[졸업 프로젝트]

Processing-in-Memory 활용 기술 개발

- 월별 보고서 -

한양대학교 컴퓨터소프트웨어학부
2021019961 장서연 2021097356 김한결

2024.08.20.

목차

1. 8월 계획

- UPMEM-PIM을 활용한 sort-merge join 알고리즘 구현 완료

2. 진행 사항

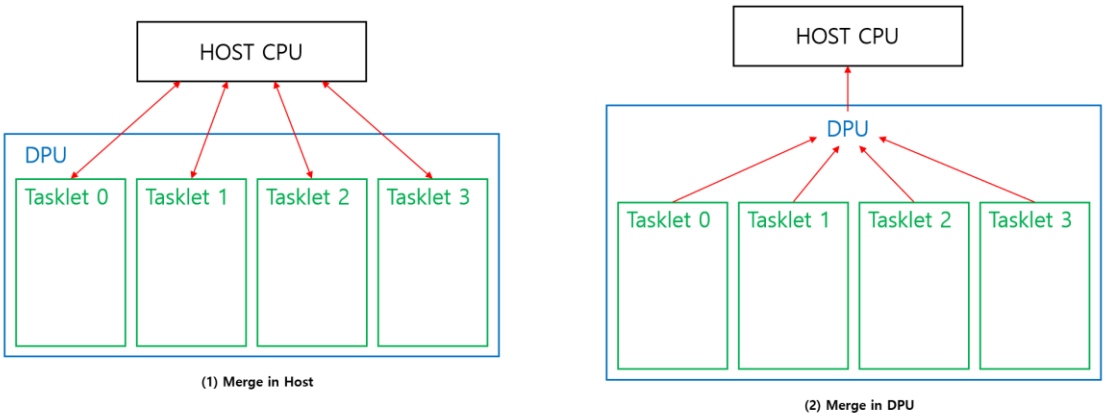
8월 계획의 일부를 달성하였다.

2.1. 구조 변경 및 구현

tasklet 별 결과를 merge하는 장소와 이에 따른 DPU 별 산출된 결과를 merge하는 방식에 대해 논의, 변경하였다. 해당 코드는 github repository에서 확인할 수 있다.

2.1.1. tasklet 별 결과 merge 방식

테이블을 dpu의 개수만큼 나누어 할당하고, 할당된 데이터를 tasklet 개수만큼 나누어 할당하였으며 그 tasklet 안에서 select 및 sort를 진행하였다. 그 후 이 결과물을 어떻게 합칠 것인지에 대해 두 가지 방안을 검토해 보았다. 첫 번째는 host가 모든 결과를 관리하는 방법이다. 이 방법에서 host는 tasklet의 결과를 받아 tasklet 0과 1을 합치고 tasklet 2와 3을 합치는 등의 행동을 자체적으로 주관하게 된다. 두 번째는 dpu에서 먼저 취합한 후, 그 결과물을 다시 host로 보내 host에서 최종적으로 합치는 방법이다. 이 경우 host는 전자보다 할 일이 줄어들며 dpu 개수만큼의 정렬된 데이터를 받아 이 데이터를 merge하는 데에만 신경 쓸 수 있다. 전자의 경우, host와 dpu 간의 전송이 불필요하게 많아지고 구조가 복잡해지는 등의 문제가 있었기에 순차적으로 merge하는 후자를 선택하여 구현하였다.

