Journal of The Korea Society of Computer and Information
Vol. 24 No. 8, pp. 1-8, August 2019

www.ksci.re.kr
https://doi.org/10.9708/jksci.2019.24.08.001

Bandwidth-aware Memory Placement on Hybrid Memories targeting High Performance Computing Systems

Jongmin Lee*

Abstract

Modern computers provide tremendous computing capability and a large memory system. Hybrid memories consist of next generation memory devices and are adopted in high performance systems. However, the increased complexity of the microprocessor makes it difficult to operate the system effectively. In this paper, we propose a simple data migration method called Bandwidth-aware Data Migration (BDM) to efficiently use memory systems for high performance processors with hybrid memory. BDM monitors the status of applications running on the system using hardware performance monitoring tools and migrates the appropriate pages of selected applications to High Bandwidth Memory (HBM). BDM selects applications whose bandwidth usages are high and also evenly distributed among the threads. Experimental results show that BDM improves execution time by an average of 20% over baseline execution.

▸Keyword: Performance, Data Migration, Bandwidth-aware

I. Introduction

CMOS 공정 기술의 발전에 따라 동일한 칩 영역에 집적할 수 있는 코어 수가 크게 증가하고 있다. 이러한 결과로, 단일 칩 공간에 수십에서 수백 개의 코어를 내장한 매니-코어(Many-core) 프로세서가 등장 하게 되었다 [1-3]. 매니-코어 프로세서 환경에서 사용자는 다수의 응용프로그램을 동시에 실행할 수 있으며 이는 해당 응용프로그램을 메인 메모리에 적재할 수 있는 대용량의 DRAM(Dynamic RAM)을 필요로 한다. DRAM은 1T1C로 메모리 셀을 구현할 수 있는 고집적성과 빠른 접근 시간의 특성으로 지난 수십 년간 메인 메모리로 주요하게 사용되어 왔다. 하지만, DRAM의 대용량 메모리로의 확장성의 제한과 고성능 컴퓨팅 시스템의 높은 대역폭 요구에 대응하기 위해 집적도가 높고 비휘발성을 갖는 차세대 메모리 기술들이 등장하고 있다. 그러나 차세대 메모리들을 메인 메모리로 대체하기에는 아직 해결해야할 기술적인 문제들이 남아있다. 이를 극복하기 위한 방편으로, 현재 메모리 시스템에 차세대 메모리 기술들의 장점들을 조합하여 구성한 하이브리드(Hybrid) 메모리 시스템에 대한 연구가 활발하게 이루어지고 있다 [4-6].

KNL(Knights Landing) 프로세서는 Intel사의 2세대 Xeon Phi 제품군의 코드 네임으로 수십 개의 코어와 MCDRAM(Multi-channel DRAM)이라 명명된 HBM(High bandwidth-memory)를 탑재하고 있다 [1]. DRAM과 MCDRAM은 액세스 시간, 대역폭, 메모리 용량 측면에서 서로 다른 특성을 가지고 있기 때문에 응용프로그램 실행 시 메모리 관리 방법에 따라 시스템 성능이 크게 달라질 수 있다. 고성능 컴퓨팅 시스템에서 멀티/매니 프로세서와 하이브리드 메모리의 등장은 다양한 구조의 마이크로프로세서가 등장할 수 있다는 것을 의미한다. 고성능 컴퓨팅 환경이 다양해지면서 시스템 자원을 효율적으로 활용하기 위한 복잡성이 증가하게 되었다.

본 논문은 응용프로그램 실행 시 하드웨어 모니터링 툴을 이용하여 추출한 데이터를 기반으로 하여 하이브리드 메모리를 효율적으로 사용할 수 있는 데이터 이주(Migration) 전략을 제

• First Author: Jongmin Lee, Corresponding Author: Jongmin Lee
*Jongmin Lee (square55@wku.ac.kr), Dept. of Computer Engineering, WonKwang University
• Received: 2019. 08. 06, Revised: 2019. 08. 26, Accepted: 2019. 08. 26.
• This paper was sponsored by Soongsan Fellowship in Wonkwang University in 2018.

안한다. 본 논문에서는 DRAM과 HBM으로 구성된 하이브리드 메모리가 구현되어 있는 Intel사의 KNL 프로세서를 대상으로 하였다. BDM(Bandwidth-aware Data Migration)이라 명명된 제안된 기법은 주기적으로 응용프로그램의 실행 상태를 모니터링하고 수집된 메모리 대역폭을 기반으로 메모리 대역폭 요구가 크다고 판단되는 응용프로그램을 선택한다. 그 후, 선택된 응용프로그램의 페이지(page)를 HBM 메모리로 이주하여 하이브리드 메모리를 효과적으로 사용한다. Intel KNL 프로세서를 사용한 실험 환경에서 제안된 기법을 평가한 결과, 기본 실행 시간 대비 제안된 기법이 평균적으로 20% 성능 향상을 보이는 것을 검증하였다.

본 논문의 구성은 다음과 같다. 먼저 다음 장에서는 차세대 메모리 디바이스를 활용한 하이브리드 메모리와 관련된 기존 연구들을 다룬다. 3장에서는 본 논문에서 제안하는 기법의 설계와 구현에 대한 내용을 제시한다. 4장에서는 KNL 프로세서 환경에서 제안된 기법에 대한 실험 방법과 결과에 대한 내용을 기술한다. 마지막으로 5장에서는 본 논문의 결론과 향후 계획을 기술한다.

