로 랭크의 I/O 버스 폭보다 크다. 따라서 여러 사이클에 걸쳐 하나의 블록을 전송하게 된다. 이때 전송된 블록(64 Byte)는 랭크의 모든 칩에 분산되어 저장되게 된다. 이로인해 모든 메모리 칩이 물리적으로 분리되어 있어도 단일 디바이스처럼 작동하므로 상당한 에너지를 소모하게 된다.

이러한 비효율성을 극복하기 위해 랭크 서브세팅(Rank subsetting) 방법[3, 4, 5, 6]이 제안되었다. 랭크 서브세팅에서는 하나의 랭크를 구성하는 칩을 여러 서브랭크(Sub-rank)로 나누어 서브랭크 단위로 액세스함으로써 메모리 시스템의 에너지 소모를 감소시키고 대역폭을 향상시킨다. 랭크 서브세팅 방법에서는 각 서브랭크의 뱅크(Bank)를 독립적으로 제어할 수 있으므로 사용 가능한 뱅크 수가 증가하여 메모리 뱅크 레벨 병렬성 (Bank-level Parallelism)을 향상시킬 수 있다. 하지만, 랭크 서브세팅에서는 하나의 데이터 블록을 기존 랭크에 비해 적은 수의 I/O 핀을 갖는 서브랭크로 전송되기 때문에 메모리 접근 지연시간이 증가하게 된다.

랭크 서브세팅의 접근 지연시간 증가로 인한 시스템 성능 감소 문제를 해결하기 위해 Memzip 기술[7]이 제안되었다. Memzip 기술은 데이터 블록을 압축된 형태로 저장함으로써 랭크 서브세팅으로 인해 증가한 접근 지연시간을 원 수준으로 다시 감소시킨다. 즉, 압축된 데이터 블록은 일반 데이터 블록에 비해 크기가 작기 때문에 일반 데이터 블록에 비해 짧은 시간 내에 읽기 또는 쓰기 동작을 수행할 수 있다. 그러나 Memzip 기술은 압축이 되지 않는 데이터 블록에 대해서는 메모리 접근시간을 줄일 수 없는 문제가 있다.

본 논문에서는 데이터 압축 기술을 사용하여 메모리 시스템의 에너지 효율 및 성능을 향상시키는 동적 랭크 서브세팅 기법(Dynamic Rank Subsetting, DRAS)을 제안한다. 메모리 시스템을 서브랭크로 구성하고 데이터 블록의 압축 여부에 따라 하나의 랭크(Full rank)로 동작하거나 두 개의 서브랭크로 동작되도록 한다. 만약 데이터 블록이 절반 크기로 압축될 경우, 압축된 데이터 블록은 두 개의 서브랭크 중 하나의 서브랭크에 저장된다. 만약 데이터 블록이 절반 크기로 압축되지 않는다면 기존 메모리 시스템에서처럼 데이터 블록을 모든 메모리 칩에 나누어 저장한다. 본 논문에서 제안하는 동적 랭크 서브세팅 기술은 데이터 블록이 압축될 때만 랭크 서브세팅을 적용하기 때문에 메모리 접근시간을 증가시키지 않는다. 또한, 압축된 데이터에 대해서는 서브랭크에 포함된 메모리 칩만 접근하기 때문에 메모리 시스템의 병렬성을 향상시킬 수 있다. 이와 더불어, 제안된 기술은 압축된 데이터 블록에 대한 메모리 요청에 대해서는 일부 메모리 칩만 접근하기 때문에 DRAM의 Read 및 Write 동작에 대한 전력 소모뿐만 아니라 행 활성화(Activation) 및 프리차지(Precharge) 동작에 대한 전력 소모도 감소시킬 수 있다.

본 논문에서 제안된 동적 랭크 서브세팅 기법은 멀티코어 프로세서와 메모리 시스템을 모델링한 시뮬레이터를 통해 성능 및 에너지 효율성 측면에서 평가되었다. 평가결과, 제안된 기법은 시스템 성능을 평균 12% 향상시키고 파워 소모를 평균 24% 감소시킬 수 있는 것으로 확인되었다.

II. Background

1. DRAM System

1.1 DRAM System Organization and Operations

고성능 컴퓨팅 시스템의 메인 메모리는 일반적으로 여러 DRAM 칩으로 구성된 DIMM 형태로 시스템에 설치된다. Fig. 1은 JEDEC 스타일의 기존 DRAM 시스템 구성을 보여준다. DIMM은 데이터 및 주소 버스, 칩 선택(CS), 클럭 활성화(CKE) 및 명령(RAS, CAS, WE) 신호를 통해 메모리 컨트롤러에 연결된다. CS 신호는 DRAM 칩이 DRAM 명령을 받아들이도록 하고 CKE 신호는 DRAM 칩의 전원 모드를 제어한다. RDIMM(Registered DIMM)에서는 주소, CS, CKE 및 명령 신호를 레지스터에 버퍼링한다. DIMM은 하나 이상의 랭크로 구성되며 각 랭크에는 여러 개의 DRAM 칩이 사용된다. 즉, 64 비트 데이터 버스를 형성하는 랭크당 8개의 x8 DRAM 칩을 사용하거나 또는 16개의 x4 DRAM 칩이 사용된다. DRAM 칩의 CS 신호는 메모리 컨트롤러의 단일 랭크 선택 신호에 의해 연결되므로 랭크의

모든 DRAM 칩은 하나의 메모리 디바이스처럼 동작된다.

