HW5

2023 - 25222

Matirx Multiplication Challenge 2

1. 자신의 병렬화 설명
   A. 커널 최적화 : memory coalescing, tiling, vector type IO, thread granularity
   B. 그 외 최적화 : Multi GPU, triple buffering, 하나의 GPU내에서도 Martix A의 M행을 여러개의 SLICE로 나누어 연산

2. 성능 최적화를 위한 적용 방법 및 고려사항
   A. 커널 :타일링 및 shared memory 적용, vector type IO, thread당 4개의 ROW 연산 처리
   B. MPI 통신 : isend, irecv를 활용하여도 결국 절대적인 통신시간을 극복할 수 없어서 1node 보다 성능이 낮음
   C. Triple Buffering : stream을 사용해 통신 계산 중첩 적용

      (각 GPU 내에서도 행렬 A를 추가적으로 더 나누어서 중첩 더 적용)

3. Matmul.cu의 각 부분에 대한 설명
   A. Matmul_initialize(각 device에 stream 생성 및 memory 할당)

      cudaMalloc : 디바이스에 메모리 할당

      cudaSetDevice : 현재 사용중인 Device를 변경

      cudaStreamCreate : 스트림을 생성
   B. matmul(device memory로 host memory의 matrix를 copy하고, kernel로 연산 후 연산 결과를 다시 host로 copy함)

      cudaMemcpyAsync : 호스트와 디바이스 사이 데이터 전송

      cudaEventRecord : 스트림에 이벤트를 기록

      cudaStreamWaitEvent : 이벤트가 발생할때까지 스트림이 wait

cudaStreamSynchronize : 스트림 내의 작업이 끝날때가지 호스트에서 대기

C. matmul_finalize(할당된 자원 Free)

cudaStreamDestroy : 스트림을 반환하고 할당된 자원 해제

cudaEventDestroy : 이벤트 자원을 해제

cudaFree : 할당된 디바이스의 메모리를 해제

4. 코드 최적화 방식 분류 및 성능 실험 결과

첫번째 시도(8000GFLOPS) : multi-gpu 적용하고 커널은 shared memory tiling까지 적용

두번째 시도(7000GFLOPS) : MPI로 A는 Iscatter, B는 Ibcast, C는 Igahter하였으나 통신 오버헤드로 더 느려짐

세번째 시도(10000GFLOPS) : MPI를 최대한 활용하고자 Isend, Irecv로 행렬 A의 M행을 16개로 쪼개서 각 노드끼리 비동기적으로 주고 받도록 하였으나 아무리 MPI를 잘 짜봐도 20000GFLOPS를 달성하지 못함

네번째 시도(13000GFLOPS) : 커널로 돌아와서 Vector type IO와 thread granularity를 적용하여 성능을 더 올렸으나 한계에 도달했다 판단하여 MPI를 버리고 one-node multi-gpu로 노선 변경

다섯번째 시도(21000GFLOPS) : Triple buffering을 적용하고 MPI에서 얻은 아이디어를 통해

각 GPU로 나누어진 행렬 A를 GPU에서도 추가로 나누어 연산을 실행하고 통신을 실행하는 방법으로 최대한 통신과 연산을 overlap하여 최종 목표를 달성하게 함.