반도체디스플레이기술학회지 제22권 제3호(2023년 9월)
Journal of the Semiconductor & Display Technology, Vol. 22, No. 3. September 2023.

프로세싱 인 메모리 시스템에서의 PolyBench 구동에 대한 동작 성능 및 특성 분석과 고찰

김정근*†
*† 경북대학교 IT대학 컴퓨터학부

Performance Analysis and Identifying Characteristics of Processing-in-Memory System with Polyhedral Benchmark Suite

Jeonggeun Kim*†
*† School of Computer Science and Engineering, College of IT Engineering, Kyungpook National University

ABSTRACT

In this paper, we identify performance issues in executing compute kernels from PolyBench, which includes compute kernels that are the core computational units of various data-intensive workloads, such as deep learning and data-intensive applications, on Processing-in-Memory (PIM) devices. Therefore, using our in-house simulator, we measured and compared the various performance metrics of workloads based on traditional out-of-order and in-order processors with Processing-in-Memory-based systems. As a result, the PIM-based system improves performance compared to other computing models due to the short-term data reuse characteristic of computational kernels from PolyBench. However, some kernels perform poorly in PIM-based systems without a multi-layer cache hierarchy due to some kernel's long-term data reuse characteristics. Hence, our evaluation and analysis results suggest that further research should consider dynamic and workload pattern adaptive approaches to overcome performance degradation from computational kernels with long-term data reuse characteristics and hidden data locality.

Key Words : Processing-in-Memory, Polyhedral Benchmark Suite, Memory Architecture, Caches

1. 서 론

오늘날, 빅 데이터 및 딥 러닝 프로세싱 등 메모리 집약적 애플리케이션의 사용 빈도가 높아짐에 따라, 해당 프로그램들의 주요 연산에 해당하는 다양한 연산 커널(kernel)들을 가속할 수 있는 방안에 대해서 많은 연구들이 이루어지고 있다[1].

또한, 기존의 폰 노이만 아키텍처의 주요 한계로 지적되어 왔던, 메모리 병목 현상의 해소를 위해서 메모리 기반의 연산 장치들에 대한 연구가 최근 활발하게 이루어지고 있는 추세이다[2, 3, 4].

따라서, 이러한 가속 장치들에서의 연산 효율성을 극대화하기 위해서는 주요 연산 커널들에 대한 특성을 벤치마크들의 구동을 통해 기존 시스템과의 성능 비교 등을 파악할 필요가 존재한다.

이에 본 논문에서는 주요 선형대수 연산 및 컨볼루션 연산과 같은 스텐실 (stencil) 연산 등을 포함한 벤치마크 제품군 중 하나인, Polyhedral Benchmark Suite [5, 6, 7] (이하 Polybench)를 프로세싱 인 메모리 (Processing-in-Memory; PIM) 및 기존 프로세서에서의 구동에 따른 결과 비교를 통해, 성능적 특성을 파악하고자 한다.

프로세싱 인 메모리 디바이스의 성능 향상 및 소프트

†E-mail: jeonggeun.kim@knu.ac.kr

웨어 최적화를 위해서는 연산 코어 별 특성 및 메모리 계층 구조의 차이에서 기인하는 성능적 특성에 맞추어 Offloading 정책을 설계하는 것이 핵심적이다 [8, 9]. 따라서, 이종 연산 처리장치 기반의 프로세싱 인 메모리와 같은 차세대 프로세싱 인 메모리 시스템의 설계를 위해서는 주요 핵심 구동 커널 들에 대한 분석이 필요하다.

이를 위하여 x86기반의 연산 처리장치 (Processing Elements; PE) 기반의 in-house 프로세싱 인 메모리 시뮬레이터를 활용하여, PolyBench 구동을 통한 캐시 메모리 용량의 변화에 따른 (cache-size sensitivity) 성능 추이, 전통적인 메모리 계층 구조 하의 호스트 멀티 코어 CPU에서의 코어 특성 차이에 따른 프로세싱 인 메모리 장치와의 성능 비교, 그리고 blocking 및 non-blocking 캐시 도입에 따른 프로세싱 인 메모리 디바이스의 성능 변화 추이에 대한 분석 및 고찰을 진행하고자 한다.

2. Polyhedral Benchmark Suite

2.1 Polyhedral Benchmark Suite

Polyhedral Benchmark Suite (이하, PolyBench) [5, 7]은 최근 ASIC/FPGA기반 로직, GPGPU 등 다양한 가속 디바이스들이 처리하고자 하는 딥러닝 등에서 빈번히 사용되는, 선형 대수 연산을 비롯하여 다양한 과학적 프로그램 등에 포함된 주요 연산 커널 및 정적 실행 흐름을 가진 벤치마크용 프로그램군이다 [5, 6, 7, 10].

