GPU 컴퓨팅 Assignment 4

이름 : 이준휘

학번 : 2018202046

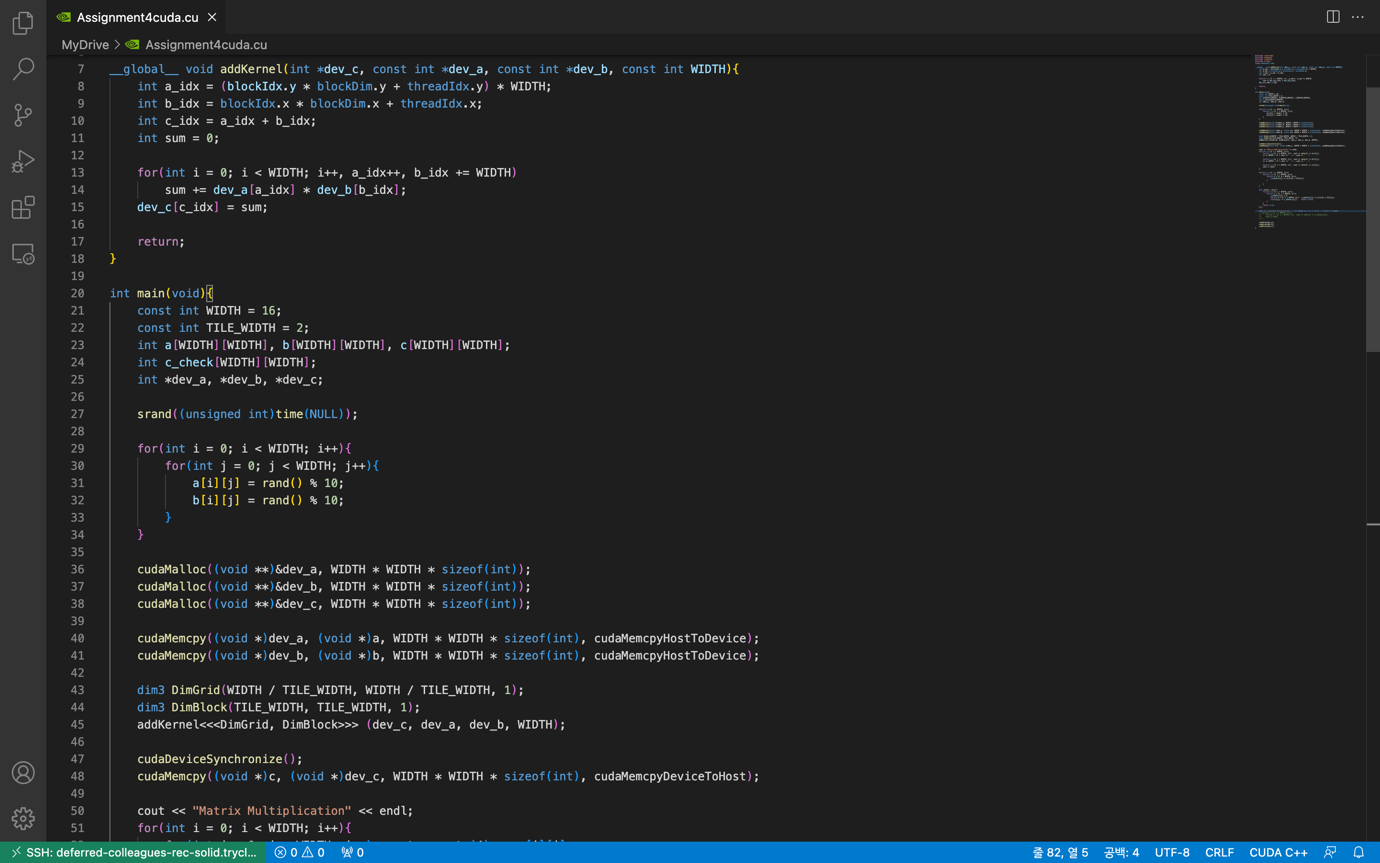
