[졸업 프로젝트]

Processing-in-Memory 활용 기술 개발

- 월별 보고서 -

한양대학교 컴퓨터소프트웨어학부
2021019961 장서연 2021097356 김한결

2024.06.20.

목차

1. 6월 계획

- UPMEM-PIM을 활용한 sort-merge join 알고리즘 구현

2. 진행 사항

5월에는 UPMEM-PIM을 활용한 sort-merge join 알고리즘 디자인을 목표로 프로젝트를 진행하였다.

2.1. sort-merge join 알고리즘 디자인 및 구현

PIM을 활용한 sort-merge join 알고리즘을 디자인하기 이전에, 성능 개선점을 찾기 위해 일반적인 sort-merge join 알고리즘을 구현하였다. 임의로 생성한 test data(table1.csv, table2.csv)와 결과 data(joined_table.csv)와 구현 코드(sort_merge_join.c)와 실행 파일(test)은 zip 파일로 첨부하였다.

C언어로 sort-merge-join을 구현했다. ID, 이름, 나이로 구성된 table(table1)과 ID, 연봉, 부서로 구성된 table(table2)을 대상으로 적용했으며, join column은 ID이다. 해당 데이터는 임의로 제작되었으며, 추후에 구조가 변경될 수 있다. table1에서는 나이를 조건으로 select를 수행 후(`select_table1`) ID를 기준으로 정렬(`sort_table1`)하였고, table2에서는 연봉을 조건으로 select를 수행 후(`select_table2`) ID를 기준으로 정렬(`sort_table2`)했다. 이후 두 개의 table을 비교하여 join(`join`)했다.

해당 코드에서는 selection sort를 정렬 알고리즘으로 사용했으나, 추후 다른 정렬 알고리즘을 사용할 수 있다. (본 프로젝트에서는 정렬 방식보다 PIM을 활용한 코드와 그렇지 않은 코드의 정렬 방식이 동일한 것을 바탕으로 비교하는 것이 중점이라고 생각해 중요한 부분이 아니라고 판단하였다.)

2.2. sort-merge join 알고리즘의 성능 개선점

알고리즘을 직접 구현해보니 크게 두 가지 과정에서의 성능 개선이 필요하다고 판단했다.

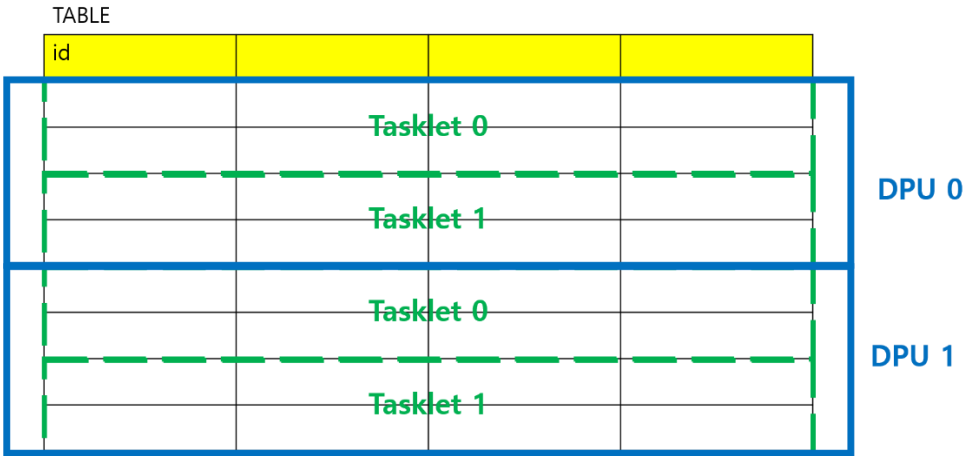
첫 번째는 방대한 데이터가 쓰일 때의 메모리 접근이다. 위의 코드에서는 3개의 column으로 구성된 100개의 dataset 2개에 접근했지만, 실제 사용에서는 더 많은 column과 row로 구성된 방대한 데이터를 쓰기 때문에 메모리 액세스에서 cost가 크게 발생한다.

두 번째는 정렬과 merge 과정이다. PIM을 활용하지 않고 단일 CPU에서 많은 수의 데이터를 대상으로 정렬을 수행할 때 오버헤드가 발생한다. 마찬가지로 두 개의 테이블을 비교할 때에도 많은 cost가 들기 때문에 보다 효율적인 정렬 방식의 구현과 join 과정이 필요하다.

2.3. UPMEM-PIM을 활용한 sort-merge join 알고리즘 디자인

2.2에서의 논의를 바탕으로, 언급한 개선점에서 UPMEM-PIM을 활용해 sort-merge join 알고리즘을 디자인했다.

sort-merge join은 join column을 정렬할 때와 정렬된 columns를 merge할 때 두 번의 메모리 접근이 필요하다. 따라서 다음 그림과 같이 전체 데이터베이스를 사용할 개수의 DPU와 각 DPU에서의 tasklet 개수로 나누어 병렬적으로 접근한다.



각 DPU의 각 tasklet은 위 그림처럼 나누어진 일부의 데이터에 접근한다. 이를 통해 PIM을 활용하지 않은 기존의 구조에서 전체 데이터를 읽어오는 것보다 빠르게 가져올 수 있다.

각 tasklet에서 조건을 만족하는 row를 찾고 join column을 기준으로 정렬을 수행한 후, divide-and-conquer 방식으로 두 개의 DPU에서 정렬한 결과를 하나의 table로 만들고, 이 과정을 반복한다. (예: DPU0의 결과와 DPU1의 결과를 비교하여 재정렬하고 DPU2의 결과와 DPU3의 결과를 재정렬 후, 전자의 결과와 후자의 결과를 다시 재정렬한다.)

여기까지의 과정을 거치면 하나의 테이블에 대해 select와 sort의 과정을 수행한 결과를 얻을 수 있다. 동시에, 병렬적으로 다른 테이블에 대해서도 동일한 과정을 수행하여 결과를 얻는다.

이후 join 과정에 대해서는 실험이 필요할 것으로 보인다. 구상한 첫 번째 방식은 위와 같이 정렬된 table을 다시 쪼개서 접근하고 각각의 tasklet이 가지고 있는 데이터를 비교하는 것이다. 그러나 이 방식은 각 tasklet이 가지고 있는 각 table의 join column에 어떤 값이 들어있는가에 따라서 오히려 동일한 join column의 값을 찾기까지 긴 시간이 걸릴 수 있으므로 비효율적일 수 있다. 두 번째 방식은 전체 data를 sequential하게 읽으며

join하는 것이다. 2.1에서 구현한 코드와 같은 join 방식이다.

2.4. 추가 논의

설계와 별개의 추가적인 논의사항이다. 첫 번째는 input 데이터의 구조이다. 구현 과정에서 input으로 들어오는 데이터의 구조를 정해두고 코드를 작성해야 하는지(2.1에서의 구현과 같은 방식), 아니면 어떤 구조의 데이터에도 동작하는 general한 코드를 작성해야 하는지에 대한 논의이다. 두 번째는 성능 측정을 위한 테스트 데이터 생성 방식이다. 구현한 알고리즘의 성능을 측정하기 위해 사용하는 데이터를 생성할 때, 어떻게 생성해야 편향되지 않고 측정에서의 객관성을 확보할 수 있는가에 대한 논의이다.

3. 6월 계획

- 2.4에 언급한 논의 사항 마무리
- 구현 시작