과제 #5

2022-29677 한주희

1 Matrix Multiplication Single GPU

- 행렬곱을 병렬로 수행하기 위해 입력 행렬을 row와 column 단위로 분할하여 처리하였다. Thread block과 thread를 2차원으로 설정하여 각 thread가 행렬 A의 일정 부분과 전체 행렬 B를 이용하여 C의 일정 부분을 계산하도록 하였다. 이를 통해 계산 작업을 병렬로 수행하며, 모든 thread가 각자 일을 하며 전체 행렬 C를 완성하도록 하였다.
- 성능 최적화를 위해 각 thread가 A와 C의 원소를 연속적으로 접근하도록 구현하여 연산의 속도가 빨라지도록 하였다. kernel 코드에서는 행렬을 tile로 나누어 GPU의 shared memory에 저장하여 재사용하는 tiling 방식을 채택하였다.
- `matmul_initialize` 함수에서는 `cudaMalloc`을 호출하여 행렬 A, B, C를 위한 디바이스 메모리를 할당한다. `matmul` 함수에서는 `cudaMemcpy`를 사용해 호스트로부터 디바이스로 행렬 A, B를 복사한다. 복사가 끝난 후 thread block과 thread를 2차원으로 설정하여 커널 코드(`matmul_kernel`)를 호출한다. 연산이 끝난 후 `cudaMemcpy`를 사용해 디바이스로부터 호스트로 행렬 C를 복사한다. `matmul_finalize` 함수에서는 `cudaFree`를 호출해 사용한 디바이스 메모리를 해제한다.
- OpenCL은 CUDA와 달리 CPU, NPU 등 여러 아키텍처에서 사용 가능한 portable한 코드를 제공한다. CUDA는 OpenCL과 달리 NVIDIA GPU 특성에 맞춘 다양한 함수를 제공하기 때문에 하드웨어를 더 세밀하게 조작할 수 있다.
- 코드 최적화 방식에 대한 성능 실험 결과
    1. TILESIZE = 16

```
srun: job 1069572 queued and waiting for resources
srun: job 1069572 has been allocated resources
Options:
  Problem size: M = 4096, N = 4096, K = 4096
  Number of iterations: 5
  Print matrix: off
  Validation: on

Initializing matrices...Done!
Calculating...(iter=0) 0.118557 sec
Calculating...(iter=1) 0.118243 sec
Calculating...(iter=2) 0.118049 sec
Calculating...(iter=3) 0.118031 sec
Calculating...(iter=4) 0.128593 sec
Validating...
Result: VALID
Avg. time: 0.120294 sec
Avg. throughput: 1142.521069 GFLOPS
```

2. TILESIZE = 32

```
srun: job 1069570 queued and waiting for resources
srun: job 1069570 has been allocated resources
Options:
  Problem size: M = 4096, N = 4096, K = 4096
  Number of iterations: 5
  Print matrix: off
  Validation: on

Initializing matrices...Done!
Calculating...(iter=0) 0.122363 sec
Calculating...(iter=1) 0.122256 sec
Calculating...(iter=2) 0.122548 sec
Calculating...(iter=3) 0.126557 sec
Calculating...(iter=4) 0.135346 sec
Validating...
Result: VALID
Avg. time: 0.125814 sec
Avg. throughput: 1092.398286 GFLOPS
```

3. Shared memory 미사용

```
srun: job 1069589 queued and waiting for resources
srun: job 1069589 has been allocated resources
Options:
  Problem size: M = 4096, N = 4096, K = 4096
  Number of iterations: 5
  Print matrix: off
  Validation: on

Initializing matrices...Done!
Calculating...(iter=0) 0.155032 sec
Calculating...(iter=1) 0.155064 sec
Calculating...(iter=2) 0.167520 sec
Calculating...(iter=3) 0.182608 sec
Calculating...(iter=4) 0.154990 sec
Validating...
Result: VALID
Avg. time: 0.163043 sec
Avg. throughput: 842.962144 GFLOPS
```

4. Shared memory 사용

```
srun: job 1069592 queued and waiting for resources
srun: job 1069592 has been allocated resources
Options:
  Problem size: M = 4096, N = 4096, K = 4096
  Number of iterations: 5
  Print matrix: off
  Validation: on

Initializing matrices...Done!
Calculating...(iter=0) 0.121673 sec
Calculating...(iter=1) 0.121786 sec
Calculating...(iter=2) 0.121645 sec
Calculating...(iter=3) 0.121605 sec
Calculating...(iter=4) 0.134653 sec
Validating...
Result: VALID
Avg. time: 0.124272 sec
Avg. throughput: 1105.948873 GFLOPS
```

