한양대학교

SW프로젝트 제안서

프로젝트명	Processing-in-Memory 활용 기술 개발
프로젝트 요약	Processing-in-Memory의 개념을 이해하고, UPMEM-PIM의 구조 및 UPMEM-PIM SDK를 분석하여 PIM을 이용한 개발 방법을 파악한다. 이를 바탕으로 PIM을 이용하여 sort-merge join 알고리즘의 성능 향상 방법을 고안 및 구현하고, PIM의 유무에 따른 성능을 비교한다.
예상 결과물	SW (○), HW ( ), 특허 ( ), 논문 (○), 프로그램등록 ( )
지도교수	강수용
예상기간	2024.03.04. ~ 2024.10.31.

전공	학번	학년	이름	연락처
컴퓨터소프트웨어학부	2021019961	4	장서연	tjdus092222@hanyang.ac.kr 010-2658-4217
컴퓨터소프트웨어학부	2021097356	4	김한결	gyeol0719@hanyang.ac.kr 010-3136-7140

• 연락처는 이메일과 전화번호를 모두 정확히 기재하고 반드시 수신 가능한 것으로 기입 해야함.
(연락을 받지 못해 불이익을 당할 수 있음)

한양대학교

목 차

1. 프로젝트 배경 및 목표

2. 프로젝트 주요 내용

3. 추진 계획

4. 결론

5. 참고 문헌

1. 프로젝트 배경 및 목표

본 프로젝트에서는 Processing-in-Memory(PIM)를 활용하여 sort-merge join 알고리즘의 성능을 향상하고, PIM의 유무에 따른 성능의 차이를 알아본다.

프로젝트 진행을 위해 데이터베이스(databases), 컴퓨터 구조에 대한 이해가 필요하다. 대용량의 데이터를 처리하는 과정에서 기존의 폰 노이만 구조로는 제한된 대역폭(bandwith)와 에너지 부족, 시스템 지연 등의 여러 한계점이 존재하며, 병목현상(bottleneck)이 발생한다. 이를 해결하고자, CPU와 DDR4 DRAM memory를 DRAM memory 다이에 통합시킨 PIM이 등장했다. PIM은 데이터가 있는 곳에서 연산을 수행할 수 있으며 CPU와 DRAM 사이에서는 데이터 자체가 아닌 프로그래밍 명령과 결과만이 이동한다.

데이터베이스 분야에서 활용되는 join 알고리즘 중 하나인 sort-merge join 알고리즘 또한 대규모 데이터 이동과 상당한 연산이 요구되므로 기존의 컴퓨터 구조로는 병목현상이 존재할 수 있다. 따라서 PIM 활용이 적합한 케이스이며, 성능을 향상할 수 있도록 PIM을 활용한 HW 설계를 고려해 프로그래밍할 필요성이 있다. 또한, 데이터 집약형(data-intensive) application의 병목현상에 대한 해결이 요구되므로 PIM을 활용한 연구, 개발은 대용량 데이터를 다루는 분야의 산업적, 학술적 발전에 이바지할 수 있을 것이다. 뿐만 아니라 PIM의 유무에 따른 성능의 차이를 비교하며 PIM의 기술적 의의를 확인할 수 있다.

본 프로젝트의 최종 목표는 UPMEM사의 PIM을 이용하여 PIM을 활용한 sort-merge join 알고리즘을 구현하고 PIM의 유무에 따른 성능을 비교하는 것이다. 이를 달성하기 위해 다음과 같은 세부 목표가 필요하다. PIM(Processing-in-Memory)의 개념을 이해하고, 본 프로젝트에서 사용할 UPMEM-PIM의 구조 및 UPMEM-PIM SDK 분석을 바탕으로 PIM을 이용한 개발 방법 파악한다. 이를 바탕으로 PIM을 이용한 sort-merge join 알고리즘의 성능을 향상할 방법 고안 및 구현하고, PIM의 유무에 따른 성능을 비교한다.

2. 프로젝트 내용

UPMEM사의 PIM을 이용하여 향상된 성능의 sort-merge join 알고리즘을 개발하고 성능을 비교하기 위해서는 다음과 같은 사항이 수반되어야 한다. UPMEM-PIM의 구조 및 특징을 파악해야 하며 이를 활용한 프로그래밍 방법인 UPMEM SDK를 숙지해야 하고, sort-merge join 알고리즘에 대한 이해가 필요하다.