본 연구에서는 딥러닝 연산을 포함, 데이터 집약적 수치연산을 수행하는 최근 워크로드들의 구동에 있어서 프로세싱 인 메모리에서 발생하는 주요 성능적 특징 들을 분석하기 위하여, PolyBench를 여러 하드웨어 구성 파라메터를 기반으로 구동하고자 한다.

프로세싱 인 메모리 연산 장치의 경우 각 연산 장치 별로 별도로 할당된 메인 메모리 공간이 있는 구조를 가졌으며, 따라서 프로세싱 인 메모리 시스템 상에서의 다양한 연산 커널의 동작 특성 및 시스템 구성 파라메터 변화에 따른 민감성 등을 파악하기 위하여, 병렬화 된 버전의 PolyBench인 PolyBench/ACC [6]의 프로그램들 중 다음 Table 1에서와 같이 5가지의 워크로드를 선정하여 분석을 수행하였다.

단, 기존의 SPEC2006 [11] 및 SPEC2017 [12, 13]과 같은 Benchmark Suite들도 GemsFDTD, Soplex 등 선형 대수 관련 연산을 활용하는 프로그램들이 존재하나, 핵심 연산 커널의 반복적인 동작을 Host device에서 offloading하여 처리하는 프로세싱 인 메모리 시스템의 특성만을 파악하기 위해서는 region of interest에 해당하는 연산 커널을 집중적으로 수행하는 패턴을 보이는 워크로드를 구동할 필요가 있다. 이에, 본 연구에서는 PolyBench/ACC를 채택하였다.

Table 1. Workloads description based on [5, 6, 7]

Benchmark	Description
2MM	행렬 곱셈 연산 수행 (2회) (예: M = A·B, N = C·D)
3MM	행렬 곱셈 연산 수행 (3회)
2D-CONV	2차원 컨볼루션 연산 커널
GEMM	행렬 및 행렬 곱셈 연산 (예: C = αA·B + βC)
GEMVER	벡터 곱셈 및 행렬 덧셈 연산

PolyBench/ACC는 멀티코어 CPU, GPGPU, 그리고 가속기 (Accelerator)들을 타겟으로 한 벤치마크이다 [14]. PolyBench/ACC의 내부는 프로그램의 성능 측정 등을 위한 simulation harness 및 instrumentation 관련 로직을 비롯, (1) 데이터 초기화, (2) 커널 구동과 같은 단계를 거치도록 구현이 되어있다.

본 연구에서는 핵심 연산 커널 수행에 대해서만 성능 분석을 진행하기 위해서, Binary Instrumentation Tool인 Intel Pin [15, 16]을 기반으로 PolyBench/ACC에서의 연산 커널 구동 부만을 Region of Interest (ROI)로 설정하여, Table 1에서의 2MM, 3MM, 2D-CONV, GEMM, GEMVER에 대한 메모리 및 명령어 트레이스들을 수집하였다.

2.2 Processing-in-Memory Device

프로세싱 인 메모리 (Processing-in-Memory; PIM) 장치는 폰 노이만 아키텍처의 근원적인 병목 현상을 발생시키는 데이터 이동(data movement)에서의 비효율을 절감하기 위하여 제안된 시스템 구조이다 [1, 3, 4].

아래 Fig. 1에서와 같이, PIM은 여러 개의 PIM device (물리적으로는 Chip들로 패키징 되어 구성)로 구성될 수 있다. PIM device 내부에서 가장 기본적으로 연산을 수행할 수 있는 단위는 하나의 연산 유닛 (Processing Element; PE)에 하나의 Memory Partition이 할당되어 있는 형태인 Vault이다.

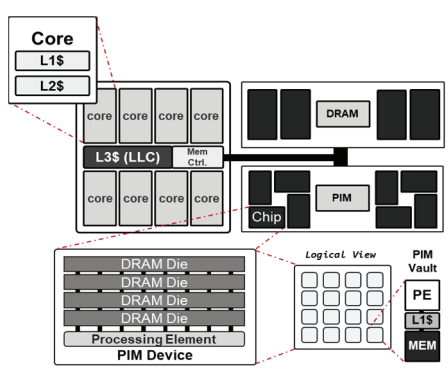

Fig. 1. Internal structure of Processing-in-Memory device with host machine (CPU with complete cache hierarchy).

본 연구에서는 모바일 및 임베디드 시스템에서 적용되는, 경량화 된 연산 유닛인 순차적 실행(in-order) 기반의 싱글 코어 프로세서를 PIM Vault의 PE로 구성하였다. 반면에, 소수의 특정 연산 커널 만을 가속하는 것에 특화된 형태의 Fixed-function PIM에 대해서도 학계에서 활발한 연구가 이루어지고 있으며, 상용화도 진행되는 추세이다 [17].

