**Assignment 4**

(GPU 컴퓨팅)

담당교수: 공영호 교수님

학과: 컴퓨터정보공학부

학년: 3

학번: 2018202074

이름: 김상우

#코드 작성

과제를 들어가기 앞서, 과제 조건에 대하여 살펴보았습니다.

* Vertor addition code의 넓이는 WIDTH로 따로 정의되며 16으로 정의된다.
* 각 Block에 대하여 TILE\_WIDTH는 변수로서 정의되며 기본적으로 8의 값을 갖는다.
* 해당 코드는 두 행렬에 대하여 행렬 곱 연산을 한다.
* a,b array는 가로와 세로 길이가 같은 n\*n 행렬로, 결과 또한 n\*n 행렬이다.

즉, WIDTH \* WIDTH의 크기의 행렬 2개를 a,b를 선언하고 gpu환경에서 행렬곱을 실행, 해당 결과를 저장한 C 행렬을 CPU에서 출력해주는 것을 목표로 하면 될 것입니다. 위 정보를 바탕으로 다음과 같이 코드를 진행하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Google colab에서 해당 코드를 Assignment4cuda.cu라는 코드로 인식하기 위해 위에 %%writefile Assignment4cuda.cu로 기술하였고 아래 필요한 헤더들을 기술하였다.

또한 행렬곱을 수행하는 함수를 CPU에서 GPU로 명령하기 위해 \_\_global\_\_을 추가하였다. 또한global 적인 요소의 위치를 알아야 행렬곱의 진행이 가능하기 때문에 blockIdx.x \* blockDim.x를 이용해 전체 grid에서 현재 block의 x축으로의 위치로 간 후 threadidx.x를 통해 block내 요소에 대해 자세한 global적인 위치를 계산하였다. 마찬가지로 blockIdx.y \* blockDim.y를 이용해 전체 grid에서 현재 block의 y축으로의 위치로 간후 threadidx.y를 통해 block 내 요소에 대해 자세한 global적인 위치를 계산하였다. 이들은 각각 x, y에 저장되었다. 행렬곱의 경우 cij를 구하기 위해선 a의 i번째 행의 n번째 요소와 b의 j번째 열의 n번째 요소를 곱한 것들의 합을 구하면 되므로 위와 같이 표현하였다. 각 요소에 대한 계산을 통해 구해진 값들을 sum 내에 더하고 해당 값을 c에 저장해주며 해당 코드를 구현했다.

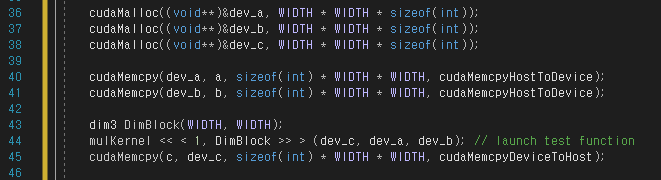
이때 2차원 배열에 대하여 가로 x, 세로 y번째에 위치한다면 y \* width + x를 적용하는 식으로 2차원배열을 1차원 배열로서 표현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Array는 width \* width 행렬이므로 가 5이므로 2차원 배열의 형태로 a,b,c를 선언하고 c에 대해서는 0으로 초기화해주었다. 이후 해당 배열들에 대해 행렬곱이 잘 진행되는 지 보기 위해 위와 같이 구분이 가는 값들로 초기화해주었다.

이후 GPU에서 a, b 그리고 행렬곱의 결과를 담을 c에 대하여 처리하기 위해 WIDTH \* WIDTH 행렬, c를 선언하고 위 배열들의 값을 GPU에서 받거나 저장할 dev\_a, dev\_b, dev\_c 또한 선언해주었다.



텍스트이(가) 표시된 사진

자동 생성된 설명

cudaMalloc을 이용, dev\_a, dev\_b, dev\_c에 int가 WIDTH \*WIDTH갯수 만큼 들어갈 수 있도록 GPU메모리를 할당시켰다.

이후 cudaMemcpy를 이용, GPU의 메모리를 할당받은 dev\_a, dev\_b에 CPU내의 a,b값들을 복사해준다. 이때 HOST에서 Device로 값을 int가 WIDTH \* WIDTH개 들어가는 만큼 복사하므로 3번째 인사는 sizeof(int) \* WIDTH \* WIDTH, 4번쨰 인자는 cudaMemcpyHostToDevice가 들어가게 된다.

