과제 #6

홍경인
M1522.006700 확장형 고성능 컴퓨팅 (001)

December 8, 2024

1 병렬화 방식 및 고려 사항

Thread 블록 단위 병렬화

- **블록 크기 정의**: 매크로로 정의된 `BM=64`, `BN=64`, `BK=8` 값을 통해 행렬 `A`와 `B`를 블록 단위로 분할하여 작업한다.
- **Shared memory 활용**: 행렬 `A`와 `B`의 타일을 각각 shared memory `As[BM][BK+1]` 및 `Bs[BK][BN+1]`에 저장한다. 이 때 `+1`의 padding을 통해 bank conflict를 방지한다.
- **데이터 재사용**: Shared memory에 적재된 데이터는 블록 내의 모든 thread에 의해 재사용되며, 이때 글로벌 메모리 접근을 최소화한다.
- **동기화**: `__syncthreads()`를 사용하여 모든 thread가 shared memory에 데이터를 로드할 때까지 대기하여 데이터 일관성을 유지한다. 동기화 횟수 역시 최소화하여 이로 인한 성능 저하를 줄일 수 있다.

Thread당 계산 부하 분산

- **레지스터 활용**: 각 thread는 계산 중간값을 저장하기 위해 `regM[TM]`과 `regN[TN]`을 활용하여 레지스터에 값을 저장한다. 레지스터는 GPU 메모리 계층 중 가장 빠르며, 접근 속도를 극대화한다.
- **타일링**: `TM=8`, `TN=8` 값을 통해 각 thread는 8×8 크기의 타일을 처리하며, thread 하나가 인접한 여러 값을 계산하도록 한다.

GPU 활용 병렬 처리

- **멀티 GPU 병렬처리**: 여러 GPU를 활용하여 행렬 곱셈 작업을 병렬로 분산 처리한다. 각 GPU는 행렬의 일부분을 담당하여 작업 속도를 향상시킨다.
- **Stream 활용**: 각 GPU 내에서 여러 stream을 생성하여 데이터 전송과 커널 실행을 병렬 처리한다. 이로 인해 데이터 전송 대기 시간이 최소화한다. 이때 비동기 메모리 복사와 커널 실행을 overlap하여 GPU 자원 활용도를 높인다.

2 각 부분별 설명

`matmul_initialize` 함수는 행렬 곱셈을 수행하기 전에 필요한 초기화 작업을 담당한다.

- `cudaGetDeviceCount(&num_device)`: 사용 가능한 GPU 디바이스의 수를 반환.
- `cudaSetDevice(i)`: 특정 index의 GPU 디바이스를 활성화.
- `cudaMalloc(&b_d[i], size)`: 지정된 크기만큼 GPU 메모리를 할당.
- `cudaStreamCreate(&streams[i][j])`: 새로운 CUDA 스트림을 생성.

`matmul` 함수는 실제 행렬 곱셈 연산을 수행한다. 이때 다중 GPU 및 스트림을 활용하고 계산 결과를 합친다.

- `cudaSetDevice(i)`: 특정 index의 GPU 디바이스를 활성화.
- `cudaMemcpyAsync(destination, source, size, cudaMemcpyHostToDevice, stream)`: 비동기적으로 데이터를 호스트에서 디바이스로 복사.
- `cudaStreamSynchronize(stream)`: 지정된 스트림의 모든 작업이 완료될 때까지 대기.
- `cudaMemcpyAsync(destination, source, size, cudaMemcpyDeviceToHost, stream)`: 비동기적으로 데이터를 디바이스에서 호스트로 복사.

`matmul_finalize` 함수는 행렬 곱셈 연산이 완료된 후 할당된 GPU 메모리를 해제하고, 생성된 스트림을 정리한다.

- `cudaSetDevice(i)`: 특정 index의 GPU 디바이스를 활성화.
- `cudaFree(pointer)`: 할당된 GPU 메모리를 해제.
- `cudaStreamDestroy(stream)`: 생성된 CUDA 스트림을 파괴.

3 버전별 설명

Kernel 1: 기본 구현

각 thread가 행렬 C의 하나의 원소를 계산하고, A와 B는 글로벌 메모리에서 직접 접근한다.

```
Avg. time: 2.184027 sec
Avg. throughput: 1006.866530 GFLOPS
```

Kernel 2: 블록 크기 최적화

`BLOCKSIZE`을 16으로 설정하여 thread 블록을 최적화하고, 블록 내 thread index를 재조정하여 coalesced memory access pattern을 형성한다.

```
Avg. time: 1.154694 sec
Avg. throughput: 1904.421082 GFLOPS
```

Kernel 3: hw5 모델

여러 GPU에서 여러 stream을 활용하여 계산이 이루어지도록 하고, 각 thread에서도 tile 단위로 데이터를 채우게 하여 coalesced access를 가능하게 한다. hw5와 유사한 모델이다.

```
Avg. time: 0.297718 sec
Avg. throughput: 7386.261276 GFLOPS
```

Kernel 4: kernel 최적화

Shared memory를 활용하는 패턴을 더욱 최적화하였다. 구체적으로는 각 thread에 필요한 메모리를 미리 로드해 두어 bank conflict가 발생할 여지를 줄였고, 인덱싱 및 디버깅에 용이하도록 A, B, C 포인터의 위치를 수정해 가며 행렬곱을 계산한다. 스트림의 경우 double buffering을 사용하여 kernel 연산과 데이터 전송을 overlap했다.

```
Avg. time: 0.132614 sec
Avg. throughput: 16582.112239 GFLOPS
```

Kernel 5: stream 최적화

Stream을 GPU 당 2개로 유지하며 double buffering하는 것의 성능이 기대만큼 크지 않아 hw5와 유사하게 각 GPU에 stream을 4개씩 할당하여 계산하였다.

```
Avg. time: 0.098736 sec
Avg. throughput: 22271.655637 GFLOPS
```