II. Preliminaries

1. Related works

1.1 Hybrid Memories

컴퓨팅 시스템에서 주요하게 사용되고 있는 DRAM을 대체하기 위한 차세대 메모리에 관한 연구가 활발히 진행 중이다 [7, 8]. PCM(Phase-change Memory)과 RRAM(Resistive RAM)은 DRAM보다 쉽게 집적할 수 있는 특성을 가지고 있기 때문에 보다 큰 용량의 메모리 설계가 가능하지만, 셀 당 쓰기 가능한 횟수가 제한되어 있어 대체로 메모리 수명이 짧은 것으로 평가된다. STT-RAM(Spin-Transfer Torque RAM)은 빠른 쓰기 속도와 우수한 쓰기 내구성을 가지고 있지만, DRAM보다 집적하기가 상대적으로 어렵기 때문에 경제성 측면에서 현재까지는 DRAM을 대체할 필요성이 낮다고 볼 수 있다. 현존하는 새로운 메모리 기술들을 고려하였을 때, 범용(universal) 메모리로의 사용은 현재까지는 해결해야할 문제가 많은 것으로 판단된다. 이러한 차세대 메모리를 효과적으로 이용하기 위해 DRAM과 다른 종류의 메모리를 함께 구성하여 운용하는 하이브리드 메모리에 대한 연구가 수행되었다 [9-15]. 하이브리드 메모리는 기존의 DRAM과 차세대 메모리를 구성하는 방법에 따라서 다양한 구조로 구현이 가능하다. 예를 들어, DRAM을 캐시로 사용하고 고용량의 PCM을 메인 메모리로 사용하는 구조가 소개 되었다 [9, 10]. 위의 방식은 DRAM을 캐시로 사용하여 메인 메모리로의 쓰기 동작을 필터링하여 PCM의 내구성과 쓰기 지연을 향상시킨다. 또한, DRAM과 PCM은 같은 레벨에 위치시킨 기법이 제안되었다 [11]. 쓰기 동작이 지연되는 것을 방지하고 PCM의 수명이 늘리기 위해 페이지 관리자는 PCM과 DRAM중 페이지를 선별적으로 할당한다. 위의 기법에

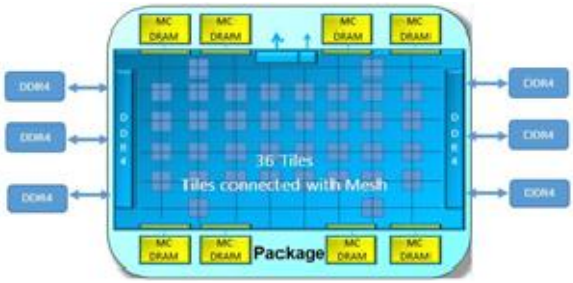


Fig. 1. Structure of Intel KNL Processor [1]

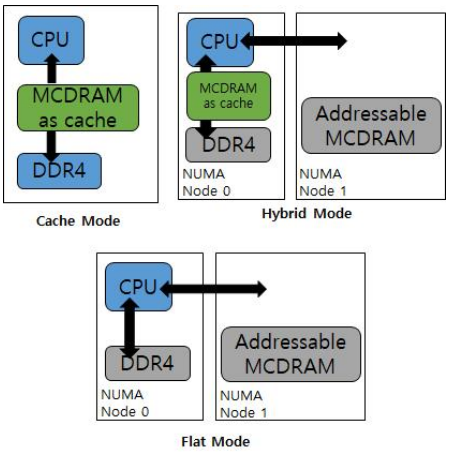


Fig. 2. Memory Configurations of KNL Processor

서는 메모리 접근 속도와 수명을 연장하기 위한 PCM의 쓰기 동작을 줄이기 위한 기법들을 제안한 반면 본 논문에서는 메모리 대역폭의 차이를 가지는 DRAM과 HBM으로 구성된 하이브리드 메모리의 효율적인 사용에 대해 다룬다는 점에서 차이가 있다.

1.2 Data Migration

데이터(혹은 페이지) 이주 기법은 메모리에 저장되는 응용프로그램 데이터의 위치를 선택적으로 결정하는 방법으로 응용프로그램 실행 중 동적으로 이주를 수행하거나 참조 패턴 프로파일링을 통해 정적으로 수행이 가능하며 NUMA(Non-Uniform Memory Access) 구조를 채용한 시스템과 하이브리드 메모리 시스템을 대상으로 활발한 연구가 이루어지고 있다. NUMA 시스템에서 메모리를 효율적으로 사용하기 위한 기본적인 방법은 응용프로그램의 데이터 참조 패턴을 분석하여 데이터의 위치를 자주 참조하는 프로세서와 가까운 곳으로 이동하여 메모리 접근 시간을 줄이는 것이다. 본 절에서는 제안된 연구와 관련 있는 하이브리드 메모리 시스템을 대상으로 한 기존의 데이터 이주 기법에 대해 다루도록 하겠다[12-14]. 관련 연구 [12]에서는 DRAM과 비휘발성 플래시 메모리로 구성된 하이브리드 메모리 시스템을 대상으로 페이지 이주 기법을 제안하였다. 응용