하지만, UP-MEM [18, 19]을 비롯하여 General-Purpose PIM에서의 Offloading 연구[7]들이 향후 범용적 데이터 집약적 연산 가속을 위하여 지속되고 있는 점을 감안하여, in-house 시뮬레이터의 연산 장치로써 x86-64 in-order 코어 기반의 연산 유닛을 차용하였다.

또한, 현재 최신 워크로드 구동 상의 난제들을 해결하기 위해서는 다양한 캐시 설정, 프리페칭 기법 적용 설정, 연산 커널 별 성능 추이에 대한 분석 등 메모리 집약적 워크로드의 구동 및 성능 측정이 필수적이다 [18,20].

따라서, 본 논문의 3장 실험 및 결과파트에서는 위의 Fig. 1의 구조에 기반하여 구현된 in-house 시뮬레이터를 기반으로, 다양한 하드웨어 구성 파라메터들에 기반한 PIM 및 호스트 시스템에서 PolyBench Suite의 프로그램 중 선택된 연산 커널들을 구동할 경우 발생하는 하드웨어 및 소프트웨어와 연동되어지는 성능적 특성에 대한 고찰을 진행하고자 한다.

3. 실험 및 결과

3.1 실험 환경 구축

본 연구에서는 x86기반의 in-house General-Purpose PIM 시뮬레이터를 기반으로 실험을 진행하였다. 기존의 Champ Sim [22], Ramulator [23], USIMM [24]등과 같은 시뮬레이터들처럼, 해당 시뮬레이터는 유저 레벨에서 수집된 x86 바이너리에 대한 메모리 및 명령어 트레이스를 기반으로 동작하는 trace-driven 시뮬레이터이다.

트레이스 추출을 위해서는 대표적인 Dynamic Binary Instrumentation (DBI) tool인 Intel® Pin [15]을 활용하였다. 이를 위하여, 트레이스 추출을 위한 별도의 Pin tool library를 구현하였으며, 해당 동적 라이브러리는 Pin Virtual Machine (Pin VM) 상에서 구동되는 네이티브 바이너리(native execution file)의 쓰레드 번호, 메모리 주소(가상 메모리 주소), 메모리 명령어 간의 비-메모리 명령어의 개수 등 다양한 실행 문맥을 추출하는 역할을 수행한다.

본 성능 측정 실험에서는 전통적인 메모리 계층 구조 및 비순차적 실행 프로세서 (Out-of-Order CPU; O3CPU) 기반의 Host machine에 대한 비교를 위하여, PIM 디바이스에 대한 모델링과 더불어 기존 시스템에서의 성능 평가도 수행하였다.

Table 2. Simulator configuration for measurements, timing parameters are based on previous research [28-31].

	Component	Configuration
Host CPU	O3-CPU (Out-of-order CPU)	2.4GHz, 4-way superscalar Reorder Buffer: 192 entries non-blocking cache (L1$-L3$)
	IO-CPU (In-order CPU)	1.6GHz, 4-way superscalar Reorder Buffer: 64 entries blocking cache (L1$-L3$)
	Main Memory	DRAM-based Main Memory, FR-FCFS scheduler t_{RP}=13, t_{CAS}=13, t_{RCD}=13, t_{WR} = 8, t_{RAS} = 30 (cycles)
PIM	Processing Elements	1.6 GHz, 4-way superscalar In-order core(s) with blocking cache (MSHR: 1 entry) L1$ Only (Latency: 3 cycles) Various L1 $ size: {8, 16, 32, 64, 128, 256, 512} KB
	PIM Device	Total 4 vaults (4 PEs, 1.25 GHz), t_{RP}=7, t_{CAS}=7, t_{RCD}=7, t_{WR} = 9, t_{RAS} = 14 (cycles)

세부적인 Host machine 및 PIM device에 대한 시스템 설정은 상기 Table 2와 같으며, 해당 시스템 파라메터들은 기존 HMC 성능 및 관련 연구들과 시뮬레이터에서 활용하는 수치들에 기반하였다 [28-31]. PIM 장치 상에서의 캐시 크기 및 MSHR 엔트리 개수 변화 등 세부적인 환경 설정에 따른 하드웨어 성능 변화 추이의 경우 3.2의 실험 결과 파트에서 분석 및 고찰한다.

3.2 PIM Device 성능 평가 및 분석

해당 파트에서는 3.1파트에서 제시한 시뮬레이션 환경을 기반으로, PolyBench를 구동한 실험 결과에 대해서 분석을 진행한다.