DRAM 칩은 여러 뱅크(Bank)로 구성되어 있고 각 뱅크는 독립적으로 동작한다. 따라서 여러 메모리 요청이 서로 다른 뱅크에서 처리될 수 있다. DRAM의 각 Bank에 대한 주요 동작은 활성화(Activation), 프리차지(Precharge), 읽기(Read), 쓰기(Write), 리프레쉬(Refresh)로 구성된다. 활성화 동작은 특정 뱅크의 행(Row)를 활성화하여 해당 행에 저장된 데이터를 행버퍼(Row buffer)에 저장한다. 그 뒤 행버퍼에 저장되어 있는 데이터에 대해 읽기 또는 쓰기 동작을 수행한다. 만약, 메모리 요청에 대한 행이 행버퍼에 저장되어 있지 않다면 해당 뱅크를 프리차지한 뒤, 활성화 동작을 수행한다. 프리차지와 활성화 동작은 수행 시간이 매우 길고, 에너지 소모도 매우 크다[8, 9].

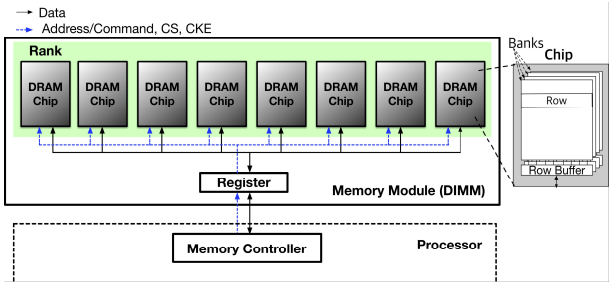


Fig. 1. DRAM System

1.2 Burst Data Reads/Writes

DDR SDRAM에서는 고속 및 고대역 데이터 전송을 위해 n-비트 프리페치(Prefetch) 아키텍처를 사용하여 버스트(Burst) 액세스를 지원한다. 초기 SDR SDRAM에는 데이터를 DRAM 코어에서 읽어 I/O 버퍼를 통해 외부 데이터 버스로 순차적으로 전송되었다. 이러한 방식은 데이터 전송 속도를 제한한다. 따라서 프리페치 아키텍처를 적용한 DDR SDRAM에서는 SDR SDRAM보다 더 많은 데이터를 I/O 버퍼로 읽어온 뒤 데이터를 외부 버스 클럭의 상승 및 하강 에지 모두에서 버스트로 전송한다. 이러한 방식으로 DRAM 코어의 클럭 주파수를 증가시키지 않고 데이터 전송 속도를 향상시킬 수 있다.

CPU는 각 메모리 채널로부터 64바이트(64B) 크기의 데이터(블록)를 하나의 쓰기 동작을 통해 읽어갈 수 있다. 이때, 메모리 채널이 8개의 DDR3 SDRAM x8 칩으로 구성될 경우, 각 DRAM 칩으로부터 8바이트(8B) 데이터를 4 클록에 걸쳐 버스트로 읽어가게 된다. Fig. 2는 여러 DRAM 칩으로 구성된 DRAM 시스템에서 기존 64B 캐시 라인이 저장되는 방식(데이터 매핑)을 보여준다. 그림에서 볼 수 있듯이 64B 캐시 라인의 각 바이트는 각 x8 DRAM 칩에 저장된다. 이때, 64B 캐시 라인에서 각 8B 워드(word)의 각 바이트는 8개 DRAM 칩에 나뉘어서 저장된다. 예를 들어, 첫 번째 바이트(B1)는 첫 번째 DRAM 칩에, 두 번째 바이트(B2)는 두 번째 DRAM 칩에, 세 번째 바이트(B3)는 세 번째 DRAM 칩에 저장된다.

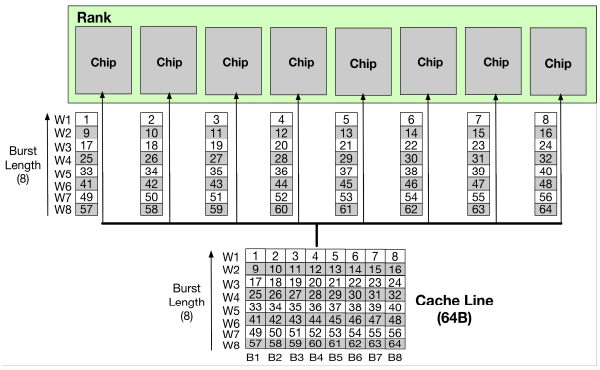


Fig. 2. Data Mapping in the Conventional DRAM System

2. Rank subsetting

일반적인 DRAM 시스템에서는 랭크 내의 모든 DRAM 칩이 물리적으로 분리되어 있어도 단일 메모리 디바이스처럼 동시에 작동하므로 상당한 에너지를 소모하고 디바이스 단위 병렬성을 제공하기 어렵다. 이러한 비효율성을 극복하기 위해 랭크 서브세팅(Rank subsetting)방법[3, 4, 5, 6]이 제안되었다. Fig. 3에서 볼 수 있듯이 랭크 서브세팅에서는 하나의 랭크를 구성하는 칩을 여러 서브랭크로 나누고 각 서브랭크가 독립적으로 동작되도록 한다. 이를 통해 메모리 시스템의 에너지 소모를 감소시키고 대역폭을 향상시킬 수 있다.