프로그램의 데이터를 참조횟수에 따라 분류하여 (HOT&COLD) 자주 참조되는 데이터를 상대적으로 접근 속도가 빠른 DRAM에 저장하고 드물게 참조되는 페이지를 플래시 메모리에 저장하는 방법으로 전력소비 감소와 성능향상을 개선하였다. DRAM과 PCM을 대상으로 한 관련 연구 [13]은 PCM의 약점인 높은 쓰기 대기 시간과 낮은 쓰기 내구성을 보완하기 위해 응용프로그램의 데이터를 분석하여 쓰기 횟수가 많은 페이지를 DRAM에 저장하는 방법으로 페이지 이주를 수행한다. 관련 연구 [14]에서는 최근 주목받고 있는 머신러닝 기법을 적용한 하이브리드 메모리 시스템의 페이지 이주 기법이 제안되었다. 위의 연구에서는 응용프로그램이 사용하는 페이지에 우선순위를 부여하여 학습을 수행하는데 시스템 리소스 부담을 감소시키는 방향으로 심층 신경망을 학습시킨다. 이 후 학습된 신경망을 이용하여 데이터 이주를 수행한다.

본 논문에서는 응용프로그램의 대역폭 정보를 활용한 동적 메모리 관리 기법을 제안한다. 본 연구의 핵심은 하드웨어나 소프트웨어 수정 없이 HPC시스템에서 서로 다른 유형의 메모리를 효율적으로 사용하는 알고리즘이다. 본 논문에서는 Intel KNL 프로세서를 주요 대상으로 했지만, 서로 다른 대역폭을 갖는 하이브리드 메모리를 채용한 메모리 시스템이라면 본 논문에서 제안된 기법이 적용가능 하다.

III. The Proposed Scheme

1. Background

본 절에서는, Intel KNL 프로세서에서 메모리와 관련된 주요 특징에 대해 다룬다. KNL 프로세서는 하이브리드 메모리 시스템이 상용 프로세서에 적용된 사례로 TACC(Texas Advanced Computing Center)의 대표 시스템인 Stampede2 및 Berkeley Lab의 CORI 슈퍼컴퓨터와 같은 고성능 컴퓨팅 시스템에서 널리 사용되고 있다. Fig. 1은 KNL 프로세서와 하이브리드 메모리의 구조를 나타낸다. KNL 프로세서는 최대 72개의 코어와 DRAM과 HBM으로 구성된 하이브리드 메모리를 채용하고 있다. HBM은 KNL 프로세서에서 MCDRAM(Multi-Channel DRAM)이라는 이름으로 명명되었다. 각각의 메모리는 다음과 같은 특성을 갖는다. 먼저, MCDRAM은 8개의 채널, 16GB의 메모리 용량, 최대 400GB/second 대역폭을 지원한다. DRAM은 DDR4가 사용되었으며 6개의 채널, 384GB의 메모리 용량, 최대 100GB/second 대역폭을 지원한다. MCDRAM은 on-chip, DRAM은 off-chip에 위치한다. Fig. 1은 KNL 프로세서의 36개 타일(tile)을 나타내고 있으며, 각 타일은 1MB L2 캐시를 공유하는 2개의 코어로 구성되어 있으며 2D-mesh network on-chip을 통해 연결되어 있다. 타일-구조의 매니 프로세서에서 프로세스와 해당 프로세스가 사용하는 페이지의 할당 위치에 따라 프로세서의 성능이 달라질 수 있는데 KNL 프로세서는 타일들을 논리적으로 그룹화 할 수 있는 All-to-All, Quadrant, SNC의 세 가지 방법을 제공한다. 본 논문에서는 L2 캐시에서

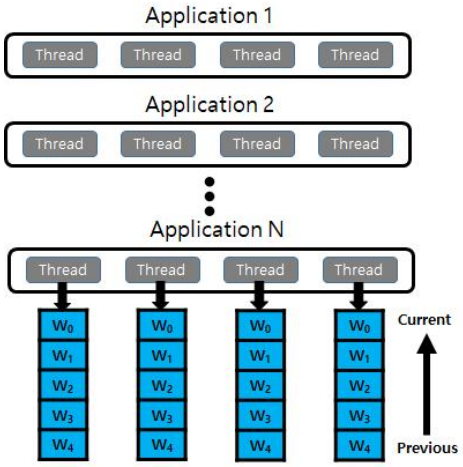


Fig. 3. Structure of Monitored Data

Algorithm 1 $BDMAlgorithm\ (p)$

Input: $p \leftarrow$ time window period
/*
L : the list of running application
W : the list of windows for monitoring data
b : the selected application id
*/
initiate L
while $length(L) > 0$ **do**
wait p
1. $L \leftarrow getCurrentApplication()$
2. $Monitoring(L, W)$
3. $b \leftarrow Selection(L, W)$
4. $Migration(b, L, W)$
end while

부재가 일어난 경우 지연 시간을 최대한 줄이기 위해 Intel에서 권장하는 Quadrant cluster 설정을 사용하였다 [16]. MCDRAM은 부팅 시 캐시(cache), 플랫(flat), 하이브리드(hybrid)의 세 가지 모드 중 하나를 선택하여 구성할 수 있다. Fig. 2는 KNL 프로세서가 지원하는 하이브리드 메모리 구조를 나타내고 있다. 플랫 모드는 MCDRAM을 DRAM과 같은 주소 공간(address space)에서 주소 지정 가능한 메모리로 사용하고, 캐시 모드는 MCDRAM을 마지막-레벨(Last-level)의 캐시로 사용한다. 하이브리드 모드는 MCDRAM을 두 부분으로 분할하여 마지막-레벨 캐시와 DRAM과 같이 주소 지정 가능한 메모리로 사용한다. 본 논문에서는 사용자가 메모리 주소를 지정하여 MCDRAM을 사용할 수 있는 플랫 모드를 대상으로 한다 [1, 17].