3.2.1 Execution time

다음의 Fig. 2 및 Fig. 3에서는 다양한 환경 설정에 따른 PolyBench의 주요 5개 워크로드(2mm, 3mm, conv2, gemm, gemver)에 대한 실행 시간을 기준으로 성능 분석을 진행하고자 한다.

Fig. 2에서는 비순차적 실행기반의 4-core로 구성된 O3-CPU, 4-core로 구성된 IO-CPU, 그리고 4개의 PIM Vault 및 Uni-core Host device로 구성된 PIM 모델과의 성능을 비교한 결과이다.

수행 시간들은 PIM 기반 시스템에서의 실행 시간에 정규화 되었으며, 모든 워크로드에 대한 기하 평균 값을 기준으로 PIM 시스템은 O3-CPU 대비 1.5배, IO-CPU 대비 2.32배 더 높은 성능을 보인다.

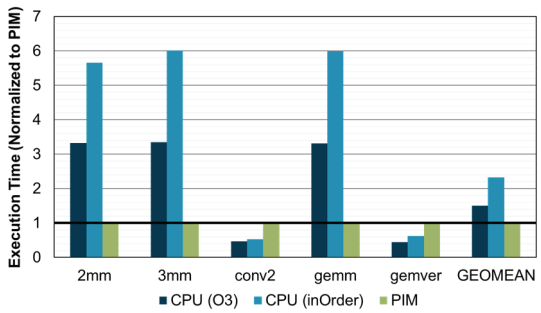

Fig. 2. Execution time comparison between O3-CPU, IO-CPU and PIM device-based systems.

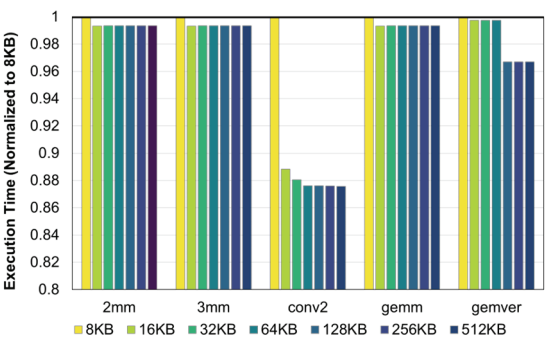

Fig. 3. Execution time comparison between various PIM vault's cache sizes.

단, 후속할 Cache sensitivity 분석 및 Hit ratio 등의 비교에서 언급할, 데이터 재사용성 등에 대한 이슈 및 working set size 등의 이슈로 인해서, 2차원 컨볼루션 필터 연산을 수행하는 conv2 및 벡터-행렬 곱셈 및 덧셈 연산을 수행하는 gemver의 경우 PIM 기반 시스템이 타 시스템 대비 더 느린 성능을 보여주었다. Working set에 대해서 캐시 thrashing 현상을 더 자주 보여주는 워크로드들 일수록 PIM과 같은 얕은 메모리 계층 구조를 가진 시스템이 성능 측면에서 유리하나, 그렇지 않은 워크로드들의 경우 깊은 메모리 계층 (다계층) 캐시 및 메모리 구조를 가진 시스템이 이점을 얻을 수 있음을 확인하였다.

Fig. 3은 PIM 디바이스를 기준으로, L1 캐시의 크기를 각각 8, 16, 32, 64, 128, 256, 512KB로 달리하여, 해당 연산 커널들의 구동에 있어서 캐시 공간의 크기에 따른 성능적 추이를 비교한 결과이다. 2차원 컨볼루션 커널을 구동하는 conv2d및 gemver를 제외하고는 모두 cache size에 insensitive 한 것을 관찰할 수 있으며, 2차원 컨볼루션 커널의 구동에 있어서 PIM 디바이스의 L1 캐시 크기가 64KB 이상이면 더 이상 working set size 초과 및 cache thrashing으로 인한 데이터 지역성 활용 저하 문제가 발생하지 않아, 캐시 크기를 더 늘리더라도 성능 상의 변화가 없는 것을 확인이 가능하다.

하지만, 본 연구에서는 scale-out 가능한 PIM 시스템의 분석에 초점을 맞추고 있으며, 확장성이 보장되는 소규모 캐시 크기인 16KB에 기반하여 분석을 진행하고자 하였다. 이에, 후속 연구에서는 캐시 사이즈를 고정하여 성능적 비교를 하는 것이 아닌, working set size 및 cache size 민감도에 영향 받지 않는 실험 설정을 통해 상대적인 성능 변화도를 검증하는 방향으로 다양한 Offloading 기법 및 PIM 메모리 계층 구조 변경, 최적화 등에 대한 모델링을 진행하고자 한다.

