UPMEM PIM의 HPC 분야 적용 가능성 연구

곽재혁[1]
[1]한국과학기술정보연구원
jhkwak@kisti.re.kr

A Study on the applicability of UPMEM PIM to HPC

Jae-Hyuck Kwak[1]
[1]Korea Institute of Science and Technology Information

요 약

PIM은 CPU와 메모리 간의 데이터 버스 오버헤드를 완화하기 위해서 메모리 내부에 프로세서를 가지며 낮은 데이터 재사용성을 가지는 데이터 집약형 워크로드에서 지연과 에너지 관점에서 장점을 가진다. 본 논문은 UPMEM사의 PIM을 이용하여 HPC분야에서 자주 사용되는 행렬 연산인 GEMV, SpMV의 벤치마크 구현을 분석하고 성능 분석을 통해 CPU 대비 가지는 장단점에 대해서 논하였다.

1. 서론

CPU와 메모리 간에 많은 데이터 이동을 요구하는 데이터 집약형 워크로드의 경우 지연과 에너지 관점에서 심각한 오버헤드를 야기하며 특히, 낮은 데이터 재사용성은 메모리 접근의 비용을 상쇄하기에 충분하지 않다. PIM은 CPU와 메모리 간의 데이터 버스 오버헤드를 완화하기 위해서 메모리 내부에 프로세서를 가진다.

UPMEM사의 PIM[1]은 DPU라고 하는 프로세서를 메모리 칩에 추가하여 계산을 수행하며 CPU와 메모리 사이의 데이터 이동을 감소시킨다. 따라서 에너지 관점에서 효율적이며 메모리 내부 프로세서는 더 높은 대역폭을 사용할 수 있기에 계산을 가속화하여 더 나은 성능을 도출할 수 있다.[2]

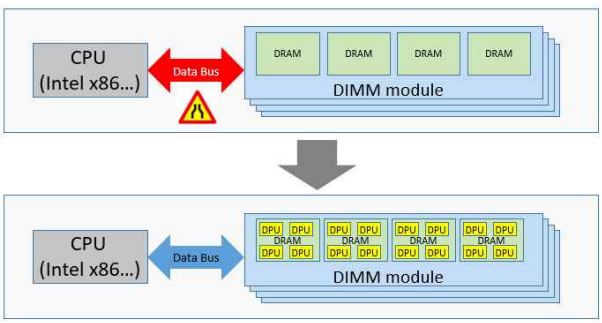


그림 1 레거시 메모리와 PIM 비교

본 논문에서는 UPMEM PIM을 이용하여 HPC분야에서 자주 사용되는 행렬 연산인 GEMV, SpMV의 벤치마크 구현을 분석하였고 CPU 대비 가지는 장단점에 대해서 논하였다.

2. UPMEM PIM 아키텍처

UPMEM PIM 모듈은 랭크당 8개의 PIM칩을 가지며 PIM칩은 8개의 DPU와 DPU별로 64MB의 메모리 뱅크(MRAM)를 가진다. DPU는 독자적인 ISA를 가진 멀티쓰레드 지원 인오더 32비트 RISC코어로서 인스트럭션을 위한 IRAM이 있고 64KB의 WRAM은 스크래치 패드나 MRAM을 위한 캐시로 사용된다. 파이프라인은 IRAM으로부터 인스트럭션을 읽고 WRAM을 load나 store를 통한 데이터 저장 공간으로 사용하며 DMA는 MRAM과 WRAM간의 데이터 이동에 사용된다. 파이프라인은 24개의 쓰레드로 구성되고 인터리브 실행되며 매 사이클마다 최대 하나의 인스트럭션을 실행한다.

UPMEM SDK[3]는 범용적인 DPU 프로그래밍을 위한 API 라이브러리와 컴파일러, 디버거 등을 제공한다. DPU의 제한된 구조로 인해 DPU 프로그래밍에 제한이 있으며 이를 잘 이해하면서 모든 DPU를 사용하도록 병렬화하는 것이 중요하다.

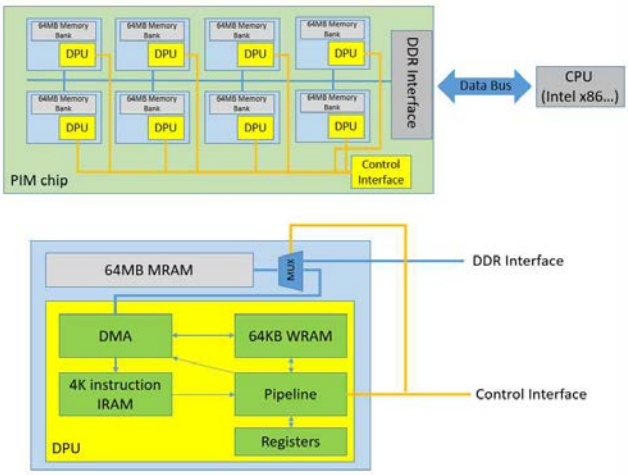


그림 2 UPMEM PIM 아키텍처

3. UPMEM PIM 기반 GEMV, SpMV 벤치마크 구현 분석

GEMV (Matrix-Vector multiply)는 크기가 m x n인 행렬과 크기가 n x 1인 벡터를 입력으로 사용하고 이들 사이의 곱셈을 수행하여 크기가 m x 1인 벡터를 생성하는 밀집 선형 대수 연산이다. SpMV (Sparse Matrix-Vector multiply)는 희소 행렬과 밀집 벡터의 곱셈을 수행하는 희소 선형 대수 연산이다. 두가지 모두 HPC 응용에서 자주 사용되는 연산으로 여기서는 PrIM 벤치마크[4]에 구현된 GEMV, SpMV 구현을 기반으로 분석하였다.