2. Proposed Technique

2.1 Overview

BDM(Bandwidth-aware Data Migration)이라고 명명된 제안된 기법은 MCDRAM과 DRAM에 할당된 응용프로그램의 프로파일링 데이터를 통해 하이브리드 메모리를 효율적으로 사용

Algorithm 2 $Monitoring(L, W)$

Input: $L \leftarrow$ the list of running application
$W \leftarrow$ the list of windows for monitoring data
/*
M : the list of monitoring data for applications
i : the index of application
*/
for $i \leftarrow L_0$ to $length(L)$ **do**
1. $M_i \leftarrow getMonitoringData(i)$
2. /* $M_i.band$: current bandwidth of threads */
3. /* $M_i.pf$: # of page faults */
end for
4. insert M to W_0

하는 것을 목표로 한다. BDM은 KNL 프로세서의 성능향상을 위해 세 단계로 나누어 수행된다. 먼저, 하드웨어 모니터링 툴을 사용하여 응용프로그램을 실시간으로 모니터링하고 그 결과를 저장한다. 다음으로, 모니터링 된 데이터를 활용하여 더 많은 메모리 대역폭을 요구하는 응용프로그램을 선택한다. 마지막으로, MCDRAM과 DRAM에 응용프로그램의 페이지를 동적으로 이주(migration) 한다. 알고리즘 1은 BDM의 전체적인 알고리즘을 나타낸다. 실행중인 응용프로그램이 있는 경우, BDM은 실행중인 응용프로그램 리스트 L을 생성한다. 리스트 L은 각 응용프로그램의 PID와 TID를 저장 후, BDM의 세 단계를 반복적으로 실행한다. 각 time window 에는 응용프로그램의 모니터링 데이터가 W = {W0, W1, ...,Wn}에 저장된다 (line 2). Fig. 3은 리스트 W의 전체적인 구조를 나타낸다. W0은 현재 time window, W1은 이전 time window이며 Wn는 n번째 이전 time window를 나타낸다. 리스트 L과 W로 Selection 단계를 거친 후, BDM은 이주를 수행할 후보 응용프로그램을 반환한다(line 3). 선택된 응용프로그램의 페이지가 다음 이주 단계를 통하여 MCDRAM으로 이주된다. 각 단계에 대한 세부동작은 이어지는 절에서 자세하게 다룬다.

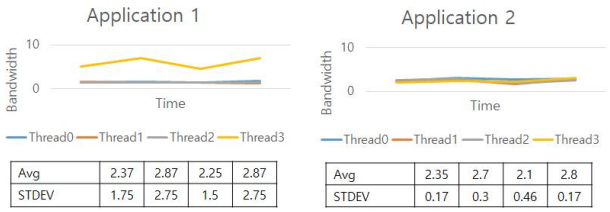
2.2 Application Monitoring

시스템이 실행되는 동안 BDM은 하드웨어 모니터링 툴을 사용하여 응용프로그램을 모니터링 한다. 대부분의 HPC 프로세서는 perf_event[18] 및 LIKWID[19]과 같은 성능 모니터링 툴을 하드웨어적으로 지원한다. 이 논문에서는 perf_event를 사용하였다.

알고리즘 2는 모니터링 단계에서 수행되는 동작을 나타낸다. 실행중인 응용프로그램 리스트 L과 각각의 응용프로그램의 모니터링 데이터 리스트 W를 입력한다. M은 time window에서 모니터링을 수행하는 쓰레드들의 데이터 목록을 나타내며, Mi는 i번째 응용프로그램을 나타낸다. 모니터링 데이터로는 쓰레드별 메모리 대역폭(Mi.band), 페이지 부재(Mi.pf)가 있으며, 해당 데이터를 Mi의 각 유형에 해당하는 항목에 저장한다. Mi.band는 이주를 수행할 응용프로그램을 선택하기 위해 다음 단계에서 사용되며 해당 작업은 현재 시스템에서 수행중인 모

Algorithm 3 $Selection(L, W)$

Input: $L \leftarrow$ the list of running application
$W \leftarrow$ the list of windows for monitoring data
Output: $b \leftarrow$ the selected application id
/*
S : the sorted list of applications
i : the index of applications
pf_{avg} : the averaged page faults
*/
1. $S \leftarrow regressionAndSort(W)$
2. $pf_{avg} \leftarrow getAvgPageFault(W_0)$
for $i \leftarrow 0$ to $length(S)$ **do**
3. if $S_i.pf > pf_{avg}$ then continue
4. $b \leftarrow getAppId(S_i, L)$
5. return b
end for