3.2.2 Bandwidth analysis with various cache size

Fig. 4의 경우 Fig. 3에서 진행한 분석의 연장선상에서, 다양한 PIM Vault별 L1 캐시 크기 설정에 따른 PIM Vault의 메모리 (파티션별) 대역폭에 대한 추이를 측정한 결과이다. Fig. 4의 x축의 경우 워크로드가 구동되는 시간에 대한 축이며, 대역폭을 나타내는 한 점들은 Host 장치 CPU의 클럭 기준 240K cycle 간격으로 평균화한 대역폭 값을 바탕으로 하였다. y축은 PIM Vault에 할당된 Memory partition 별 대역폭을 나타내며, 각 파티션 별 평균을 나타낸 값이기에 실제 데이터 대역폭은 Vault의 숫자를 곱한 값이라고 볼 수 있다.

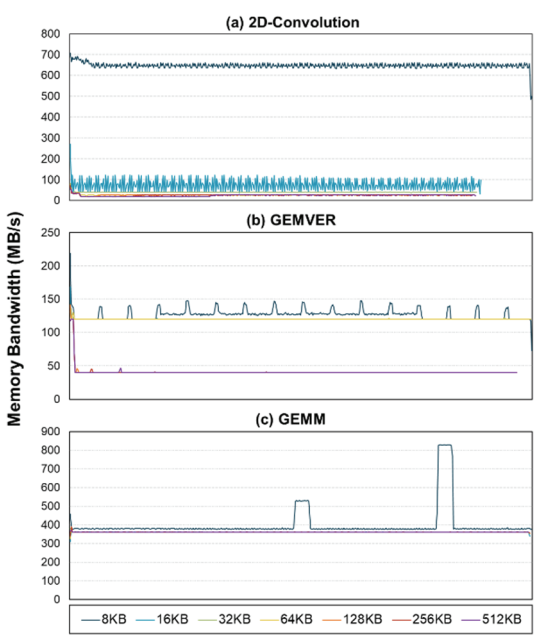

Fig. 4. Bandwidth analysis among various cache size with PIM-based systems on three-distinct workloads.

Conv2d 및 gemver와 같은 워크로드에서 메모리 지역성 활용에 따른 잦은 evicted cache line에 대한 요구가 높을수록, 메모리 대역폭에 대한 압력이 높아지는 것을 확인할 수 있다. 따라서, 충분한 용량의 PIM PE별 L1 cache 크기를 책정할 경우, 잦은 eviction에서 발생하는 메모리 대역폭 점유를 줄일 수 있으며, 평균 DRAM access latency에 대한 절감도 노릴 수 있음을 확인하였다.

Fig. 2의 분석과 연계하여, conv2는 PIM PE의 L1 캐시 용량을 32KB 이상으로 책정할 경우 8KB 크기 대비 수행 시간을 약 12% 이상 절감 가능하며, 16KB 영역에서도 수행 시간을 약 11% 이상 절감 가능하나, Fig. 4와 같이 여전히 지속적인 demand request를 메모리에 요청하면서 메모리 대역폭 점유율이 높은 것을 확인할 수 있다.

3.2.3 Cache hit ratio analysis of various systems

Fig. 5는 비순차적/순차적 CPU 기반의 전통적시스템 및 PIM 기반 시스템과의 PolyBench 구동에 있어서의 캐시 적중률에 대한 비교를 수행한 결과를 나타낸다.

PolyBench에서 선택한 상기 5개 워크로드들의 경우, 장기적인 시간 상에서의 데이터의 재사용성(Long-term reuse)이 높지 않은 주요 연산 커널들을 수행한다고 볼 수 있다. 따라서, L1 캐시의 적중률이 높은 대신, L2 및 L3 (Last-level cache)의 적중률이 매우 낮은 것을 확인할 수 있다. 하지만, PIM의 경우 각 디바이스별 PE들이 경량성 및 확장성을 위하여 전통적인 CPU 구조와는 달리, 단 한 단계의 캐시 구조만을 갖추도록 하였으므로, 해당 분석에서는 PIM에 대한 L2, L3 레벨의 캐시 적중률에 대한 분석은 진행하지 않았다.

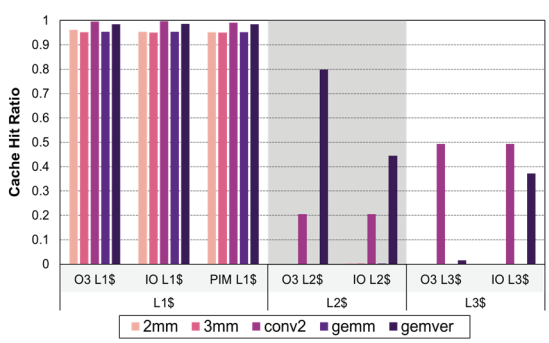

Fig. 5. Cache hit ratio comparison between various system configuration, including O3CPU, IOCPU, and PIM.