전체적인 과정은 크게 (1) CPU에서 DPU로 입력 데이터 분배, (2) DPU에서 병렬 계산, (3) DPU에서 CPU로 출력데이터 취합 단계로 나뉜다. DPU프로그램은 파이프라인 구조로 인해 멀티쓰레드를 사용하며 UPMEM SDK는 태스크릿(tasklet)을 통해서 멀티쓰레드를 지원한다. 따라서 DPU에 분배된 입력데이터는 다시 태스크릿으로 분배되어 병렬적으로 계산된다.

호스트 프로그램에서 DPU 프로그램으로 입력데이터 전송을 효율화하기 위해서 dpu_prepare_xfer, dpu_push_xfer를 사용한 랭크 전송 인터페이스를 사용한다. 이때 타겟이 되는 DPU메모리에 따른 정렬(align) 제한이 있는데 WRAM 주소와 길이는 4바이트로 정렬되어야 하고 MRAM 주소와 길이는 8바이트로 정렬되어야 한다.

GEMV의 경우 데이터 타입은 uint32_t형이며 SpMV는 float형을 사용한다. UPMEM PIM의 경우 실수형은 소프트웨어적으로만 지원하기 때문에 float형을 사용하는 SpMV는 이로 인한 성능 저하가 발생할 수 있음을 예상할 수 있다.

4. PIM서버 구축 및 성능 분석

성능 분석을 위한 서버는 UPMEM 바이오스 업데이트가 완료된 Intel Server System R2312WFTZSR을 사용하였다. 프로세서는 Xeon Silver 4215R 3.2GHz (8 Cores, 16 Threads) 2개가 장착되어 있다. 메인 메모리는 64GB DDR4 DIMM을 소켓당 2개씩 1개의 메모리 채널을 공유하도록 장착하여 총 256GB용량을 가지며 UPMEM PIM은 8GB DDR4-2400 PIM DIMM으로 소켓당 10개씩 총 160GB의 용량을 가진다. 운영체제는 SDK의 DPU 프로파일링 도구의 일부 기능이 데비안에서만 지원하여 Debian 10.12 (Buster)를 사용하였다.

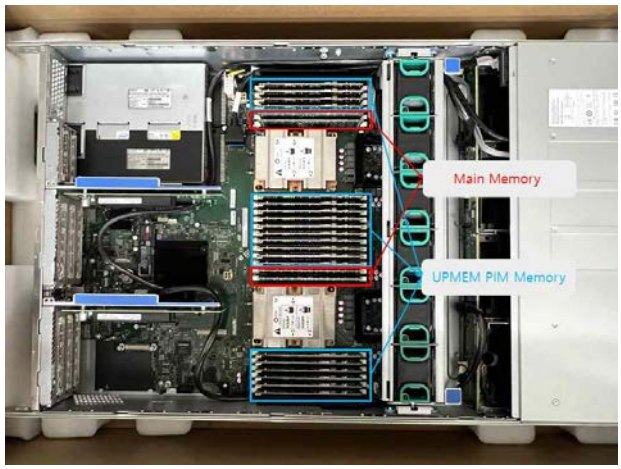


그림 3 PIM 서버 구축

그래프를 보면 GEMV, SpMV 벤치마크에 대해서 DPU 개수에 따라 싱글 랭크, 멀티 랭크에서의 강확장성(Strong Scalability)과 싱글 랭크에서의 약확장성(Weak Scalability)를 비교하였으며 태스크릿 개수는 16을 사용하였다. 주황색 가로선은 CPU에서의 실행 결과값이다.

GEMV의 그래프를 보면 강확장성인 경우에 싱글 랭크는 DPU 실행 시간이 DPU갯수에 따라 감소함을 알 수 있고 멀티랭크는 CPU 실행 시간보다 크게 우세함을 알 수 있다. 약확장성인 경우는 DPU갯수가 증가함에 따라 DPU 실행 시간이 일정하게 유지됨을 확인할 수 있다.

SpMV의 그래프를 보면 강확장성인 경우에 싱글 랭크와 멀티랭크 모두 DPU 실행 시간이 CPU 실행 시간과 비교하여 크게 우세하지 못함을 알 수 있다. 특히, CPU에서 DPU로 입력 데이터 전송 시간이 크게 발생하는데 이것은 DPU가 정수형과 달리 부동소수점을 소프트웨어적으로 에뮬레이션하여 처리하기 때문에 발생하는 비용으로 볼 수 있다. 약확장성

인 경우는 DPU갯수가 증가함에 따라 DPU 실행 시간이 일정하게 유지됨을 확인할 수 있다.