Avg	2.37	2.87	2.25	2.87
STDEV	1.75	2.75	1.5	2.75

Avg	2.35	2.7	2.1	2.8
STDEV	0.17	0.3	0.46	0.17

Fig. 4. Example of Application Selection

든 응용프로그램에 대해 수행된다. 메모리 대역폭을 계산하기 위해서 KNL 프로세서의 L2_REQUESTS.MISS 이벤트를 사용한다. 해당 이벤트는 L2 캐시에서 일어난 miss 횟수를 측정하며 해당 이벤트에 캐시라인의 크기(64Bytes)를 곱하여 메모리 대역폭을 측정할 수 있다. 페이지 부재 횟수를 측정하기 위해 perf_event에서 자체적으로 지원하는 이벤트를 사용한다. 마지막으로, 현재 모니터링 데이터가 저장된 M을 큐의 W0 위치에 삽입 한다 (line 4). BDM은 적은 수의 모니터링 데이터만 저장하기 때문에 전체 메모리에 비교했을 때 스토리지 부담(Overhead)이 매우 적다.

2.3 Application Selection

제안된 BDM은 모니터링 단계에서 수집된 각 응용프로그램의 정보를 기반으로 메모리 대역폭이 더 필요하다고 판단되는 응용프로그램을 선택한다. 알고리즘 3은 응용프로그램을 이주 후보로 선택하는 단계의 동작을 나타낸다. regressionAndSort 함수는 모니터링 단계에서 전달된 W(즉, Wtime.pid.band)에 저장된 각 응용프로그램의 쓰레드들의 평균 메모리 대역폭에 선형회기 분석을 수행하여 다음 메모리 대역폭을 예측한다. 그 후, 해당 메모리 대역폭이 쓰레드 간에 균등하게 분산되고 메모리 대역폭 사용량이 많은 응용프로그램 순으로 정렬을 수행한

다(line1). Fig. 4는 regressionAndSort 함수에서 수행되는 선택 알고리즘의 예제를 나타낸다. Application1과 Application2는 서로 비슷한 수치의 메모리 대역폭이 관측되었지만 BDM은 쓰레드 간에 메모리 대역폭이 균등하게 분산된 Application2에

Algorithm 4 $Migration(b, W)$

Input: $b \leftarrow$ the selected application id
$W \leftarrow$ the list of windows for monitoring data
/*
t : the threshold ratio of MCDRAM use
*/
if $isMigrationPossible(t, b)$ **then**
1. $migrationToMCDRAM(b, W)$
end if

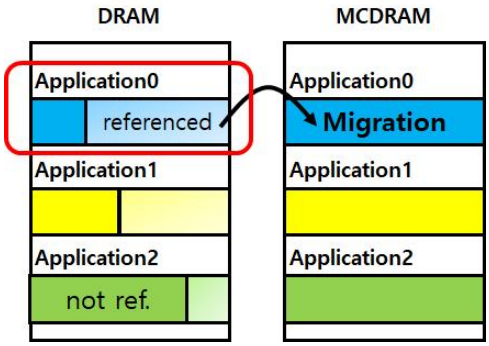


Fig. 5. Migration Example

높은 우선순위를 부여한다. 시스템에서 실행중인 응용프로그램들의 정렬 결과는 리스트 S에 저장된다.

메모리 지역성이 낮은 응용프로그램을 필터링하기 위해 모니터링 절차에서 저장한 페이지 부재 횟수(Wtime.pid.pf)를 이용한다. BDM은 정렬된 리스트 S에 대해 순서대로 평균 페이지 부재와 비교를 수행한다(line3). 만약, 첫 번째 이주 후보 응용프로그램이 평균 페이지 부재보다 높은 값을 가지고 있다면, 응용프로그램 리스트 S의 다음 응용프로그램에 대한 검사를 수행한다. 모든 동작이 완료되면 선택 알고리즘은 이주 후보 응용프로그램 b를 반환 한다 (line 5).

2.4 Data Migration

알고리즘 4는 페이지 이주 절차를 나타낸다. KNL 프로세서에 구성된 MCDRAM(16GB)의 용량 제한 때문에 이주를 수행하기 전에 먼저 isMigrationPossible 함수에서 해당 응용프로그램이 이주가 가능한지 여부를 먼저 확인한다. 선택한 응용프로그램의 메모리 사용량을 포함한 총 메모리 사용량이 임계값 t(예: 90%)를 초과하지 않는지 확인한다. MCDRAM 사용량이 임계 값 t보다 작을 경우, time window W에서 MCDRAM으로 해당 응용프로그램의 페이지를 이주 한다 (line 1). 응용프로그램의 전체 페이지를 이주할 경우 페이지 이주에 대한 성능 부담이 크기 때문에 BDM에서는 응용프로그램에서 최근 참조된 페이지를 선택적으로 이주 한다 [20, 21]. 각 어플리케이션에서 최근 참조된 페이지를 확인하기 위해선 리눅스 시스템의 /proc/<PID>/smaps를 활용한다. 해당 파일에는 각 페이지의 가상주소와 참조 횟수를 포함하고 있다. Fig. 5는 이주가 수행되는 예를 보여준다. MCDRAM의 사용량이 임계 값 t보다 클 경우 BDM은 페이지 이주 동작을 수행하지 않는다.