2.1 Processing-In-Memory(PIM)

데이터 집약형 워크로드는 CPU와 메인 메모리 간에 많은 데이터 이동을 요구하므로, 지연과 에너지 측면에서 상당한 오버헤드를 발생시킨다. 이러한 병목 현상으로 인해 산술 연산과 메모리 접근 간의 격차는 점점 더 커지고 있으며 메모리 접근 비용 역시 비싸지고 있다. 연구 결과에 따르면, 데이터 이동이 전체 시스템 에너지의 62%(2018년), 40%(2014년), 35%(2013년)를 차지한다고 한다.

병목 현상을 완화하리라 기대되는 방법의 하나가 메모리 내부에 프로세스를 가지는 processing-in-memory(PIM)이다. 3D-stacked 메모리에 DRAM 레이어와 로직 레이어를 통합하는 processing-near-memory(PNM), SRAM, DRAM 또는 비휘발성 메모리의 메모리 셀의 아날로그 운영 특성을 활용하여 특정 유형의 작업을 효율적으로 수행하는 processing-using-memory(PUM)이 있었지만, 전자는 높은 비용, 제한된 용량, 로직 레이어로 의한 임베디드 프로세싱 컴포넌트에 제약이 존재한다는 이유로, 후자는 복잡한 연산이 어렵고 규칙적인 계산에 더 적합하다는 한계 때문에 두 방법을 기반으로 하는 완벽한 PIM 시스템은 상용화가 되지 않았다.

2.2 UPMEM-PIM

UPMEM PIM은 실제 하드웨어에서 상용화된 첫 번째 PIM이다. PNM과 PUM의 한계를 극복하기 위해, 2D DRAM 배열을 사용하고 이를 동일한 칩에 DRAM Processing Units(DPUs)와 결합한 구조를 가진다. UPMEM은 비교적 깊은 파이프라인을 가지고 멀티스레드로 처리되는 DPU 코어를 사용한다.

UPMEM PIM 구조는 다른 PIM 방식에 비해 다음과 같은 장점을 가진다. 첫째로, 2D DRAM을 사용하므로 PNM에서 발생하던 문제를 피할 수 있고, 둘째, DPU가 다양한 연산 및 자료형을 지원한다. 셋째, 스레드가 독립적으로 실행될 수 있기에 불규칙적 연산에 적합하고, 넷째, UPMEM에서 C 언어로 DPU 프로그램을 작성할 수 있게끔 정보를 제공한다.

아래 그림은 호스트 CPU와 UPMEM PIM 시스템을 보여준다. PIM을 사용하는 메모리는 N개의 UPMEM PIM 모듈로 이루어지며, 이 모듈은 랭크 당 8개(총 16개)의 PIM 칩을 가진다. PIM 칩은 8개의 DPU로 이루어지며, 각각의 DPU는 메인 메모리인 MRAM, 명령어를 위한 IRAM, 스크래치 패드나 MRAM을 위한 캐시로 사용되는 WRAM을 가진다. MRAM은 호스트 CPU에서 데이터를 복사하고(CPU-DPU) 결과를 찾는데(DPU-CPU), 이는 MRAM에서 전송되는 버퍼의 크기가 동일할 때 병렬로 일어날 수 있다. 그렇지 않다면, 하나의 MRAM에서

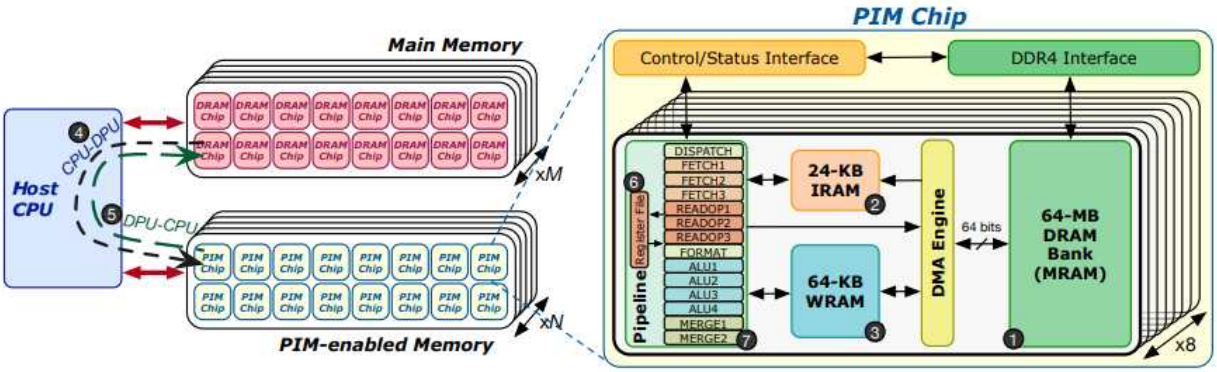

그림 1 UPMEM-based PIM system with a host CPU, standard main memory, and PIM-enabled memory (left), and internal components of a UPMEM PIM chip (right) [2].