랭크 서브세팅 방법을 적용한 DRAM 시스템은 에너지 소모 및 메모리 대역폭 측면에서는 기존 DRAM 시스템에 비해 뛰어나지만, 메모리 접근 지연시간을 증가시키는 단점이 있다. 예를 들어, Fig. 3과 같이 하나의 랭크를 두 개의 서브랭크로 나눌 경우, 64B 블록을 메모리에서 읽어오기 위해 기존 DRAM 시스템에서는 4 클럭 사이클(Clock cycle)이 소요되었으나 랭크 서브세팅 방법을 적용한 DRAM 시스템에서는 그 두 배인 8 클록 사이클이 소요된다. 그 이유는 서브랭크의 I/O 핀 수가 기존 랭크에 비해 절반밖에 되지 않기 때문이다. 따라서 메모리 접근시간이 성능에 크게 영향을 미치는 워크로드에 대해서는 랭크 서브세팅 방법을 사용했을 때 시스템 성능이 더 나빠질 수 있다.

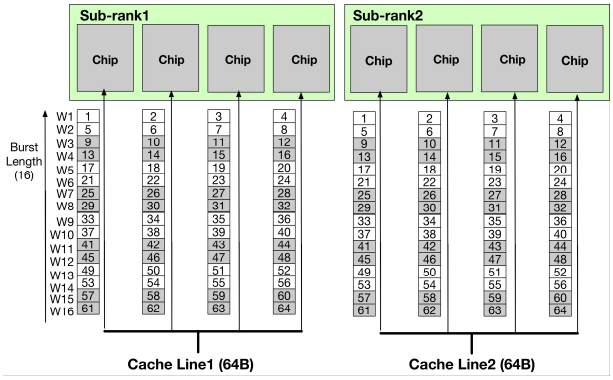


Fig. 3. Rank Subsetting

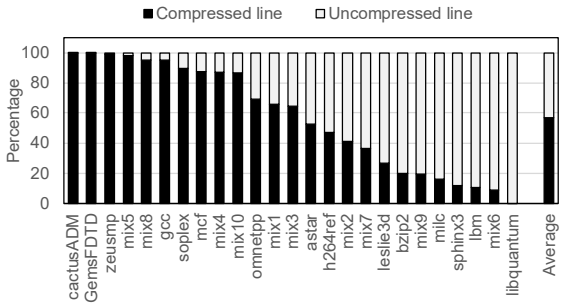


Fig. 4. The Percentage of Cache Lines Compressible to 32 Bytes

III. Proposed Scheme

1. Motivation: Data Compression

메모리에 저장되어 있는 데이터는 일반적으로 그 값이 매우 유사하다. 즉, 저장되어 있는 데이터의 엔트로피가 매우 낮기 때문에 단순한 압축 알고리즘을 사용하더라도 높은 데이터 압축률을 얻을 수 있다. 최근 캐시 메모리 및 메인 메모리에 적용 가능할 정도로 비용 효율적인 데이터 압축 알고리즘이 제안되었다[8, 9, 10]. Base-Delta-Immediate(BDI) 압축 알고리즘[8, 9]은 64B 데이터 블록을 구성하는 각 워드(4B)의 값은 매우 유사하다는 점을 이용하여 블록(64B) 단위로 압축을 수행한다. 즉, 64B 블록을 하나의 Base 워드(4B)와 Delta(Base 워드와 다른 워드의 차이, 1B~2B)로 표현하는 방식으로 압축한다. BDI는 간단한 하드웨어로 구현 가능하고 압축복원(Decompression)도 빠르게 수행할 수 있다. Frequent-Pattern Compression(FPC) 압축 알고리즘[10]은 64B 블록을 구성하는 데이터의 패턴을 파악하여 가장 빈번하게 사용되는 데이터 패턴에 대해 더 적은 bit을 할당하는 방식을 사용하는 데이터 압축방법이다. BDI와 FPC 압축 알고리즘은 캐시와 메인 메모리의 성능 및 에너지 소모를 감소시키기 위해 여러 연구에서 자주 사용되었다[12, 13, 14, 15, 16].

Fig. 4는 SPEC CPU2006 벤치마크를 실행할 때, 메인 메모리에 저장되는 64B 캐시 라인 중 BDI 압축 알고리즘을 통해 절반 크기(32B)로 압축 가능한 데이터 블록의 비율을 보여준다. 벤치마크에 따라 압축률에 차이는 있으나 평균적으로 약 50%의 데이터 블록이 절반 크기로 압축될 수 있음을 알 수 있다.

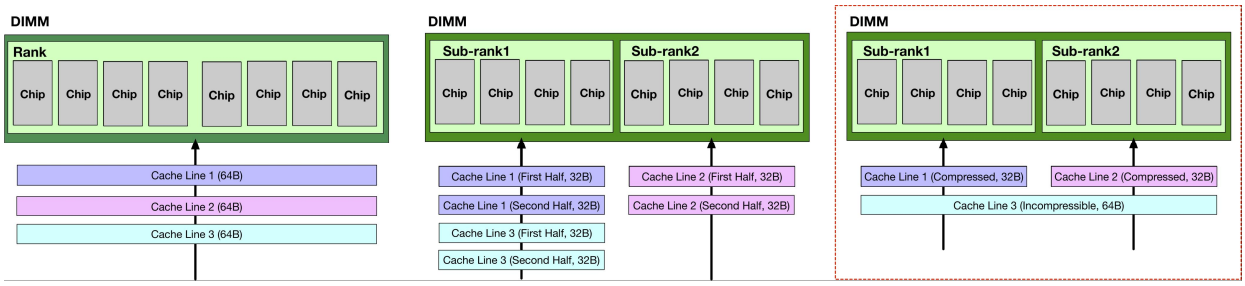
2. Dynamic Rank Subsetting with Data Compression (DRAS)