IV. Experiments

1. Experimental Methodology

본 논문에서 제안한 BDM 기법의 성능결과를 측정하기 위해 KNL 프로세서가 탑재된 시스템을 사용하여 실험을 수행하였다. 실험 시스템에는 Intel Xeon Phi(TM) CPU 7250@1.40GHz, 소켓 당 68코어, 코어 당 4개 쓰레드, 하이퍼스레딩 기술로 총 272개의 쓰레드가 장착되어 있다. 또한, 96GB DDR4(DRAM)와 16GB HBM(MCDRAM)이 포함되어 있다. 성능 평가를 위한 벤치마크 프로그램으로 전산 유체 역학과 관련된 NAS Parallel Benchmark (NPB)를 사용하였다 [22]. NPB는 5개의 커널 벤치마크(IS, EP, CG, MG, FT)와 3개의 가상 벤치마크(BT, SP, LU)로 구성된다. 모든 실험에는 표준 테스트 클래스 (CLASS-C)를 사용하였다. Table 1은 NPB 벤치마크의 개별 프로그램을 시스템에서 단독으로 수행했을 때의 평균 실행 시간, 부동 소수점 동작, 메모리 액세스, 최고 메모리 사용량을 나타낸다.

MCDRAM의 용량보다 큰 메모리 사용량이 필요한 시스템 상황을 가정하기 위해 각 NPB 벤치마크 프로그램을 중복으로 동시에 (즉, bt0, bt1, bt2, bt3) 실행하였고, 모든 실험은 응용프로그램 당 4개의 쓰레드를 사용하였다. 따라서 32개의 응용프로그램이 병렬로 실행되고 총 128개의 쓰레드가 생성된다. 성능 비교를 위한 실험에서 BDM의 모니터링 주기는 100ms, 각 응용프로그램의 모니터링 정보를 저장하기 위한 윈도우의 수는 5개, MCDRAM의 임계 사용량은 90%로 설정하였다. Intel KNL 프로세서는 타일-구조로 구성되어 있기 때문에 쓰레드가 할당되는 타일 및 코어의 위치에 따라 성능이 달라질 수 있다. 본 연구에서는 기본 시스템 정책을 이용하여 랜덤하게 코어를 할당하였다. 적절한 타일 위치에 쓰레드를 할당하는 방법에 관한 내용은 본 논문의 범위를 벗어나기 때문에 본 논문에서 다루지 않는다.

2. Experimental Results

Fig. 6은 시간에 따른 NPB 벤치마크의 메모리 대역폭 변화를 나타낸다. 그래프의 x-축은 시간을 나타내고 y-축은 메모리 대역폭을 나타낸다. 병렬로 실행된 32개의 벤치마크 중 가상 벤치마크인 bt0과 커널 벤치마크인 cg0을 표시하였다. 또한, BDM에 의해 페이지 이주가 수행된 지점을 그래프 상에 Migration Point로 표시하였다. bt0의 경우 Default에서 메모

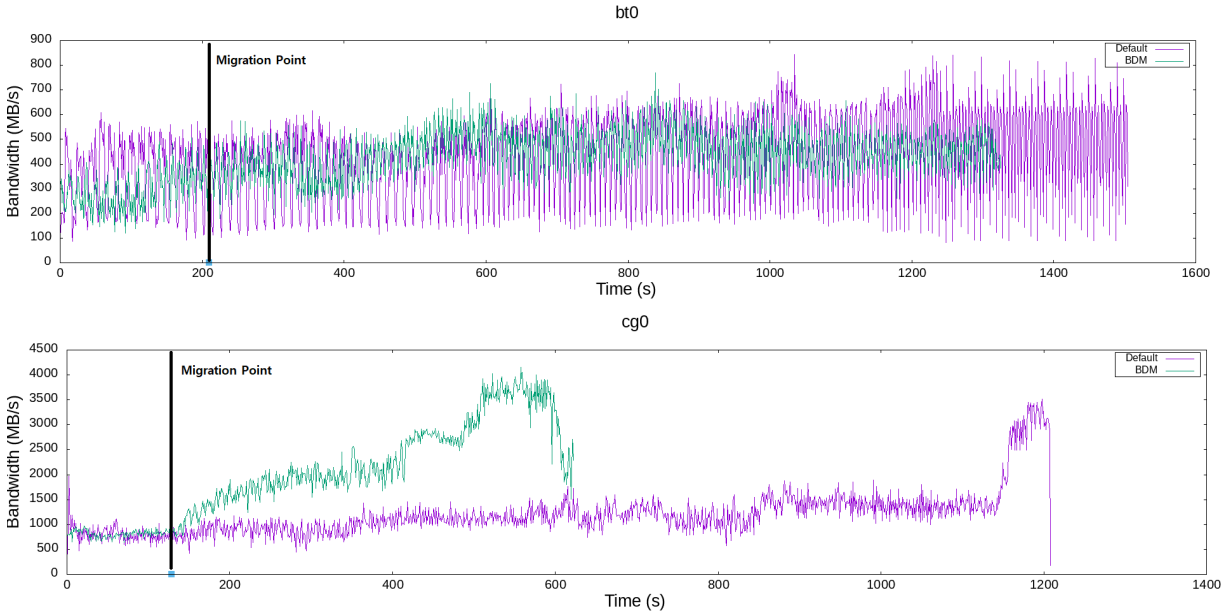


Fig. 6. Memory Bandwidth Comparison: bt0 (upper), cg0 (bottom)