전송이 완료된 후 다른 MRAM에서의 전송이 시작되는 식으로 순차적으로 전송이 발생한다. 모든 DPU는 호스트 CPU에 대한 병렬 보조 프로세서로 함께 작동하며 DPU 간의 직접적인 통신은 불가능하다. 모든 DPU 간의 통신은 호스트 CPU를 통해 DPU에서 CPU로 결과를 검색하고 CPU에서 DPU로 데이터를 복사하는 방식으로 이루어진다. DPU는 24개의 하드웨어 스레드를 가지며, 14단계의 파이프라인 깊이를 가지지만 마지막 세 단계만이 DISPATCH 및 FETCH 단계와 병렬로 실행될 수 있다. 따라서 파이프라인을 완전히 활용하려면 적어도 11개의 스레드가 사용해야 한다.

UPMEM PIM 시스템의 일반적인 프로그래밍 권장 사항은 다음과 같다. 첫째, 호스트 CPU와 빈번한 상호 작용을 피하고 가능한 한 긴 병렬 코드를 DPU에서 실행한다. 둘째, 워크로드를 독립적인 데이터 블록으로 분할하여 사용한다. 셋째, DPU를 최대한 많이 사용한다. 넷째, 각 DPU에서 적어도 11개의 tasklet을 사용한다.

2.3 UPMEM-PIM SDK

UPMEM사는 개발자들이 PIM 프로그래밍에 쉽게 적응할 수 있도록 Software Development Kit(SDK)와 toolkit, 디버거를 제공한다.

첫 번째로 functional library이다. 이는 tasklet을 통해 소프트웨어 추상화와 기본적인 하드웨어 스레드에 스택과 같은 시스템 기능을 수행한다. 이 기능을 기반으로 runtime libraray는 mutex, semaphore 등 다양한 동기화 기본 요소를 제공한다. 또한, WRAM 동적 관리, MRAM과 WRAM 사이의 트랜잭션을 관리하며 MRAM에 접근을 수행할 수 있고, tasklet으로 자신의 목적을 위해 작업 메모리에 버퍼를 가져오거나 공동 작업을 위해 미리 예약된 일부 공유 메모리에 접근이 가능하다.

두 번째는 communication library이다. 호스트 측에서, C API를 사용해 호스트 메모리와 DPU 메모리(IRAM, WRAM, MRAM) 중 임의의 메모리 사이에 데이터를 전송하는 기능을 제공한다.

세 번째는 LLVM 컴파일러와 LLDB 디버거이다. DPU를 타겟으로 한 컴파일러는 clang 10.0.0을 사용하는 LLVM을 기반으로 한다. PIM 아키텍처는 수천 개의 프로세서를 한 번에 디버깅해야 하므로 lldv 10.0.0을 기반으로 효율적인 디버깅 툴을 고안했다. 이는 코드 구조 유지를 위해 CPU와 DPU를 동시에 작용하고, 그룹 또는 개별 DPU 단위로 작동하여 정확한 파악이 가능하며, 성능을 고려한 낮은 수준의 프로파일링으로 성능 저하가 발생하는 특정

DPU를 파악할 수 있다.

네 번째는 functional simulator이다. toolkit은 instruction trace와 DPU 그라인드로 구성된 functional simulator를 제공한다. functional simulator에는 cycle-accurate가 없기 때문에 trace 카운터를 cycle 카운터로 해석해서는 안 된다. 따라서 성능 평가가 허용되지 않으며, 평가를 위해서는 클라우드 서버나 평가 서버가 필요하다.

마지막은 PIM driver이다. linux driver를 사용하면 DRAM에 데이터를 읽고 쓸 수 있으며 DRAM 내장 처리 장치와 통신이 가능하다.

2.4 sort-merge join algorithm

데이터베이스에서, join은 두 개 이상의 테이블을 하나의 집합으로 만드는 연산이다. sort-merge join 알고리즘은 두 개의 테이블에서 각각 join 대상을 먼저 읽은 후 정렬하고 merge하여 join을 수행한다. 각 테이블에 대해 동시에 독립적으로 데이터를 먼저 읽어 들이고 읽힌 각 테이블의 데이터를 join column 기준으로 정렬한 후, join 작업을 수행한다.