2.1 Overview

본 논문에서는 데이터 압축을 통해 메모리 시스템의 에너지 효율 및 성능을 향상시키는 동적 랭크 서브세팅 기법(Dynamic Rank Subsetting, DRAS)을 제안한다. 전술한 바와 같이 랭크 서브세팅 방법은 메모리 대역폭을 향상시킬 수 있으나 메모리 접근 지연시간을 증가시키는 단점이 있다. 본 논문에서 제안하는 DRAS 기법에서는 데이터 블록(64B)을 메모리에 저장할 때 절반 크기(32B)로 압축하여 하나의 서브랭크에 저장한다(Fig. 5c). 따라서, 압축된 캐시 라인에 대해서는 기존 랭크 서브세팅(Fig. 5b)에 비해 쓰기 동작의 지연시간을 기존 메모리 시스템 수준(Fig. 5a)으로 낮출 수 있다. 읽기 동작의 지연시간도 압축된 데이터 블록에 대해서는 일반 메모리 시스템에서와 동일한 접근지연시간으로 수행된다. 만약, 데이터 블록이 절반 크기로 압축되지 않는다면 일반 메모리 시스템에서와 같이 데이터 블록을 압축되지 않은 형태로 두 서브랭크에 저장한다(Fig. 5c). 따라서, 데이터 블록이 압축되지 않는 경우에 대해서도 메모리 읽기 및 쓰기 동작의 지연시간이 증가하지 않는다.

2.2 Partial Chip Access

DRAS 기법은 압축된 데이터 블록에 대해서는 일부 DRAM 칩(즉, 서브랭크)에만 읽기 및 쓰기 동작을 수행한다(Partial Chip Access). 그림 6은 DRAS 기법을 적용한 메모리 시스템의 쓰기 동작에 대한 예시를 보여준다. 데이터 블록(64B)은 메모리 컨트롤러(Controller)에서 절반 크기(32B)로 압축되고, 압축된 블록은 하나의 서브랭크에 저장된다(Fig. 8 예시에서는 Sub-rank0). 이때 서브랭크 0(Sub-rank0)에 속한 DRAM 칩(Chip0, Chip1, Chip2, Chip3)만 활성화되어 특정 메모리 행(Row[n])에 압축된 블록이 저장된다. 반면, 서브랭크1(Sub-rank1)에 속한



(a) Baseline System (b) Rank Subsetting (c) Dynamic Rank Subsetting

Fig. 5. Comparison of Baseline System, Rank Subsetting, and Dynamic Rank Subsetting (DRAS)

DRAM 칩(Chip4, Chip5, Chip6, Chip7)은 활성화되지 않는다. 따라서, 이들 DRAM 칩의 동일 메모리 행(Row[n])에는 데이터가 저장되지 않는다.

DRAS 기법은 압축된 데이터 블록에 대해서는 절반의 DRAM 칩만 사용하기 때문에 사용되지 않는 DRAM 칩에 대해서는 활성화, 프리차지, 읽기, 쓰기 동작을 수행하지 않는다. 따라서, 메모리 시스템의 에너지 소모를 크게 감소시킬 수 있다. 이때, 기존 랭크 서브세팅 기법과는 달리 메모리 읽기 및 쓰기 동작의 지연시간이 증가하지 않기 때문에 시스템 성능에는 큰 영향을 주지 않는다.

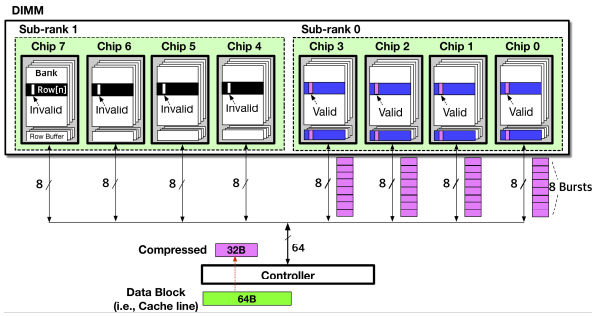


Fig. 6. Partial Chip Access

2.3 Sub-Rank Interleaving

DRAS 기법은 두 서브랭크를 독립적으로 사용할 수 있다. 본 논문에서는 DRAS 기법이 적용된 메모리 시스템에서 메모리 뱅크 레벨 병렬성(Bank-level Parallelism)을 향상시킬 수 있는 데이터 배치 방식을 제안한다. DRAS 기법에서는 압축된 데이터를 두 개의 서브랭크 중 하나에만 저장한다. 이때 메모리 주소에 따라 압축된 데이터가 저장될 서브랭크가 결정된다. 이러한 데이터 배치 방식은 압축된 데이터들을 독립적으로 동작하는 두 개의 서브랭크에 분산 저장함으로써(Sub-rank Interleaving) 메모리 병렬성을 향상시킨다. Fig. 7 예시에서 볼 수 있듯이 Row[n]에 사상된 데이터 블록은 서브랭크0(Sub-rank0)에 저장되고 Row[n+1]에 사상된 데이터 블록은 서브랭크1(Sub-rank1)에 저장됨으로써 이들 블록에 대한 메모리 요청을 동시에 처리할 수 있다.

