Lab5: Tiled Matrix Multiplication

소프트웨어학과 201921085 곽수정

1. Describe your design and implementation

우선 사용자가 원하는 matrix 크기를 Ax(행), Ay(열), Bx(행), By(열)로 받아오도록하였습니다. Host와 device에서 사용할 matrix를 1차원 배열로 선언하였기 때문에 A와 B matrix를 같은 크기로 동적할당하여 1차원 배열이지만 마치 2차원 배열처럼 사용하기 위해 가장 큰 행/열 크기를 구하였습니다. 따라서 입력받은 행열의 크기에 따라 vector_size를 정해주고 vector_size만큼 Ahost, Bhost에 동적할당 해주고 0으로 초기화해주었습니다. 각각의 for문으로 실제 행렬 값이 들어가야하는 부분에는 0~5.0 사이의 랜덤한 float 값을 저장해주었습니다.

이어서 CPU와 GPU에서 사용할 A, B, C matrix를 동적할당 해주고 host에서 device로 data를 cudaMemcpy 해주었습니다. 선언한 TILE_WIDTH로 dimGrid와 dimBlock을 지정하여 MatrixMulKernel 함수를 실행하도록 하였습니다.

Tiled kernel 의 MatrixMulKernel 에서는 하나의 thread block 마다 TILE_WIDTH * TILE_WIDTH 개의 thread 가 동시에 연산하게 되는데 이 thread 들이 shared memory 에 tile 을 읽어오고, 이를 사용하여 연산하기 위해 tile 크기와 동일한 A_shared, B_shared 행렬을 선언하여 memory bandwidth 를 줄일 수 있도록 하였습니다. thread 들이 Adevice 행렬과 Bdevice 행렬을 shared memory 로 load 하여 여러 thread 들이 사용 가능하도록 하였고 두 tile 에 대해 모든 thread 들이 fetch 할 때까지 기다리기 위해 __syncthreads 를 사용하였습니다. 이후 shared memory 에 있는 행렬로 연산을 한 후에 또 모든 thread 들이 현재 tile 에 대한 연산을 완료할 때까지 기다리기 위해 __syncthreads 를 사용하였습니다. shared memory 를 이용하여 계산 후 축적시킨 값을 global memory 인 Cdevice 에 저장하여 행렬 곱을 수행하도록 하였습니다.

반면 Non Tiled kernel 의 MatrixMulKernel 에서는 shared memory 를 사용하지 않으며, thread 들이 행렬을 load 하고 store 할때마다 global memory 에 접근하도록 구현되어있습니다.

Tile 와 non tiled kernel 둘 다 시간을 측정하여 성능을 비교할 수 있도록하였고, CPU 에서 serial 하게 실행되는 행렬 곱셈 결과와 비교하여 행렬 곱 결과가 일치하는지 출력할 수 있도록 하였습니다. 행렬 곱 결과가 일치하는지 확인할 때, 실수형 자료는 부동소수점 형태로 구현되어 있기 때문에 두 값의 차가 매우 작은 값보다 더 작다면 일치하는 것으로 판단하였습니다.

(결과값 비교 부분은 serial 하게 동작하여 오래걸리기 때문에 TILE_WIDTH 변화에 따른 성능 비교 시에 값 일치 부분은 주석처리하였습니다.)

2. Make a performance comparison for each of small, medium, large matrix

- Non-tiled kernel (TILE_WIDTH = 16 으로 고정 후 측정)

TILE_WIDTH	small - (32 * 32), (32 * 32)	Medium - (256 * 256), (256 * 256)	Large - (4096 * 2048), (2048 * 4096)
time (ms)	0.037	0.623	1655.744

- Tiled kernel

Tile 크기를 정하고, 3 가지 크기의 matrix 연산에 대해 성능 측정을 하기 위해 tile 은 8 * 8 부터 TILE_WIDTH 를 두배씩 늘려가며 시간을 측정하였습니다.

1) Small - (32 * 32), (32 * 32)

TILE_WIDTH	4	8	16	32
time (ms)	0.051	0.039	0.037	0.040

Matrix size 가 small 일때 TILE_WIDTH 를 16 까지 늘렸을 때는 성능이 개선되었지만, 32 로 TILE_WIDTH 가 커졌을 때는 오히려 시간이 더 오래 걸리게 되었습니다.

3) Medium - (256 * 256), (256 * 256)

TILE_WIDTH	4	8	16	32
time (ms)	1.159	0.424	0.351	0.036

Medium matrix 에서는 TILE_WIDTH 를 늘렸을 때 성능이 개선된 것을 좀 더 뚜렷하게 볼 수 있었지만 small 처럼 32 TILE_WIDTH 에서 오히려 시간이 더 오래 걸렸습니다.

3) Large - (4096 * 2048), (2048 * 4096)

TILE_WIDTH	4	8	16	32
time (ms)	1588.934	1131.003	961.827	898.472

Large matrix 에서는 TILE_WIDTH 가 커질수록 뚜렷하게 성능이 향상된 것을 볼 수 있었으며, 32 TILE_WIDTH 에서도 시간이 더 짧게 걸렸습니다.

GPU Information (deviceQuery)에서 max thread per block 이 1024 로, TILE_WIDTH 가 32 보다 커질 수 없음을 알 수 있었습니다. 위 실행 결과를 통해 행렬의 크기가 작을 때는 TILE_WIDTH 도 작은 경우가, 행렬의 크기가 큰 경우에는 TILE_WIDTH 가 32 인 경우가 가장 좋은 성능을 보였습니다.

또한 global memory 로 load 하고 store 하는 시간을 shared memory 를 통해 bandwidth overhead 를 줄여 non-tiled kernel 보다 tiled kernel 이 더 좋은 성능을 보이는 것을 확인할 수 있었습니다.