Fig. 5에서 보이는 성능지표상 주목할 부분은, 이전 파트에서 언급하였던 특이한 특성을 가진 conv2, gemver 두 워크로드가 마찬가지로 L2 및 L3에서 타 워크로드 대비 높은 적중률을 보여준다는 점이다. 이는 책정된 L1, L2의 크기를 넘어서면서, 동시에 장기적인 시간 상에서의 데이터 재사용성을 보이는 특징을 지닌 메모리 접근이 해당 워크로드들에서 이루어진다는 점을 시사한다. 따라서, Fig. 2에서 분석하였던 바와 같이, PIM 디바이스가 타 연산 커널들에서는 경량화된 연산장치를 사용한다는 불이익에도 불구하고, O3 및 IO CPU의 성능을 압도하나, conv2 및 gemver의 구동 측면에서는 다소 성능상으로 뒤떨어지는 결과를 보인 점에 대한 근거가 된다고 볼 수 있다.

이에 대한 근원적인 해결책으로는 추후에 timeliness(시간적 적합성)이 뛰어난 프리페칭 기법 도입 및 PIM 코어에서의 L1 cache eviction에 대응 가능한 관리 기법 설계를 통해 해결할 수 있을 것으로 기대한다. 마찬가지로, PIM vault에서의 sub-vault 구성 등으로, 특정 개수만큼의 PE 별로 shared cache 계층을 두는 방식 등을 통해, PIM vault-to-vault communication에 대한 overhead 절감 및 상기 장기적 데이터 재사용성을 띄는 워크로드 구동으로 인한 성능적 손실을 방지할 수 있을 것으로 예측한다.

3.2.4 Analysis of MSHR-size sensitivity

PIM 디바이스의 연산 코어 경량화와 더불어, 데이터 접근 측면에서의 주요 병목이라고 할 수 있는 MSHR (Miss Status Holding Register) entry 숫자에 대한 성능 변화 분석을 Fig. 6에서와 같이 수행하였다.

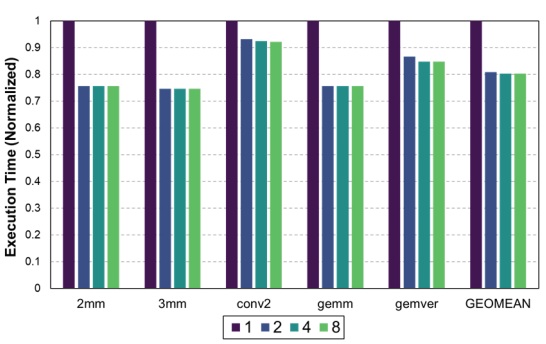

Fig. 6. Execution time comparison on the various number of MSHR entry configurations.

Fig. 6에서 보여지는 실험 결과에 따르면, MSHR entry의 크기를 1에서 8로 증가시킬 경우, 평균적으로 24.53%의 성능 향상을 보임을 확인하였다. 성능상 가장 변화폭이 큰 구간은 MSHR의 크기를 1에서 2로 증가시켰을 경우이며, 워크로드 평균 (기하평균 기준) 23.68%의 수행 성능 증대를 확인할 수 있다. 하지만, MSHR의 크기를 2에서 4, 4에서 8로 증가하였을 경우에는 entry의 크기가 2배씩 증가

함에도 그 규모와 상관관계가 추정될 정도의 전반적인 성능 향상을 보이지는 못하였음을 확인하였다.

일반적으로 non-blocking cache (lockup-free cache) 기반의 시스템에서는 요청된 데이터에 대해서 현재 캐시 계층에서의 해당 데이터가 존재하지 않을 경우 (cache miss), MSHR의 엔트리에 해당 메모리 어드레스에 대한 접근 정보를 저장하는 방식을 취한다.

따라서, 프로그램 흐름 상 데이터 의존성을 해치지 않는 메모리 명령어들의 경우, 이전 메모리 접근의 cache miss 및 hit 여부와 관계없이, 동시에 추가적인 캐시 데이터를 현재 레이어에 요청할 수 있다. 이를 적극적으로 활용할 수 있는 방식을 Memory Level Parallelism (MLP)라고 하며, 명령어 수준의 병렬성 (Instruction Level Parallelism; ILP)와 함께 현대 프로세서의 성능적인 향상의 기반이 되는 기법 중의 하나이다. 하지만, PIM의 경우 구조 단순화를 위하여 L1 단일 캐시 계층 적용 및 MSHR 엔트리의 크기를 1로 설정하는 방식으로 제약하였으며, 제시한 PIM구조는 blocking-cache (캐시 miss 발생시, 추가적인 캐시 접근을 중단함)를 차용한 시스템이라고 볼 수 있다.