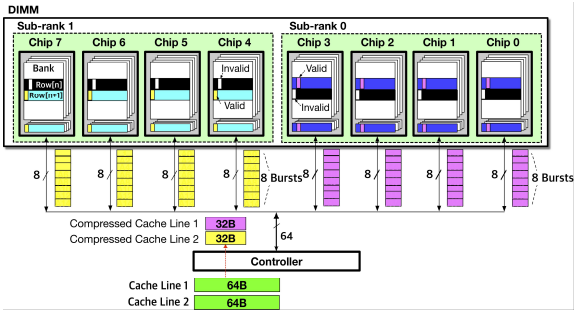


Fig. 7. Sub-Rank Interleaving

3. Implementation

3.1 Memory System Architecture

메모리 모듈(DIMM)은 버퍼 칩(Buffer Chip)과 다수의 DRAM 칩으로 구성된다(Fig. 8). 메모리 컨트롤러에는 DRAS 기법을 구현하기 위한 디바이스 제어 유닛(Device Control Unit, DCU)이 포함된다.

DCU는 데이터 압축엔진을 사용하여 데이터 블록을 메모리에 저장할 때 해당 블록을 절반 크기로 압축한 뒤 하나의 서브랭크에 저장한다. 읽기 메모리 요청에 대해서는 요청된 데이터 블록이 압축된 상태로 저장되어 있다면 해당 서브랭크에서 데이터 블록을 읽어 원본 데이터로 복구한 뒤 CPU로 전달한다.

DCU는 메모리 읽기 및 쓰기 요청을 메모리 모듈로 보낼 때, 캐시 블록의 압축 여부에 따라 칩 선택 정보 (Chip selection information)을 생성하여 함께 보낸다. 이때 메모리 요청에 대한 메모리 주소와 데이터 블록의 압축 여부를 바탕으로 서브랭크를 선택한다. 이를 통해, 데이터 블록이 압축되어 있다면 두 서브랭크 중 하나의 서브랭크에 포함된 DRAM 칩이 선택되도록 하고, 압축되어 있지 않다면 모든 DRAM 칩이 선택되도록 한다.

서브랭크를 개별적으로 제어하기 위해 메모리 모듈의 버퍼 칩은 DCU가 전달한 칩 선택 정보를 바탕으로 각 서

브랭크에 대한 칩 선택(Chip-select, CS) 신호를 생성한다. 버퍼 칩에서 메모리 명령을 각 DRAM 칩에 보낼 때, CS 신호가 활성화된 DRAM 칩은 해당 명령어에 따라 동작하게 되고 CS 신호가 비활성화된 칩에서는 모든 메모리 명령이 무시된다.

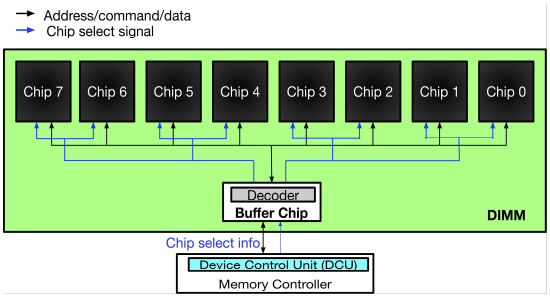


Fig. 8. Memory System Architecture

3.2 Device Control Unit

DCU는 압축기(Compressor), 압축 복원기(Decompressor), 요청 큐(Request Queue), 스케줄러(Scheduler), 메타 데이터 캐시(Metadata Cache), 제어 로직(Control Logic)으로 구성된다 (Fig. 9).

3.2.1 Compressor and Decompressor

압축기 및 압축 복원기는 각 데이터 블록의 압축률을 최대화하기 위해 여러 압축 알고리즘을 제공하도록 구성될 수 있다. 압축기 엔진이 여러 압축 알고리즘을 제공하는 경우, 데이터 블록은 각 압축 알고리즘을 통해 다른 데이터 크기로 병렬 압축되고 압축된 블록 중 가장 작은 크기로 압축된 블록을 요청 큐에 저장된다.

3.2.2 Metadata Cache

메타 데이터 캐시(Metadata Cache)에는 가장 최근 접근되었던 데이터 블록의 메타 데이터(데이터 블록의 압축 여부)가 저장된다. 메타 데이터는 하나의 캐시 블록 (64B)

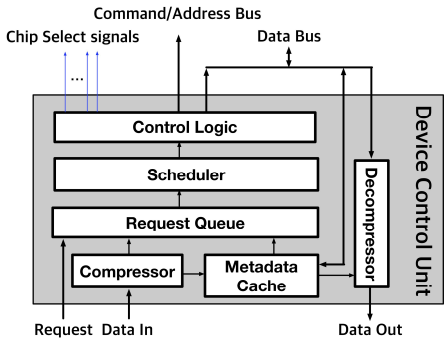


Fig. 9. Device Control Unit (DCU)

에 대해 1bit가 필요하지만, 메모리 시스템의 전체 용량이 256GB일 경우 메타 데이터의 크기는 512B가 된다. 멀티코어 프로세서의 L3 캐시 크기가 16~32MB인 점을 고려할 때 메타 데이터를 프로세서 내부에 저장하는 것은 불가능하다. 따라서 메타 데이터도 메모리에 저장되어야 하는데, 이로 인해 모든 메모리 읽기 및 쓰기 요청에 대해 메타 데이터를 읽기 위한 추가적인 메모리 접근이 필요하게 되어 메모리 시스템 성능을 크게 떨어뜨릴 수 있다. 따라서, 최근 접근되었던 데이터 블록에 대한 메타 데이터만을 작은 크기의 메타 데이터 캐시에 저장하게 된다. 일반적으로 메모리 접근 패턴에는 지역성(Locality)가 있기 때문에 메타 데이터 캐시를 사용할 경우, 메타 데이터 접근으로 인한 추가적인 메모리 접근 수를 크게 감소시킬 수 있다.

