[Lab-5] CUDA programming (tiled matrix multiplication)

소프트웨어학과 201720707 나용성

[구현 설명]

tiled kernel은 non tiled kernel에서의 compute를 여러 단계로 나눈 뒤 각 단계에서 global data인 M matrix, N matrix를 tiled 단위로 shared memory로 load하고 결과값에 더한 다음 모든 단계간 진행됐을 때 결과값을 결과를 저장할 global data에 저장하는 방식으로 구현된다.

```
for(int tileNUM = 0; tileNUM < gridDim.x; tileNUM++){
        int i = tileNUM * TILE_WIDTH + threadIdx.y;
        int j = tileNUM * TILE_WIDTH + threadIdx.x;

        d_M_tile[threadIdx.y][threadIdx.x] = d_M[Row * width + j];
        d_N_tile[threadIdx.y][threadIdx.x] = d_N[i * width + Col];

        __syncthreads();
```

그림1. Shared memory에 TILE을 load하는 파트

각각의 iteration시작점에서 Thread block의 thread들이 global data에서 shared memory로 load할 tile의 index를 구한 뒤 shared memory로 load하고 matrix multiplication을 하기 전에 모든 데이터가 load되어 있어야 하므로 barrier역할로 __syncthreads() API를 호출한다.

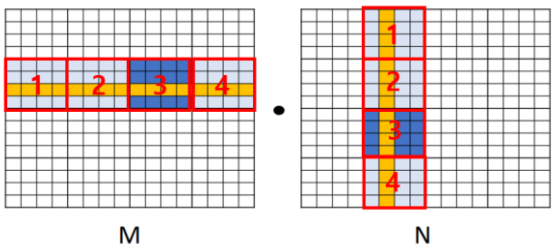


그림2. tile load의 예시

과제 소개 자료를 예시로 위의 코드가 위 데이터에서 실행됐을 때 4*4 크기 tile을 load할 때 위의 순서대로 각 iteration에서 tile들이 load된다.

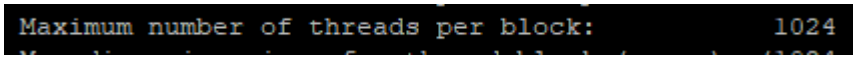
```
        for(int k = 0; k < TILE_WIDTH; k++)
                Pvalue += d_M_tile[threadIdx.y][k] * d_N_tile[k][threadIdx.x];
        __syncthreads();
}
d_P[Row*width+Col] = Pvalue;
```

그림3. Shared memory에 load된 tile을 이용해 multiplication을 수행하는 코드

[tile 크기 기준]

먼저 tile의 너비와 높이는 일치하도록 설정되는데 thread block안의 thread는 2의 제곱수 개가 포함되어야 적절하게 warp단위로 나뉘기 좋을 것이다. 그러므로 너비와 높이는 2의 제곱수 중 하나를 사용할 것이다. 그리고 서버에서 사용된 GPU의 block마다 사용할 수 있는 최대shared memory는 49152bytes인데 한 block에서 두개의 tile을 shared memory에 저장해둬야 하고 data의 자료형은 float이므로 최대로 사용 가능한 tile의 셀의 개수는 49152 / (2 * 저장해둘 tile개수 *

sizeof(float)(=4)) = 6144이므로 2의 제곱수 중 가능한 최대 tile의 너비는 64이다. 그러나 또 다른 bounded resource인 thread는 한 block에서 허용되는 최대 개수가 1024개 이므로 tile의 최대 너비 32로 shared memory per thread block보다 더 tile의 최대 width를 많이 제한시킨다..



이 때 32 * 32크기 tile로 나눠도 충분히 많은 양의 tile이 존재 가능하면 TILE_WIDTH를 32로 설정하는 것이 좋을 것이고 만약 작은 work set에서는 여러 tile이 나오지 못하므로 TILE_WIDTH를 줄여서 여러 tile이 나오도록 하는 것이 parallel processing에 좋을 것이다.

성능측정

1) Small matrix size

TILE_WIDTH	2	4	8	16	32
Execution_time(ms)	0.0256	0.040	0.01843	0.017408	0.0184

Tile의 너비가 16일 때 가장 좋은 성능을 나타냈으나 성능이 향상되는 정도가 매우 미미했다. Working set이 작기 때문에 너비가 32일 땐 16보다 성능이 떨어졌다.

1) Medium matrix size

TILE_WIDTH	2	4	8	16	32
Execution_time(ms)	1.0508	0.1637	0.0485	0.0417	0.04854

Tile의 너비가 16일 때 역시 가장 좋은 성능을 나타냈다. 이번엔 너비가 2일 때와 비교하여 두배 이상의 성능 향상을 보였으나 여전히 32너비의 tile을 쓸 땐 성능이 악화됐다.

2) Large matrix size

TILE_WIDTH	2	4	8	16	32
Execution_time(ms)	5136.48	904.96	340.48	215.62	186.80

Tile의 너비가 32일 때 가장 좋은 성능을 나타냈다. 너비가 2일때와 비교하여 약 30배 정도의 성능 향상을 보일 정도로 큰 차이가 있었다. Working set이 충분히 크기 때문에 tile의 크기가 커져도 tile의 개수가 많아 이점을 온전히 활용한 것으로 보인다.

Non tile processing

Working set	Small matrix size	Medium matrix size	Large matrix size
Execution_time(ms)	0.017408	0.088064	670.01

Small matrix size에선 tile에서의 최고성능과 차이가 없었고 medium matrix size에선 두배 가량 차이가 났고 large matrix size에선 약 네배 정도의 차이가 있음을 보아 working set이 클수록 shared memory로 load하는 방식이 효과적임을 알 수 있다.