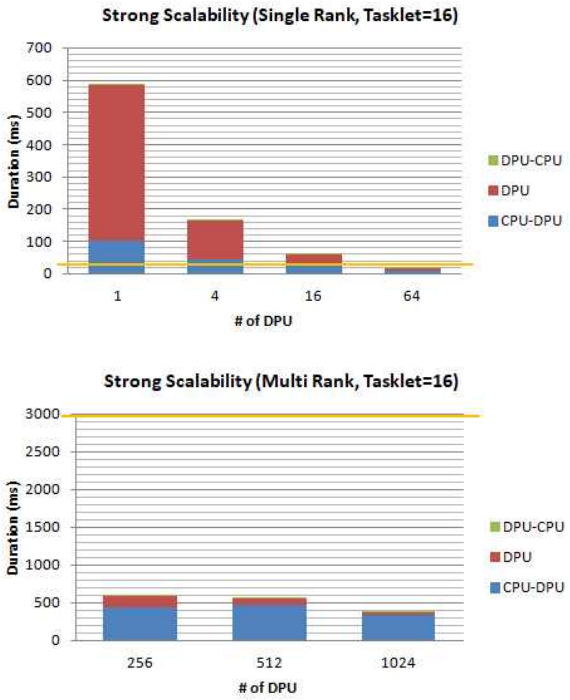
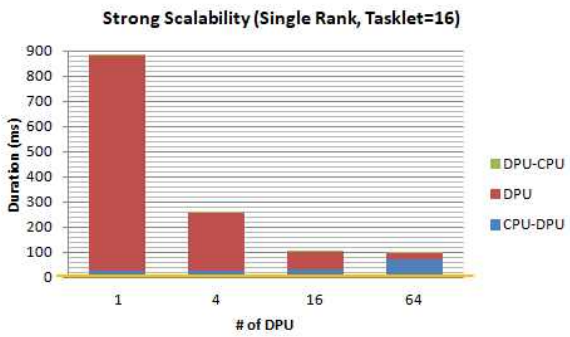


그림 4 GEMV 성능 분석



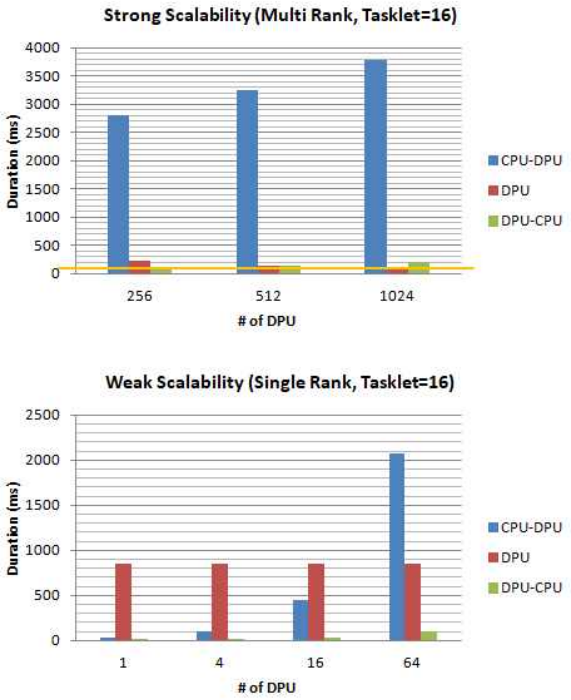


그림 5 SpMV 성능 분석

5. 결론 및 향후계획

UPMEM PIM의 경우 DPU 프로그래밍을 통해 범용적인 PIM 연산을 제공하지만 아키텍처 한계로 인해서 정수형이 아닌 실수형 데이터 타입에 대해서는 성능 한계를 가짐을 확인할 수 있었다. 따라서 UPMEM PIM의 경우 타겟 응용 별로 성능 차이가 크게 나타날 수 있으며 타겟 응용의 데이터 타입과 특성을 면밀하게 분석하여 적용할 필요가 있다.

※ 본 연구는 2022년도 한국과학기술정보연구원(KISTI) 주요사업 과제(K-22-L02-C06-S01, 초고성능컴퓨팅 공동활용을 위한 통합 환경 개발 및 구축)로 수행한 결과임

참고문헌

[1] Juan Gómez-Luna, et al., "Benchmarking a new paradigm: experimental analysis of a real processing-in-memory architecture.", https://doi.org/10.48550/arXiv.2105.03814, 2022.

[2] Joel Nider, et al. "A Case Study of Processing-in-Memory in off-the-Shelf Systems", 2021 USENIX Annual Technical Conference, 2021.

[3] UPMEM SDK, https://sdk.upmem.com/2021.3.0/

[4] PrIM Benchmark, https://github.com/CMU-SAFARI/prim-benchmarks