과제 #5

홍경인
M1522.006700 확장형 고성능 컴퓨팅 (001)

December 1, 2024

1 Matrix Multiplication Single GPU

이 코드는 CUDA를 이용하여 한 개의 GPU 위에서 행렬 곱셈을 병렬화하였다. 각 스레드 블록은 행렬의 일부분을 담당하며, shared memory를 활용하여 데이터 재사용을 극대화한다. Tile Size `TS`와 Work Per Thread `WPT`를 설정하여 계산 효율을 높였다.

- **Tiling:** 행렬을 작은 블록으로 분할하여 캐시 적중률을 높였다.
- **Shared Memory:** 전역 memory 접근을 최소화하고 shared memory에 데이터를 저장하여 접근 속도를 향상시켰다.
- **Work Per Thread:** 각 스레드가 여러 데이터 요소를 처리하도록 하여 병렬 연산의 효율성을 높였다.

코드의 각 부분은 다음과 같이 이루어진다. `matmul_initialize`는 CUDA device를 sec기화하고 memory를 할당하는 함수이다.

- `cudaGetDeviceCount`: 사용 가능한 CUDA device의 수를 얻는다.
- `cudaMalloc`: device memory에 memory를 할당한다.

`matmul`는 행렬 데이터를 device로 전송하고 커널을 실행한 뒤 결과를 수신하는 함수이다.

- `cudaMemcpy`: host와 device 간에 memory를 복사한다.
- `cudaDeviceSynchronize`: device에서 실행 중인 모든 연산이 완료될 때까지 대기한다.

`matmul_finalize`는 할당된 device memory를 해제하는 함수이다.

- `cudaFree`: device memory를 해제한다.

이와 같은 CUDA는 OpenCL보다 NVIDIA GPU에 특화되어 최적화된 성능과 개발 도구를 제공한다. 하지만 OpenCL은 다양한 하드웨어에서 동작하여 이식성과 호환성이 높다.

적용한 최적화 방식의 성능을 평가하기 위하여 $M = 4096$, $N = 4096$, $K = 4096$에서 실험을 수행하였다. 실험 결과는 아래와 같다.

`ms_01.cu`

기본적인 행렬 곱셈 구현으로, tiling과 shared memory 활용 없이 global memory만 사용하였다.

- **실행 시간:** 평균 15.211118 sec
- **처리량:** 15.211118 GFLOPS

`ms_02.cu`

Tiling과 shared memory를 도입하여 memory 접근 패턴을 최적화하였다.

- **실행 시간:** 평균 0.134149 sec
- **처리량:** 1024.526931 GFLOPS

`ms_03.cu`

WPT를 활용해 레지스터 최적화를 추가로 적용하여 연산 효율을 더욱 향상시켰다.

- **실행 시간:** 평균 0.071152 sec
- **처리량:** 1931.627437 GFLOPS

2 Matrix Multiplication Multi GPU

이 코드는 CUDA를 이용하여 여러 개의 GPU 위에서 행렬 곱셈을 병렬화하였다. 전체 행렬을 여러 GPU에 분할하여 할당하고, 각 GPU 내에서 여러 stream을 활용하여 병렬로 행렬 곱셈을 수행함으로써 계산 자원의 활용도를 극대화하였다. Tile Size `TS`, Work Per Thread `WPT`, 그리고 stream을 설정하여 데이터 전송과 계산을 동시에 수행하여 성능을 향상시켰다.

- **Tiling:** 행렬을 작은 블록으로 분할하여 캐시 적중률을 높였다.
- **Shared Memory:** 전역 memory 접근을 최소화하고 shared memory에 데이터를 저장하여 접근 속도를 향상시켰다.
- **Work Per Thread:** 각 스레드가 여러 데이터 요소를 처리하도록 하여 병렬 연산의 효율성을 높였다.

- **Multiple Streams:** 각 GPU에서 여러 stream을 사용하여 데이터 전송과 커널 실행을 병렬로 수행하였다.
- **Pinned Memory:** host memory를 고정 memory로 할당하여 데이터 전송 속도를 향상시켰다.
- **Multi-GPU Utilization:** 전체 행렬을 여러 GPU에 분산시켜 병렬 처리를 통해 계산 속도를 향상시켰다.

코드의 각 부분은 다음과 같이 이루어진다. `matmul_initialize`는 CUDA device를 sec기화하고 memory를 할당하는 함수이다.

- `cudaHostAlloc`: host memory를 pinned memory로 할당한다.
- `cudaGetDeviceCount`: 사용 가능한 CUDA device의 수를 얻는다.
- `cudaGetDeviceProperties`: 각 device의 속성을 조회한다.
- `cudaSetDevice`: 특정 GPU를 활성화한다.
- `cudaMalloc`: device memory에 memory를 할당한다.
- `cudaStreamCreate`: 새로운 CUDA stream을 생성한다.

`matmul`는 행렬 데이터를 device로 전송하고 커널을 실행한 뒤 결과를 수신하는 함수이다.

- `cudaMemcpyAsync`: 비동기적으로 host와 device 간에 memory를 복사한다.
- `cudaStreamSynchronize`: 특정 stream의 모든 작업이 완료될 때까지 대기한다.
- `memcpy`: host memory 간의 데이터를 복사한다.

`matmul_finalize`는 할당된 device memory를 해제하는 함수이다.

- `cudaFree`: device memory를 해제한다.
- `cudaStreamDestroy`: 생성된 CUDA stream을 파괴한다.
- `cudaFreeHost`: pinned memory로 할당된 host memory를 해제한다.

적용한 최적화 방식의 성능을 평가하기 위하여 $M = 16834$, $N = 4096$, $K = 4096$에서 실험을 수행하였다. 실험 결과는 아래와 같다.

`mm00.cu`

기본적인 다중 GPU 행렬 곱셈 구현으로, stream과 pinned memory를 사용하지 않았다.

- **실행 시간:** 평균 16.540845 sec

- **처리량:** 33.236260 GFLOPS

`mm01.cu`

Tiling과 shared memory를 도입하여 memory 접근 패턴을 최적화하였다.

- **실행 시간:** 평균 0.244407 sec
- **처리량:** 2249.347655 GFLOPS

`mm02.cu`

다중 stream과 pinned memory를 추가로 활용하여 데이터 전송과 계산을 병렬화하였다.

- **실행 시간:** 평균 0.164463 sec
- **처리량:** 3342.724376 GFLOPS