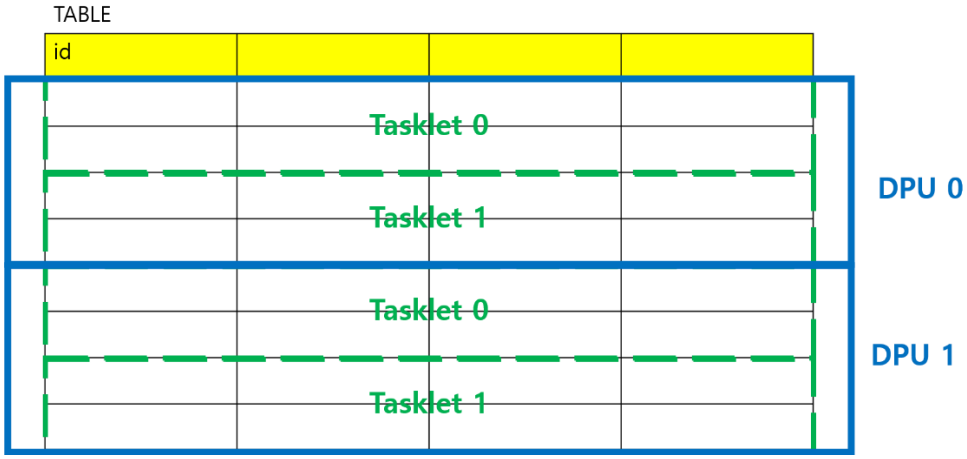


tasklet 결과 merge

우선 host에서 전달한 데이터를 받아 각 tasklet에 할당한다. 다음으로 select를 진행하

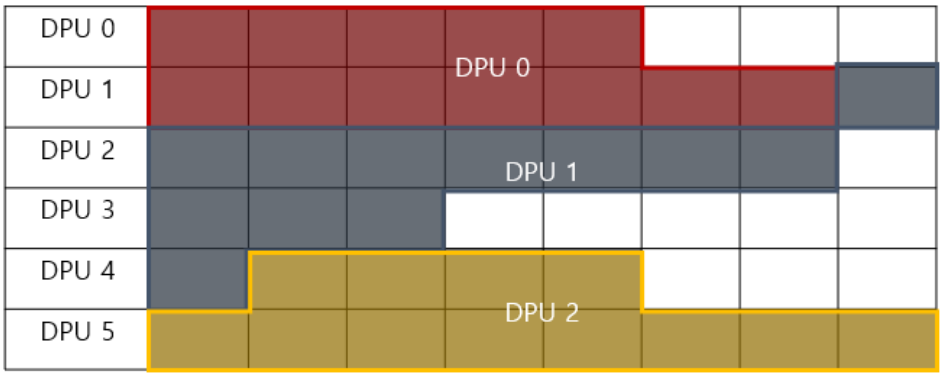
며, select의 조건은 매크로 상수인 JOIN_COL번째의 요소가 JOIN_VAL보다 큰지를 따진다. 선별된 행은 quick sort를 통해 오름차순으로 정렬된다. 이 결과는 result 배열의 tasklet_id번째에 저장되며, 행의 개수 역시 같이 저장된다. merge는 while문을 통해 진행되며 첫 번째 tasklet은 result[0]과 result[result_size-1]에 저장된 데이터를, 두 번째 tasklet은 result[1]과 result[result_size-2]에 저장된 데이터를 합치는 식으로 작동한다. 따라서 result_size 절반의 tasklet만이 사용되며 이는 result_size가 1이 될 때까지 반복된다. 최종적으로 result[0]에 결과물이 저장되고 host로 전달된다.

2.1.2. DPU 별 결과 merge 방식



기존의 방식: DPU를 단위로 merge (5월 보고서에 첨부된 자료)

기존의 방식은 DPU i와 DPU i+1의 결과를 DPU (i/2)에서 sort 후 merge하는 과정을 하나의 DPU에서 처리하여 하나의 sort된 table을 결과로 낼 때까지 반복하는 방식이었다.