이후 dim3를 이용, 2차원 형렬에 맞는 형태로 GPU에 값을 넣어 동작시킬 수 있도록 하기 위해 사용하였다. dimGrid의 경우 DimBlock을 1번째 인자\*인자 만큼 받게되어야 하며 2차원 배열이기에 WIDTH/ TILE\_WIDTH로 각 인자를 넣어주었다. 위 경우 전체 행렬은 16\*16 , block은 8\*8의 형태를 가지므로 각 인자는 2,2,1가 될 것이다. DimBlock의 경우 TILE\_WIDTH에 적힌대로 8\*8의 형태를 가지게 될 것이다. 이들은 mulkernel의 내부에서 기준이 되어 동작될 것이다. 이때 행렬의 크기를 함수가 인지해야 하므로 이를 나타내기 위해 WIDTH를 추가적으로 인자로서 넣어주었다. GPU에서 모든 2차원 배열의 요소 대해 실행할 수 있도록 하였다. 위의 mulKernel의 결과로 gpu로부터 받은 값을 받은 dev\_c의 값을 cudaMemcpy를 이용 cpu의 c배열에 복사한다. Device에서 Host로 값을 int가 WIDTH \* WIDTH개 들어가는 만큼 복사하므로 3번째 인사는 sizeof(int) \* WIDTH \* WIDTH, 4번쨰 인자는 cudaMemcpyDeviceToHost가 들어가게 된다.