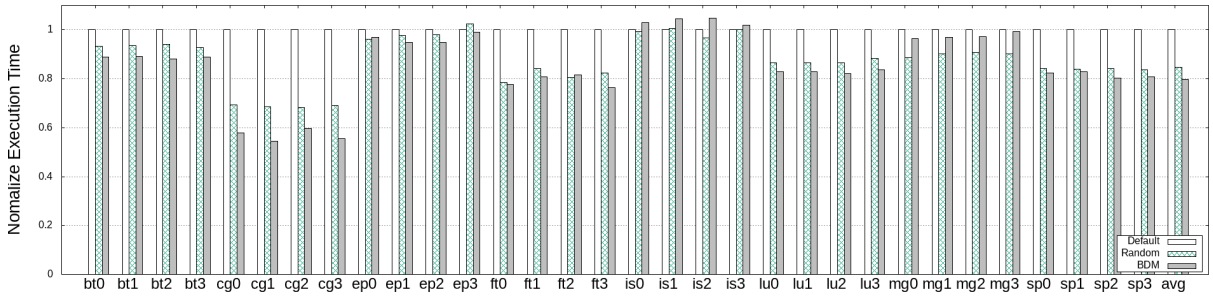


Fig. 7. Execution Times Normalized to that of The Baseline

리 대역폭의 상하한 차이가 비교적 크게 발생하는 것을 확인할 수 있다. 제안된 BDM을 통해 MCDRAM으로 페이지 이주가 수

Table 1. NAS Parallel Benchmarks (NPB)

Name	Average execution time (sec.)	FP operation (GFlop)	Memory Accesses (read/write)	Peak Memory use(MB)
IS.C	31	23	758/338	1,572
EP.C	462	494	2,739/1,028	20
CG.C	319	261	31,174/810	1,102
MG.C	129	213	7,507/2,940	3,536
FT.C	335	837	18,555/17,270	7,188
BT.C	920	2,560	45,682/17,270	1,676
SP.C	626	1,918	92,526/47,727	1,416
LU.C	733	2,504	84,716/38,302	760

행된 후 메모리 대역폭의 상하한 차이가 크게 감소하여 실행 시간이 단축된 것으로 분석할 수 있다. cg0은 BDM이 페이지 이주를 수행한 후, 메모리 대역폭이 급격하게 상승하는 것을 확인 할 수 있는데 Table 1에 보인 것과 같이 cg벤치마크는 FP 연산 대비 메모리 사용량이 많은 벤치마크이다. 따라서 MCDRAM으로 페이지 이주 후 다수의 메모리 요청이 빠르게 처리되어 실행 시간이 크게 감소한 것을 확인할 수 있다.

BDM의 성능을 평가하기 위해 평균 실행시간을 측정하였다. Fig. 7은 5개의 모니터링 윈도우 크기를 적용하여 벤치마크들을 100회 실행한 결과의 평균 실행시간을 나타낸다. Default는 BDM이 적용되지 않고 MCDRAM과 DRAM을 인터리빙(interleaving) 모드로 사용하였고, Random은 응용프로그램을 랜덤하게 선택하여 MCDRAM으로 이주한 경우를 나타내고 제안된 BDM의 실행 시간을 비교하기 위해 추가하였다. Default를 기준으로 실행 시간을 정규화 하였다. BDM과 Random은 대부분의 벤치마크에서 실행시간을 단축시켰다. 특히, CG(42%~45%), FT(13%~17%), SP(14%~17%) 벤치마크는 실행시간이 큰 폭으로 감소하였다. BDM은 더 많은 메모리 요청을 가진 응용프로그램을 효과적으로 선택하여 MCDRAM으로 이주를 수행하기 때문에 Random 보다 성능을 더욱 향상시키는 것을 확인하였다. 그러나 IS와 같은 일부 벤치마크에서는 실행시간이 증가(2%~7%)한 경우가 발생했는데, 해당 벤치마크는 정수 정렬 연산과 랜덤 메모리 액세스 패턴을 가지고

있으며 짧은 수행시간의 특징을 갖는다. 이주하는 동안 응용프로그램은 중지되어 실행시간이 짧은 경우 이주 오버헤드가 상대적으로 커 보일 수 있으며 메모리 액세스 패턴 상 페이지 부재가 많이 일어나 높은 메모리 대역폭을 지원하는 MCDRAM이 효과적으로 사용되지 않았다. 평균적으로 BDM과 Random은 각각 20%, 14% 실행시간이 감소하였다.

V. Conclusions

하이브리드 메모리는 고성능 컴퓨팅 시스템의 성능 한계를 극복하기 위한 유망한 기술로 주목받고 있다. 본 논문에서는 BDM(Bandwidth-aware Data Migration)으로 명명된 하드웨어 성능 모니터링 툴을 활용한 동적 데이터 이주 전략을 제안하였다. BDM은 시스템에서 실행중인 응용프로그램 쓰레드의 메모리 사용 현황을 주기적으로 취합하여 페이지 이주를 수행할 응용프로그램을 선택한다. 메모리 대역폭 요구가 큰 응용프로그램의 페이지를 고대역 메모리로 동적으로 이주하여 시스템의 성능 효율화를 성취하였다. 하이브리드 메모리가 구현된 Intel사의 KNL 프로세서를 대상한 성능 평가를 통해 제안된 BDM이 시스템의 성능을 향상시키는 것을 검증하였다.