수정된 방식: 각 DPU 별 row 개수와 전체 row 개수를 고려한 분할

수정된 방식은 DPU 별로 merge하여 할당하는 방식이 아닌 전체 row 개수를 고려하여 DPU의 개수에 맞게 할당한다. 위의 그림에서, 각 색칠된 칸은 각 DPU 별로 할당된 row를 select 과정을 거쳐 산출한 것이다. 수정 전의 방식은 각 DPU 별 결과에서 row의 개수가 다르더라도 DPU id를 기준으로 재할당한다. 반면 수정 후의 상태인 그림의 경우, 전체 row 수는 36개이고 이전 단계에서 DPU를 6개 사용하였으므로 다음 단계에서는 3개의 DPU를 사용한다. 이렇게 구현한다면 DPU 별 정렬 속도가 비슷하게 맞춰지므로 총

대기 시간이 줄어들고, 이는 sort-merge join의 성능 향상점 중 하나이다. 이렇게 첫 번째 loop를 반복하면 이후에는 각 DPU에 들어있는 row의 개수가 동일하다. 할당 받은 배열에 대해 quick sort를 수행한 후, 이후부터는 DPU i에서 DPU i와 DPU i+1에 저장되어 있는 테이블의 row를 비교하여 sort-merge한다.

해당 방식의 시간 복잡도를 계산하였다. (DPU를 할당하는 시간은 제외한다.) 예를 들어 row의 개수가 MN인 dataset을 N개 row로 이루어진 M개 dataset으로 나누어 각각 DPU를 하나씩 할당하면 M-1번의 sort-merge 과정을 거친다. M개의 dataset을 처리하는 첫 번째 단계에서는 2N개의 row를 quick sort하여 merge하고 각 DPU는 병렬적으로 돌아가므로 첫 번째 단계가 종료되기까지는 O(2N*log(2N))의 시간이 소요된다. 이후의 i번째 단계부터 (M/2)번째 단계까지는 각 DPU가 O(2^(i-1)*N/NR_TASKLETS + 2^(i-1)*N/NR_TASKLETS)의 sort-merge를 반복한다. 따라서 전체 시간은 O(2N*log(2N) + 2^(i-1)*N/NR_TASKLETS + 2^(i-1)*N/NR_TASKLETS)이다. 반면, host에서 전체 row에 대해 quick sort를 적용한다고 하면 O(MN*log(MN))이다. 두 식을 정리하여 비교하면, UPMEM-PIM을 적용하여 설계한 알고리즘이 시간 측면에서 압도적으로 우수하다.

2.2. 구현 및 개선 사항

현재 하나의 테이블에 대해 DPU 개수와 tasklet 개수에 맞게 분할하여 할당 후 select하고, 산출된 DPU 별 결과에 대해 DPU에 할당, 이후 정렬된 배열을 row 단위로 비교하여 merge하는 과정을 구현했다.

- 기존 코드에서는 데이터를 주고 받기 위해 배열을 최대 크기로 초기화하고 있었다. 하지만 이 메모리를 다 쓸 일이 발생하지 않기 때문에 메모리 누수의 문제점이 있다고 판단하고 prim-benchmark를 참고하여 수정하게 되었다.

  host가 dpu에게 직접적으로 주는 정보는 테이블의 col_num과 row_num 뿐이며 데이터는 MRAM의 DPU_MRAM_HEAP_POINTER_NAME 부분에 쓰여진다. (즉, DPU 프로그램의 사용되지 않은 MRAM의 시작 부분) dpu는 받은 정보를 통해 크기를 알 수 있으므로 mram_read를 통해 읽어올 수 있다. dpu에서 host로의 전달도 동일하게 col_num과 row_num만 전달하며, 같은 위치에 write하여 host에서 접근 가능하다. 이를 통해 메모리가 낭비되는 문제는 해결할 수 있었으며, 크기가 커지면 wram overflow로 인해 컴파일부터 안되는 에러를 픽스하였다.
- DPU 별 결과를 merge하는 과정에서 더 많은 DPU를 사용할 수 있는 방안이 있을지 논의한다.

3. 9월 계획

- UPMEM-PIM을 활용한 sort-merge join 알고리즘 구현 완료
  - 메모리 관리
  - 다른 table에도 적용
  - join 수행 과정을 마친 최종 sort-merge join 알고리즘
- 성능 측정