3.2.3 Request Queue

요청 큐는 별도의 읽기 및 쓰기 큐 또는 통합 큐로 구현될 수 있다. 쓰기 요청의 경우, 압축기는 해당 데이터 블록을 압축하고 압축된 데이터를 요청 큐의 각 항목 (Entry)에 저장한다. 이때 메모리 요청에 대한 데이터의 압축 여부도 요청 큐에 함께 저장된다. 읽기 요청의 경우에는 데이터의 압축 여부를 메타 데이터 캐시에서 읽어와 요청큐의 각 항목에 저장한다.

3.2.4 Scheduler

스케줄러는 메모리 요청의 대기시간, 각 서브랭크의 뱅크 별 활성화된 행(Open row), 그리고 메모리 요청에 대한 데이터의 압축 여부를 참고하여 요청 큐에서 메모리 요청을 선택한다. FCFS(first-come-first-serve) 정책이 사용되는 경우 오랫동안 요청 큐에서 대기하고 있는 메모리 요청에 더 높은 우선순위가 부여된다. FR-FCFS (first-ready first-come-first-serve) 정책이 사용되는 경우, 현재 활성화된 행(Open row)에 대한 메모리 요청에 더 높은 우선순위를 준다. 이때, 메모리 요청에 대한 데이터가 압축되어 있지 않을 경우, 두 서브랭크에서 데이터가 저장된 행이 활성화되어 있는지 확인한다.

IV. Experimental Results

1. Environment

DRAS의 성능 이점을 평가하기 위해 USIMM [11]을 기반으로 메모리 시스템 시뮬레이터를 개발하였다. USIMM

Table 1. Simulated System Configuration

Number of Cores Processor Clock	4 3.2Ghz
L1 Cache L2 Cache LLC (Last-Level Cache)	32KB, 8-Way, 64B line, 4 Cycles 256KB, 8-Way, 64B line, 12 Cycles 8MB, 16-way, 64B line, 30 Cycles
Memory Bus Frequency Memory Channels Ranks per Channel Bank Groups Banks per Bank Group Rows per Bank	1600MHz 2 1 4 4 64KB
DRAM Access Timing: tRCD-tRP-tCAS tRFC/tREFI	 22-22-22 350ns/7.8us

을 확장하여 프로세서 코어 및 캐시 메모리를 모델링하였고, 프로세서는 비순차적 실행 (Out of Order Execution)을 모델링하였다. 본 연구에서 시뮬레이션한 컴퓨터 시스템 구성은 Table 1에 요약되어 있다.

성능 평가를 위해 SPEC CPU2006 벤치마크에서 MPKI(Miss Per Kilo Instruction) 값이 큰 메모리 집약적 벤치마크 15개를 사용하였다. 또한, 서로 다른 특성을 가진 어플리케이션이 동시에 실행되는 시나리오를 평가하기 위해 서로 다른 SPEC CPU2006 벤치마크로 구성된 MIX 워크로드 10개를 사용하였다 (Table 2).

Table 2. MIX Workloads

mix1	bzip2, libquantum, astar, cactusADM
mix2	lbm, omnetpp, mcf, GemsFDTD
mix3	h264ref, milc, bwaves, gcc
mix4	sphinx3, astar, soplex, zeusmp
mix5	bzip2, mcf, dealII, gcc
mix6	omnetpp, bwaves, GemsFDTD, cactusADM
mix7	libquantum, milc, sphinx3, lbm
mix8	leslie3d ,xalancbmk, h264ref, astar
mix9	soplex, dealII, GemsFDTD, cactusADM
mix10	lbm, sphinx3, leslie3d, h264ref