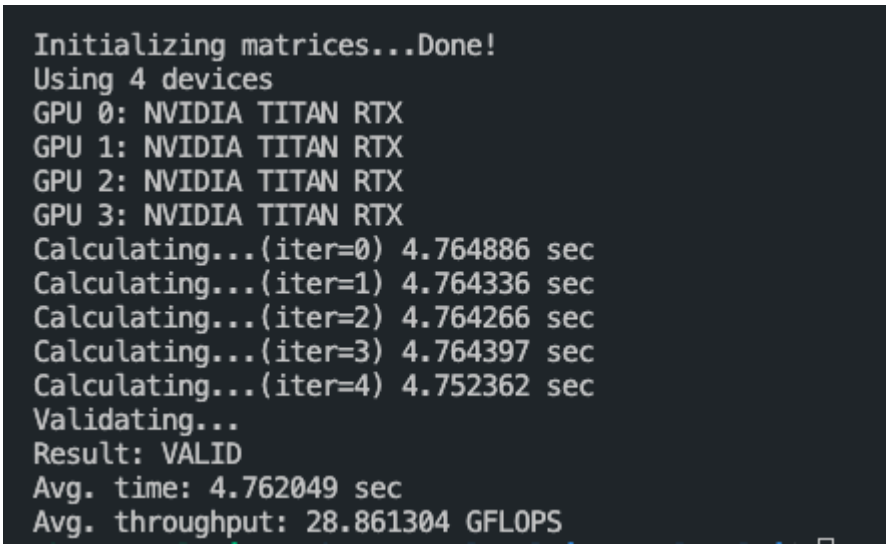
2 Matrix Multiplication Multi GPU

- 행렬 A는 행 단위로 여러 GPU에 분할되어 할당하고, 각 GPU는 자신에게 할당된 행렬 범위에 대해 독립적으로 연산을 수행하도록 하였다. CPU에서는 OpenMP를 사용하여 GPU 간 작업을

병렬로 수행하고 각 GPU의 작업은 별도의 OpenMP thread에서 비동기적으로 실행하도록 하였다. 또한, GPU 내부에서 작업을 병렬화하기 위해 Grid와 Block으로 작업을 나누었다. Block 내의 `TILESIZE×TILESIZE` thread들은 타일 단위로 행렬 곱셈을 수행하였으며 Grid는 `N/TILESIZE`와 `(Mend[i]-Mbegin[i])/TILESIZE`로 구성되어 행렬을 병렬 처리하였다.

- 성능 최적화를 위해 각 thread가 A와 C의 원소를 연속적으로 접근하도록 구현하여 연산의 속도가 빨라지도록 하였다. kernel 코드에서 A와 B tile을 shared memory에 저장하여 글로벌 메모리 접근 횟수를 감소시켰으며, 동일한 타일 데이터를 여러 thread에서 재사용하도록 하였다.
- `matmul_initialize` 함수에서는 `cudaSetDevice`를 호출해 사용할 GPU를 설정한다. `cudaMalloc`을 호출하여 행렬 A, B, C를 위한 디바이스 메모리를 할당한다. `matmul` 함수에서는 `cudaMemcpy`를 사용해 호스트로부터 디바이스로 행렬 A, B를 복사한다. 복사가 끝난 후 thread block과 thread를 2차원으로 설정하여 커널 코드(`matmul_kernel`)를 호출한다. 연산이 끝난 후 `cudaMemcpy`를 사용해 디바이스로부터 호스트로 행렬 C를 복사한다. `matmul_finalize` 함수에서는 `cudaFree`를 호출해 사용한 디바이스 메모리를 해제한다.
- 코드 최적화 방식에 대한 성능 실험 결과

  1. CUDA를 사용한 기초적인 구현



  2. OpenMP를 사용한 병렬 처리

```
Initializing matrices...Done!
Using 4 devices
GPU 0: NVIDIA TITAN RTX
GPU 1: NVIDIA TITAN RTX
GPU 2: NVIDIA TITAN RTX
GPU 3: NVIDIA TITAN RTX
Calculating...(iter=0) 0.108868 sec
Calculating...(iter=1) 0.106432 sec
Calculating...(iter=2) 0.139637 sec
Calculating...(iter=3) 0.130704 sec
Calculating...(iter=4) 0.103535 sec
Validating...
Result: VALID
Avg. time: 0.117835 sec
Avg. throughput: 1166.363825 GFLOPS
```

3. OpenMP 활용한 단일 루프 병렬화

```
Initializing matrices...Done!
Using 4 devices
GPU 0: NVIDIA TITAN RTX
GPU 1: NVIDIA TITAN RTX
GPU 2: NVIDIA TITAN RTX
GPU 3: NVIDIA TITAN RTX
Calculating...(iter=0) 0.044749 sec
Calculating...(iter=1) 0.044833 sec
Calculating...(iter=2) 0.045459 sec
Calculating...(iter=3) 0.045509 sec
Calculating...(iter=4) 0.045690 sec
Validating...
Result: VALID
Avg. time: 0.045248 sec
Avg. throughput: 3037.453555 GFLOPS
```