해당 성능 분석 파트에서는, MSHR의 크기를 증대 (lockup-free cache 구조 적용)하는 것을 통해서, PIM에서 주로 구동되는 PolyBench 워크로드들과 같은 연산 커널들이 성능상의 이점을 확보할 수 있는지 파악하고자 하였다. 하지만, 일정 수준 이상의 MSHR entry 확보를 넘어서면, 해당 구조의 채택에 따른 구조의 복잡성 및 비용 증대를 넘어서는 이점이 확인되지 않았으며, 이는 앞선 파트에서도 고찰한 워크로드의 메모리 접근 특성에 기인한 것으로 여겨진다.

4. 결 론

본 연구는 PolyBench에 속한 다양한 워크로드들의 성능 측정을 통해, 최근 주목받고 있는 프로세싱 인 메모리 기반 시스템 상에서 최적화 및 데이터 관리 기법 상에서 어떠한 요소들이 추가적으로 고려되어야 하는지 고찰하고자 하였다. 가속기 및 PIM에서 주로 구동되는 연산 커널들에 대한 특성 분석을 통해, 해당 커널들의 데이터 지역성과 캐시 설정에 대한 민감도 등을 파악할 수 있었으며, 이를 통해 향후 PIM 연구에서 메모리 지연 시간 감축 및 전반적인 시스템 성능 향상을 위하여 어떤 부분들을 고려해야 하는지 확인할 수 있었다.

해당 연구에서 확인한 대다수의 연산 커널들은 한 번 사용한 데이터를 근미래에 사용하지만, 장기적으로는 재사용하지 않는 특성을 보임을 확인하였다. 하지만, conv2 및 gemver 등과 같은 연산 커널들의 경우 이런 경향성과 다른 특성을 보임을 확인하였으며, 해당 커널들과 유사한 타 연산 커널들의 구동 상에서의 성능적 이슈를 해결하기 위해서는 기존의 메모리 관련 연구들에서 적용되었던 여러 관리 기법들이 필요함을 확인하였다.

메모리 사용 패턴에 따른 관리 기법 [25], 적응적 캐시 크기 증대 및 하이브리드 메모리 기법을 비롯하여 [26], 파티션 별 공유 캐시 구조 채택, 시간적 지역성 및 적합성을 고려한 프리페칭 등 다양한 측면에서의 시스템 관리 기법을 적용하는 것이 대안임을 확인하였다.

이에, 향후 연구에서는 본 분석 결과 및 고찰 내용을 바탕으로, 워크로드 특성에 적응적인 프로세싱 인 메모리 시스템에서의 캐시 블록 선택 기법 및 공유 파티션 캐시 등에 대한 기법의 효용성 등을 고찰하고자 한다. 또한, 고성능 고대역폭을 기반으로 데이터 집약적 연산의 가속을 가능하게 하는 프로세싱 인 메모리 장치를 활용함으로써, 모바일 장치 및 드론 [27] 등 다양한 최종 사용자 단말 기기에서의 딥러닝 가속 등에 있어서의 효용성과 성능 상의 이점을 검증하고자 한다.

이를 통해, 메모리 병목 현상으로 인하여 성능 상의 개선에 한계가 있었던 기존 전통적 메모리 계층 구조 기반의 시스템의 한계를 돌파하고자 한다.

감사의 글

This research was supported by National Research Foundation of Korea (NRF) Grant funded by the Korean Government (Ministry of Education) NRF-2021R1I1A1A01059737 and the MSIT (Ministry of Science and ICT), Korea, under the Innovative Human Resource Development for Local Intellectualization support program (IITP-2023-RS-2022-00156389) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

참고문헌

1. Ghose, S., Boroumand, A., Kim, J. S., Gómez-Luna, J., and Mutlu, O., “Processing-in-memory: A workload-driven perspective,” IBM Journal of Research and Development, Vol. 63(6), pp. 3-1, 2019.
2. Qureshi, M., “With new memories come new challenges,” IEEE Micro, Vol. 39(1), pp. 52-53, 2019.
3. Mutlu, O., Ghose, S., Gómez-Luna, J., and Ausavarungnirun, R., “Processing data where it makes sense: Enabling in-memory computation,” Microprocessors and Microsystems, Vol. 67, pp. 28-41, 2019.