NL Join을 수행할 경우에는 random access 방식으로 데이터를 읽기 때문에 넓은 범위의 데이터를 처리할 때 부담이 된다. 반면 sort-merge join은 PGA에서 수행되어 random access 부하가 없고, full table scan 방식으로 넓은 범위의 데이터를 처리할 때 유용하다. 일반적으로 대량의 join 작업에서 정렬 작업을 요구하는 sort-merge join보다 CPU 작업 위주로 처리하는 hash join이 성능상 유리하지만, join 조건으로 equal 연산자를 사용하는 Equi Join에서만 사용 가능한 hash join과 달리 Non-Equi Join에 대해서도 작업이 가능하다. 또한 join column의 인덱스가 존재하지 않을 경우에도 사용할 수 있다는 장점이 있다. 그러나 정렬할 데이터가 많아 메모리에서 모든 정렬 작업을 수행하기 어려운 경우에는 임시 영역으로 디스크를 사용하기 때문에 성능이 저하될 수 있다.

따라서 sort-merge join 알고리즘의 성능 개선점은 다음과 같다. 데이터가 대량일 때 정렬 단계에서 더 효율적으로 메모리를 관리하며, 테이블에 접근하는 속도와 정렬 속도를 향상하고, 두 테이블 중 어느 한쪽이라도 정렬 작업이 종료되지 않으면 한쪽이 대기 상태가 되고 다른 한쪽의 정렬이 완전히 끝날 때까지 join 작업이 시작될 수 없으므로 양쪽의 정렬 완료 시점을 맞춘다.

2.5 연구개발 계획

본 프로젝트에서 활용할 UPMEM-PIM의 구조 및 특징을 파악한다. 이를 토대로 UPMEM-PIM 프로그래밍 방법을 숙지하고, UPMEM-PIM의 특징을 고려한 sort-merge join 알고리즘을 개발 및 구현한다. 구현한 알고리즘의 성능을 실험하기 위해 자체 dataset을 만들고 실제로 적용 후 결과를 분석한다. 이후 실험 결과를 바탕으로 UPMEM-PIM을 사용하지 않았을 때와 성능을 비교한다.

3. 프로젝트 추진 계획

본 프로젝트의 팀원은 컴퓨터소프트웨어학부 4학년 2021019961 장서연과 2021097356 김한결이다.

별도의 역할 분담은 없으며, 매주 회의를 통해 아이디어를 공유하고 공동으로 작업한다. 프로젝트 기간 동안 월 단위 계획은 다음과 같다.

3월	프로젝트 주제 선정 및 관련 지식 사전 조사, 제안서 작성
4월	UPMEM-PIM SDK 사용 방법 및 PIM 프로그래밍 방법 숙지
5월	PIM을 활용한 sort-merge join 알고리즘 디자인 및 설계
6월	아이디어 구현
7월	아이디어 구현
8월	실험(성능 비교) 및 결과 분석
9월	아이디어 보완 및 재실험
10월	논문 작성

4. 결론

UPMEM PIM을 활용한 sort-merge join 알고리즘이 구현될 것이며, 기존 환경에서의 성능과 PIM을 사용하였을 때의 성능을 비교하여 분석한 결과를 도출할 수 있을 것으로 예상된다.

프로젝트를 수행하며 low-level 관점에서 HW를 고려한 프로그래밍 능력이 향상될 것으로 기대된다. 또한 PIM에 대한 깊은 이해와 이를 활용한 프로그래밍 역량을 쌓을 수 있을 것이다. 알고리즘의 특징과 성능 개선점을 찾아 새로운 아이디어를 고안하고 구현하는 과정을 통해서 설계 및 구현 능력을 발전시킬 수 있다.

산업적 기대효과로는 PIM을 활용한 sort-merge join 알고리즘을 통해 PIM이 데이터베이스 분야에서 사용되는 효율을 증진할 수 있으리라 기대된다. 또한 학술적인 관점에서, PIM의 활용 여부에 따른 성능을 비교하며 PIM의 기술적 의의를 제시할 수 있을 것이다.

5. 참고 문헌

[1] UPMEM, “UPMEM Processing In-Memory (PIM): Ultra-efficient acceleration for data-intensive applications”, UPMEM, 2022.

[2] 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.

[3] 곽재혁, “UPMEM PIM의 HPC 분야 적용 가능성 연구”, 한국정보처리학회, 2022.

[4] Gavin JT Powell, “Oracle Data Warehouse Tuning for 10g”, Elsevier Sciencd, 2011.

[5] UPMEM SDK, https://sdk.upmem.com/2023.2.0/