향후 연구는 BDM 기법을 다른 구성의 하이브리드 메모리 시스템으로 적용하여 범용성을 높이고, 타일-구조의 단일 프로세서 환경에서 다수의 노드가 결합된 실제 슈퍼컴퓨터 환경인 멀티-노드 환경 시스템에 적용하는 것이다.

REFERENCES

[1] A. Sodani, Knights landing (KNL): 2nd Generation Intel Xeon Phi processor, in IEEE Hot Chips 27 Symposium, Cupertino, CA, USA, pp. 1-24, 2015.

[2] R. Espasa, Larrabee - A Many-Core Intel Architecture for Visual Computing, in ACM CF, Ischia, Italy, pp. 225-225, 2009.

[3] A. K. Singh, M. Shafique, A. Kumarm, and J. Henkel, Mapping on multi/many-core systems - survey of current and emerging trends, in DAC, Austin, TX, USA, 2013.

[4] I. B. Peng, R. Gioiosa, G. Kestor, P. Cicotti, E. Laure, and S. Markidis, Exploring the Performance Benefit of Hybrid Memory System on HPC Environments, in IPDPSW. IEEE, Lake Buena Vista, FL, USA, pp. 683-692, 2017.

[5] I. B. Peng, R. Gioiosa, G. Kestor, J. S. Vetter, P. Cicotti, E. Laure, and S. Markidis, Characterizing the performance benefit of hybrid memory system for HPC applications, Parallel Computing, vol. 76, pp. 57-69, May, 2018.

[6] O. Mutlu, Memory scaling: A systems architecture perspective, in IMW. IEEE, Monterey, CA, USA, pp. 21-25, 2013.

[7] J. Meena, S. Sze, U. Chand, and T.-Y. Tseng, Overview of emerging nonvolatile memory technologies, Nanoscale Research Letters, vol. 9, no. 1, p. 526, September, 2014.

[8] R. F. Freitas and W. W. Wilcke, Storage-class memory: The next storage system technology, IBM Journal of Research and Development, vol. 52, no. 4.5, pp. 439-447, July, 2008.

[9] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, Scalable high performance main memory system using phase-change memory technology, ACM SIGARCH Computer Architecture News, vol. 37, no. 3, p. 24, June, 2009.

[10] Y. Ro, M. Sung, Y. Park, and J. H. Ahn, Selective DRAM cache bypassing for improving bandwidth on DRAM/NVM hybrid main memory systems, IEICE Electronics Express, vol. 14, no. 11, pp. 20 170 437-20 170 437, May, 2017.

[11] G. Dhiman, R. Z. Ayoub, and T. Rosing, PDRAM - a hybrid PRAM and DRAM main memory system. in DAC, San Francisco, CA, USA, pp. 664, 2009.

[12] Y. Tan, B. Wang, Z. Yan, Q. Deng, X. Chen and D. Liu, UIMigrate: Adaptive Data Migration for Hybrid Non-Volatile Memory Systems, IEEE Design, Automation & Test in Europe Conference & Exhibition, Florence, Italy, 2019.

[13] M. Lee, D. Kang and Y. Eom, M-CLOCK: Migration-optimized Page Replacement Algorithm for Hybrid Memory Architecture, ACM Transactions on Storage, vol. 14, no. 3, pp. 1-17, November, 2018.

[14] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, A. Gavrilovska, Kleio: A Hybrid Memory Page Scheduler with Machine Intelligence, ACM International Symposium on High-Performance Parallel and Distributed Computing, Phoenix, AZ, USA, pp. 37-48, 2019.

[15] H. T. Mai, K. Park, H. Lee, C. Kim, M. Lee and S. Hur, Dynamic Data Migration in Hybrid Main Memories for In-Memory Big Data Storage, ETRI Journal, vol. 36, no. 6, December, 2014.

[16] Colfax, Clustering Modes in Knights Landing Processors, 2016.

[17] J. Jeffers, J. Reinders, and A. Sodani, Knights Landing architecture. Morgan Kaufmann, Jan. 2016.

[18] V. M. Weaver, Linux perf event Features and Overhead, in FastPath Workshop, Austin, TX, USA, 2013.

[19] J. Treibig, G. Hager, and G. Wellein, LIKWID: A Lightweight Performance-Oriented Tool Suite for x86

Multicore Environments, in ICPPW. ACM, San Diego, CA, USA, pp. 207-216, 2010.
[20] L. E. Ramos, E. Gorbatory, and R. Bianchini, Page placement in hybrid memory systems, in ICS. ACM, Tuscon, Arizona, USA, pp. 85-95, 2011.
[21] D. Shin, S. Park, S. Kim, and K. Park, Adaptive page grouping for energy efficiency in hybrid PRAM-DRAM main memory, in ACM Research in Applied Computation Symposium, San Antonio, Texas, USA, pp. 395-402, 2012.
[22] D. Bailey, J. Bartion, T. Lasinski and H. Simon, The NAS Parallel Benchmarks, Technical Report RNR-91-002, NASA Ames Research Center, August 1991.

Authors

Jongmin Lee received the B.S. degree in computer science and engineering from Dankook University, Korea, in 2008. He received the integrated master's Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 2015. Dr. Lee joined the faculty of the Department of Computer Engineering at WonKwang University, Iksan, Korea, in 2018. He is currently an assistant Professor in the Department of Computer Engineering, WonKang University. He is interested in computer architectures, memory systems, and embedded systems/software.