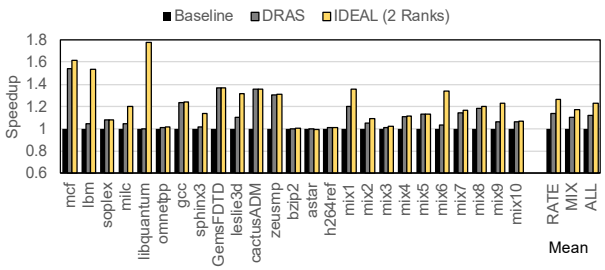


Fig. 10. Performance

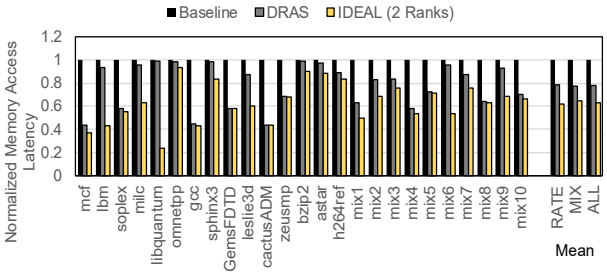


Fig. 11. Memory Access Latency Reduction with DRAS

2. Performance

Fig. 10은 DRAS 기법을 적용한 메모리 시스템의 성능을 보여준다. DRAS 기법은 시스템 성능을 평균 12% 향상시킬 수 있는 것으로 평가되었다. 특히, mcf, gcc, GemsFDTD와 같은 메모리 집약적인 벤치마크에 매우 높은 성능향상을 보였다. DRAS 기법은 시스템 성능을 mcf에 대해서는 48%, gcc에 대해서는 38% 향상시켰다. 다만, lbm, libquantum과 같이 메모리 집약적이지만 데이터 압축률이 낮은 벤치마크에 대해서는 DRAS를 통한 시스템 성능향상이 크지 않았다. 그 이유는 libquantum과 lbm 벤치마크에서는 절반 크기로 압축되는 메모리 블록이 메모리에 저장되는 전체 블록 중 10% 미만이기 때문이다(Fig.4). 이들 벤치마크에 대해서도 보다 복잡한 압축 알고리즘을 사용하여 데이터 압축률을 높인다면 DRAS 기법을 통해 시스템 성능을 더 높일 수 있다. 특히, libquantum 벤치마크는 모든 메모리 블록이 절반 크기로 압축될 경우 시스템 성능을 최대 80% 향상시킬 수 있을 것으로 평가되었다.

DRAS 기법은 메모리 시스템의 뱅크 단위 병렬성을 높여 메모리 요청 처리율을 향상시킨다. 이를 통해 평균 메모리 접근 지연시간이 감소되고 이는 전체 컴퓨터 시스템의 성능 향상으로 이어진다. Fig. 11은 DRAS 기법을 통한 메모리 접근 지연시간 감소 수준을 보여준다. DRAS 기법을 적용할 경우 메모리 접근 지연시간은 평균 21% 감소하는 것으로 평가되었다. 특히, 메모리 집약적인 벤치마크에 대해 메모리 접근 지연시간 감소 효과가 컸다. DRAS 기법은 메모리 접근 지연시간을 mcf와 gcc에 대해서는 58%, GemsFDTD에 대해서는 40% 감소시켰다.

3. Power Consumption

DRAS 기법을 적용한 메모리 시스템에서는 압축된 데이터 블록에 대한 메모리 요청에 대해서는 전체 DRAM 칩 중 절반만이 활성화된다. 하나의 메모리 요청에 대해 절반의 DRAM 칩만을 사용할 경우 메모리 시스템의 동적 파워 소모(Dynamic power consumption)을 최대 절반 수준

으로 감소시킬 수 있다. Fig. 12는 DRAS 기법을 적용했을 때의 메모리 시스템의 파워 소모 감소를 보여준다. 데이터 압축률이 높은 메모리 집약 벤치마크의 경우 메모리 시스템 파워소모가 높다는 것을 알 수 있다. DRAS 기법은 메모리 시스템 파워소모를 최대 30% 감소시켰고 평균 24% 감소시킬 수 있는 것으로 평가되었다. 특히, 메모리 집약적이면서 데이터 압축률이 높은 벤치마크인 mcf, gcc, GemsFDTD에서 파워소모 감소율이 높았다.

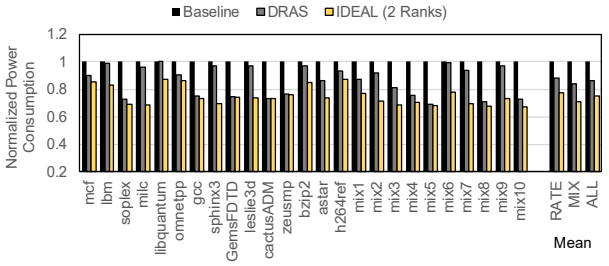


Fig. 12. Power Consumption of Memory System

Fig. 13은 DRAS 기법을 사용했을 때의 각 벤치마크에 대한 EDP(Energy-delay product)를 보여준다. EDP는 에너지 소모와 성능향상을 함께 고려한 지표로써 시스템의 에너지 효율성을 평가할 때 자주 사용된다. EDP 값이 작을수록 우수하다고 할 수 있다. DRAS 기법은 거의 대부분의 벤치마크에 대해 EDP를 감소시켰으며 평균 20% 감소시킬 수 있는 것으로 평가되었다. 메모리 집약적인 벤치마크인 mcf에 대해서는 DRAS 기법을 통해 EDP를 약 60% 감소시켰다. libquantum 벤치마크에 대해서는 모든 데이터가 절반크기로 압축될 경우 (IDEAL), EDP가 70% 감소되어 DRAS 기법에 의해 가장 큰 효과를 볼 수 있을 것으로 예상되었다.

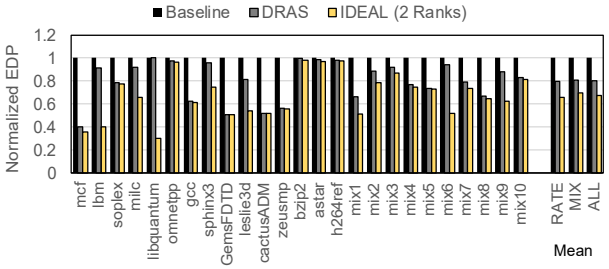


Fig. 13. Energy-Delay Product (EDP)