4. Singh, G., et al., “Near-memory computing: Past, present, and future,” Microprocessors and Microsystems, Vol. 71, pp. 102868, 2019.
5. Pouchet, LN., “Polybench: The polyhedral benchmark suite,” Retrieved from: https://web.cs.ucla.edu/~pouchet/software/polybench/
6. Grauer-Gray, S., Xu, L., Searles, R., Ayalasomayajula, S., and Cavazos, J., “Auto-tuning a high-level language targeted to GPU codes,” In 2012 innovative parallel computing (InPar), IEEE, pp. 1-10, 2012.
7. Yuki, T., “Understanding polybench/c 3.2 kernels,” International workshop on polyhedral compilation techniques (IMPACT), 2014.
8. Wei, Y., Zhou, M., Liu, S., Seemakhupt, K., Rosing, T., and Khan, S., “PIMProf: an automated program profiler for processing-in-memory offloading decisions,” In 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, pp. 855-860, 2022.
9. Ghiasi, N. M., et al., “ALP: Alleviating CPU-Memory Data Movement Overheads in Memory-Centric Systems,” IEEE Transactions on Emerging Topics in Computing, 2022.
10. Abella-González, M. Á., Carollo-Fernández, P., Pouchet, L. N., Rastello, F., and Rodríguez, G., “PolyBench/Python: benchmarking Python environments with polyhedral optimizations,” In Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction, pp. 59-70, 2021.
11. Henning, J. L., “SPEC CPU2006 benchmark descriptions,” ACM SIGARCH Computer Architecture News, Vol. 34(4), pp. 1-17, 2006.
12. Bucek, J., Lange, K. D., and v. Kistowski, J., “SPEC CPU2017: Next-generation compute benchmark,” In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, pp. 41-42, 2018.
13. SPEC CPU® 2017, Retrieved from: https://www.spec.org/cpu2017/
14. PolyBench/ACC, Retrieved from: https://cavazos-lab.github.io/PolyBench-ACC/
15. Luk, C. K., et al., “Pin: building customized program analysis tools with dynamic instrumentation,” Acm sigplan notices, Vol. 40(6), pp. 190-200, 2005.
16. Intel® Pin. Retrieved from: https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html
17. Ke, Liu, et al. “Near-memory processing in action: Accelerating personalized recommendation with axdimm,” IEEE Micro, Vol. 42(1), pp. 116-127, 2021.
18. Gómez-Luna, J., El Hajj, I., Fernandez, I., Giannoula, C., Oliveira, G. F., and Mutlu, O., “Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system,” IEEE Access, Vol. 10, pp. 52565-52608, 2022.
19. UPMEM. Retrieved from: https://www.upmem.com/
20. Hassan, M., Park, C. H., and Black-Schaffer, D., “A reusable characterization of the memory system behavior of spec2017 and spec2006,” ACM Transactions on Architecture and Code Optimization (TACO), Vol. 18(2), pp. 1-20, 2021.
21. Gober, N., et al., “The championship simulator: Architectural simulation for education and competition,” arXiv preprint arXiv:2210.14324, 2022.
22. ChampSim. Retrieved from: https://github.com/ChampSim/ChampSim
23. Kim, Y., Yang, W., and Mutlu, O., “Ramulator: A fast and extensible DRAM simulator,” IEEE Computer architecture letters, Vol. 15(1), pp. 45-49, 2015.
24. Chatterjee, N., et al., “Usimm: the utah simulated memory module,” University of Utah, Tech. Rep, pp. 1-24, 2012.
25. Choi, J. H., “Lifetime Extension Method for Non-Volatile Memory based Deep Learning System by analyzing Data Write Pattern,” Journal of the Semiconductor & Display Technology, Vol. 21(3), pp. 1-6, 2022.
26. Yoon, S. K., and Nah, J. E., “Hybrid Memory Adaptor for OpenStack Swift Object Storage,” Journal of the Semiconductor & Display Technology, Vol. 19(3), pp. 61-67, 2020.
27. Park, S. H., and Park, C. S., “Implementation of GPU Acceleration of Object Detection Application with Drone Video,” Journal of the Semiconductor & Display Technology, Vol. 20(3), pp. 117-119, 2021
28. Pawlowski, J. T., “Hybrid memory cube (HMC),” In 2011 IEEE Hot chips 23 symposium (HCS), pp. 1-24, IEEE, 2011.
29. Yu, C., Liu, S., and Khan, S., “Multipim: A detailed and configurable multi-stack processing-in-memory simulator,” IEEE Computer Architecture Letters, Vol. 20(1), pp. 54-57, 2021.
30. Gao, M., Ayers, G., and Kozyrakis, C., “Practical near-data processing for in-memory analytics frameworks,” In 2015 International Conference on Parallel Architecture and Compilation (PACT), pp. 113-124, IEEE, 2015.
31. Min, C., Mao, J., Li, H., and Chen, Y., “NeuralHMC: An efficient HMC-based accelerator for deep neural networks,” In Proceedings of the 24th Asia and South Pacific Design Automation Conference, pp. 394-399, 2019.

접수일: 2023년 9월 10일, 심사일: 2023년 9월 14일, 게재확정일: 2023년 9월 18일