텍스트이(가) 표시된 사진

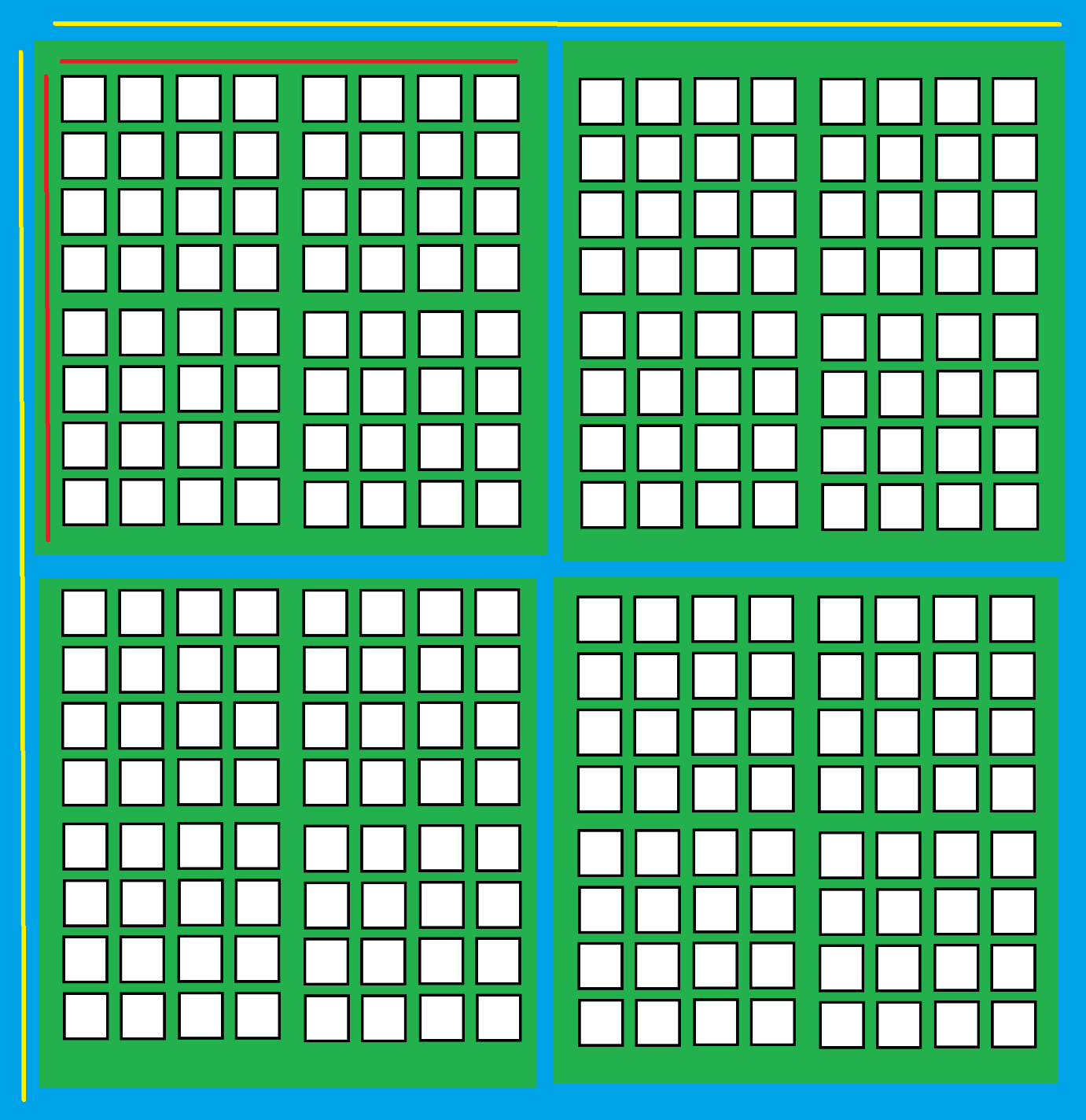
자동 생성된 설명

이후 a,b,c의 배열의 각 요소를 출력한다. 예상이 맞다면 난수의 값들인 a[i],b[i]의 처음에 초기화한 값들과 함께 행렬곱이 진행된 값이 c[i][j]를 통해서 출력될 것이다.

또한 Tile\_WIDTH 기준에 맞추어 \n과 ‘ ’에 의해 8\*8 Block 간 사이에 공백이 생기게 될 것이다.

이후 dev\_a, dev\_b, dev\_c에 할당된 GPU 메모리를 cudaFree로 해제하고 return 0로 코드를 종료한다.

#Draw Structure



위에서 작성한 코드를 기반으로 Structure를 그리게 된것이다.

파란부분은 grid, 녹색 부분은 각 block, 하얀 네모들이 thread가 된다.

이때, 우린 16\*16 행렬에 대해 8\*8행렬을 기준으로 위 함수들을 진행할 것이기 때문에 WIDTH를 16, TILE\_WIDTH를 8로 명시했다. 그 때문에 Block의 너비는 빨간색 줄로 표현된 8이 되었고, 실제로 block당 8\*8의 형태로 thread가 들어있다. 또한 16\*16행렬에 대해 8\*8 행렬을 기준으로 나눠, grid는 노란색 줄이 나타내는 WIDTH / TILE\_WIDTH, 즉 2\*2의 block으로 이뤄진 형태로 나타내어 지고 있다.

#Google Colab

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

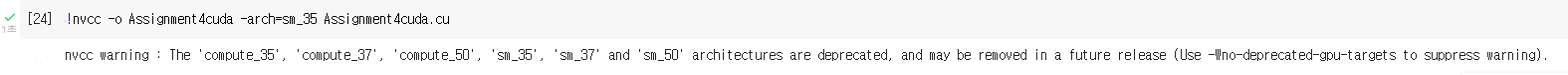
테이블이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

위에서 작성된 코드를 적용하였다. 맨 위에 적힌 코드에 의해 Assignment4cuda.cu가 생기게 된다.



이후 nvcc -o를 이용 cuda capability를 sm\_35 버전으로 컴파일 할 수 있도록 실행파일을 생성하며 이때 이름을 Assignment4cuda로 지정해주었다.

테이블이(가) 표시된 사진

자동 생성된 설명

(a행렬 출력)

테이블이(가) 표시된 사진

자동 생성된 설명

(b행렬 출력)

테이블이(가) 표시된 사진

자동 생성된 설명

(c행렬 출력)

이렇게 생긴 실행파일을 !./Assignment4cuda를 통해 실행하자, 기존의 초기화한 a,b의 행렬과 이들에 대해 행렬곱을 실시한 값이 알맞게 출력되는 것을 확인할 수 있다.

다른 값에서도 알맞게 나와야 하므로 다른 값 또한 넣어보았다.

텍스트이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

TILE\_WIDTH를 2로 바꿔 block을 2\*2행렬로 변경하여 진행하였다.

결과가 적합하게 나온 것을 확인했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

TILE\_WIDTH를 4로 바꿔도 적합한 값이 나오는 것을 확인했다.

#고찰

GPU 컴퓨팅 Assignment4을 진행하면서 google colab에서 Matrix Multiplication을 진행하며 assignment 3에 대해 복습할 수 있었습니다. 또한 block에 대해서 GPU는 어떤 식으로 동작해야 하는 지를 위의 과정을 통해 알 수 있었습니다. 또한 Assignment3에서 배운 GPU를 가진 가장 큰 이점인 병렬에 대하여 2차원 배열에 대해 어떻게 적용하는지 복습할 수 있었습니다. 2차원 배열로 thread를 분해하는 것이 계산에 있어 도움이 될 수 있다는 사실과 blockid와 같은 추가적인 변수를 알아가는 것이 가장 큰 수확이었던 것 같습니다.