V. Conclusions

빅데이터 처리 및 인공지능 기술의 발전으로 인해 컴퓨터 시스템에서 처리하는 데이터의 양이 급속히 증가하고 있고 이로 인해 메모리 시스템의 성능 및 에너지 효율성이 컴퓨터 시스템 설계에 있어 매우 중요해지고 있다. 본 연구에서는 데이터 압축 기술을 사용하여 메모리 시스템의 성능 및 에너지 효율성을 향상시킬 수 있는 동적 메모리 랭크 구성 기법 (Dynamic Rank Subsetting, DRAS)을 제안하였다. DRAS 기술은 데이터를 압축한 뒤, 압축된 데이터를 일부 DRAM 칩에만 저장하도록 하여 메모리 시스템의 전력 소모를 감소시키는 동시에, 메모리 대역폭을 향상시킬 수 있는 기술이다. DRAS 기술을 기존 메모리 시스템에 적용했을 때, 시스템 성능을 평균 12% 향상시킬 수 있고, 메모리 시스템 파워 소모를 평균 24% 감소시킬 수 있을 것으로 평가되었다. 향후 DRAS 기술을 활용하여 비휘발성 메모리(NVDIMM)의 Endurance를 향상시키는 연구를 수행할 계획이다. DRAS 기술은 압축된 데이터에 대해서는 일부 메모리 칩만 사용하기 때문에 쓰기 횟수를 감소시킬 수 있고 이를 통해 비휘발성 메모리의 내구성 (Endurance)를 향상시킬 수 있을 것으로 기대된다.

ACKNOWLEDGEMENT

This study was partially supported by the BK21 Plus project (SW Human Resource Development Program for Supporting Smart Life) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (No. 21A20131600005) and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1G1A1011403).

REFERENCES

[1] Justin Meza, Mehul A. Shah, Parthasarathy Ranganathan, Mike Fitzner, and Judson Veazey, “Tracking the power in an enterprise decision support system,” Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design, pp. 261-266, New York, USA, 2009. DOI: 10.1145/1594233.1594295

[2] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna and C. Le, "RAPL: Memory power estimation and capping," Proceedings of the 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design, pp. 189-194, Austin, USA, 2010. DOI:

10.1145/1840845.1840883

[3] Hongzhong Zheng, Jiang Lin, Zhao Zhang, Eugene Gorbatov, Howard David, and Zhichun Zhu, “Mini-rank: Adaptive DRAM architecture for improving memory power efficiency,” Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture, pp. 210–221, USA, 2008. DOI: 10.1109/MICRO.2008.4771792

[4] Aniruddha N. Udipi, Naveen Muralimanohar, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Norman P. Jouppi, “Rethinking DRAM design and organization for energy-constrained multi-cores,” Proceedings of the 37th annual international symposium on Computer architecture , pp. 175-186, New York, USA, 2010. DOI: 10.1145/1815961.1815983

[5] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich and R. S. Schreiber, "Future scaling of processor-memory interfaces," Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1-12, Portland, OR, 2009. DOI: 10.1145/1654059.1654102

[6] J. H. Ahn, J. Leverich, R. Schreiber and N. P. Jouppi, "Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs," IEEE Computer Architecture Letters, vol. 8, no. 1, pp. 5-8, Jan. 2009. DOI: 10.1109/L-CA.2008.13

[7] A. Shafiee, M. Taassori, R. Balasubramonian and A. Davis, "MemZip: Exploring unconventional benefits from memory compression," 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 638-649, Orlando, FL, 2014. DOI: 10.1109/HPCA.2014.6835972

[8] Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry, “Linearly compressed pages: a low-complexity, low-latency main memory compression framework,” Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 172–184, New York, USA, 2013. DOI:10.1145/2540708.2540724

[9] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 377-388, Minneapolis, MN, 2012. DOI: 10.1145/2370816.2370870

[10] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep, vol. 1500, 2004.

[11] N. Chatterjee et al., USIMM: The Utah Simulated Memory Module, tech. report UUCS-12-002, Univ. of Utah, 2012.

[12] Vinson Young, Prashant J. Nair, and Moinuddin K. Qureshi, “DICE: Compressing DRAM Caches for Bandwidth and Capacity,” Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17), pp. 627-638, New York, USA. DOI: 10.1145/3079856.3080243

[13] V. Young, S. Kariyappa, and M. Qureshi, “Enabling Transparent Memory-Compression for Commodity Memory Systems,” Proceedings of 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 570-581, Washington, USA, 2019. DOI: 10.1109/HPCA.2019.00010

[14] Seokin Hong, Bulent Abali, Alper Buyuktosunoglu, Michael B. Healy, and Prashant J. Nair, “Touché: Towards Ideal and Efficient Cache Compression By Mitigating Tag Area Overheads,” Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO ’52). Association for Computing Machinery, pp. 453–465, New York, USA, 2019. DOI:10.1145/3352460.3358281

[15] Seokin Hong, Prashant J. Nair, Bulent Abali, Alper Buyuktosunoglu, Kyu-Hyoun Kim, and Michael B. Healy, “Attaché: towards ideal memory compression by mitigating metadata bandwidth overheads,” Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-51), pp. 326–338, 2018. DOI:10.1109/MICRO.2018.00034

[16] Somayeh Sardashti and David A. Wood. 2017. Could Compression Be of General Use? Evaluating Memory Compression Across Domains. ACM Trans. Archit. Code Optim. 14, 4, Article 44, 24 pages, 2017. DOI: 10.1145/3138805

Authors

Seokin Hong received the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 2015. From 2015 to 2017, he had been a senior engineer at Samsung Electronics. In 2017, he moved to IBM T.J. Watson Research Center where he worked on secure processor architectures and emerging memory/storage systems. He is currently an assistant professor at Kyungpook National University. His current research interests include the design of low power, reliable, and high-performance processor architectures and memory systems. He received Best Paper Awards from International Conference on Computer Design (ICCD) in 2010 and Design Automation and Test in Europe (DATE) in 2013.