교수 : 공영호 교수님

강의 시간 : 월 수

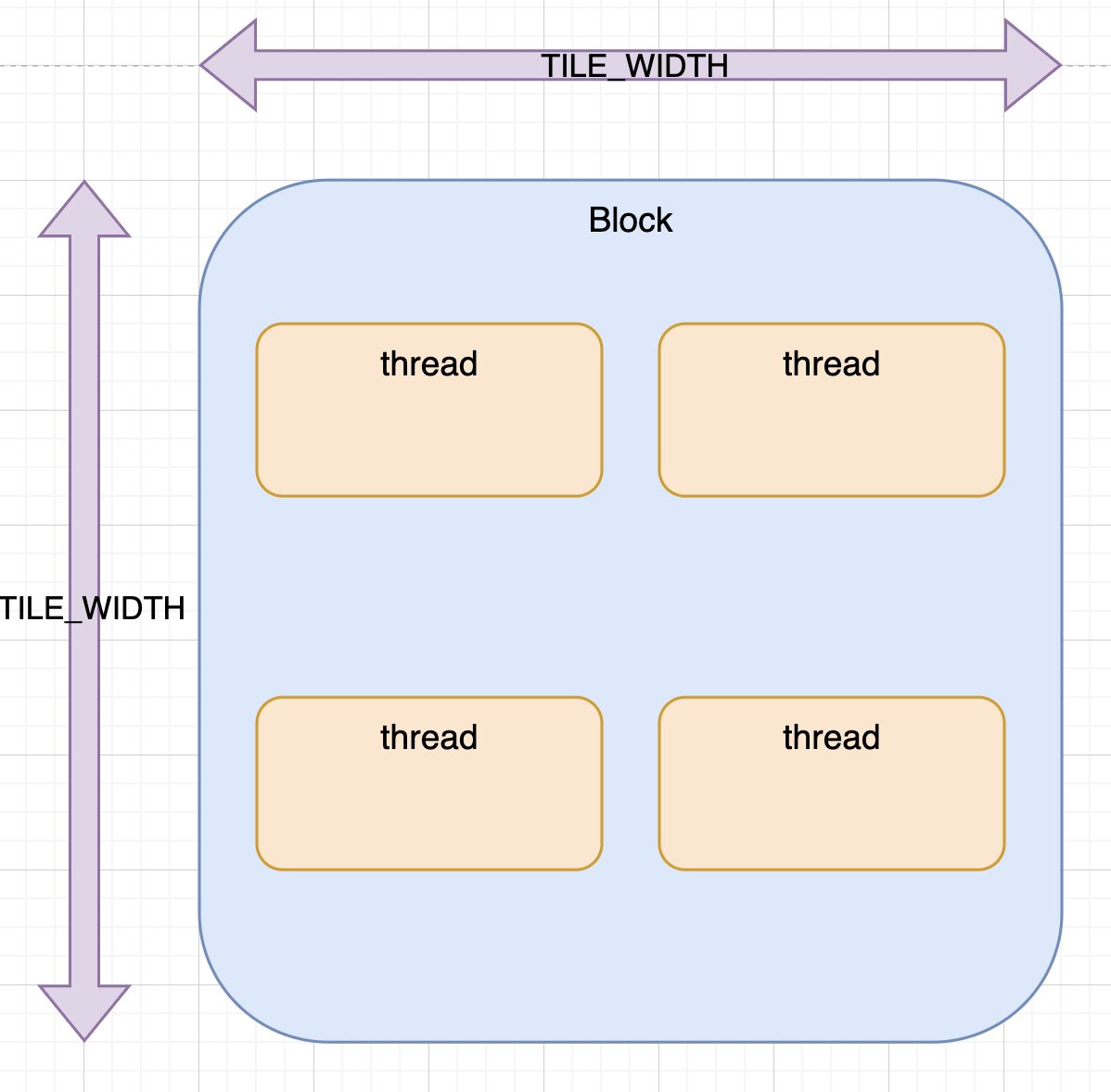
1. Introduction

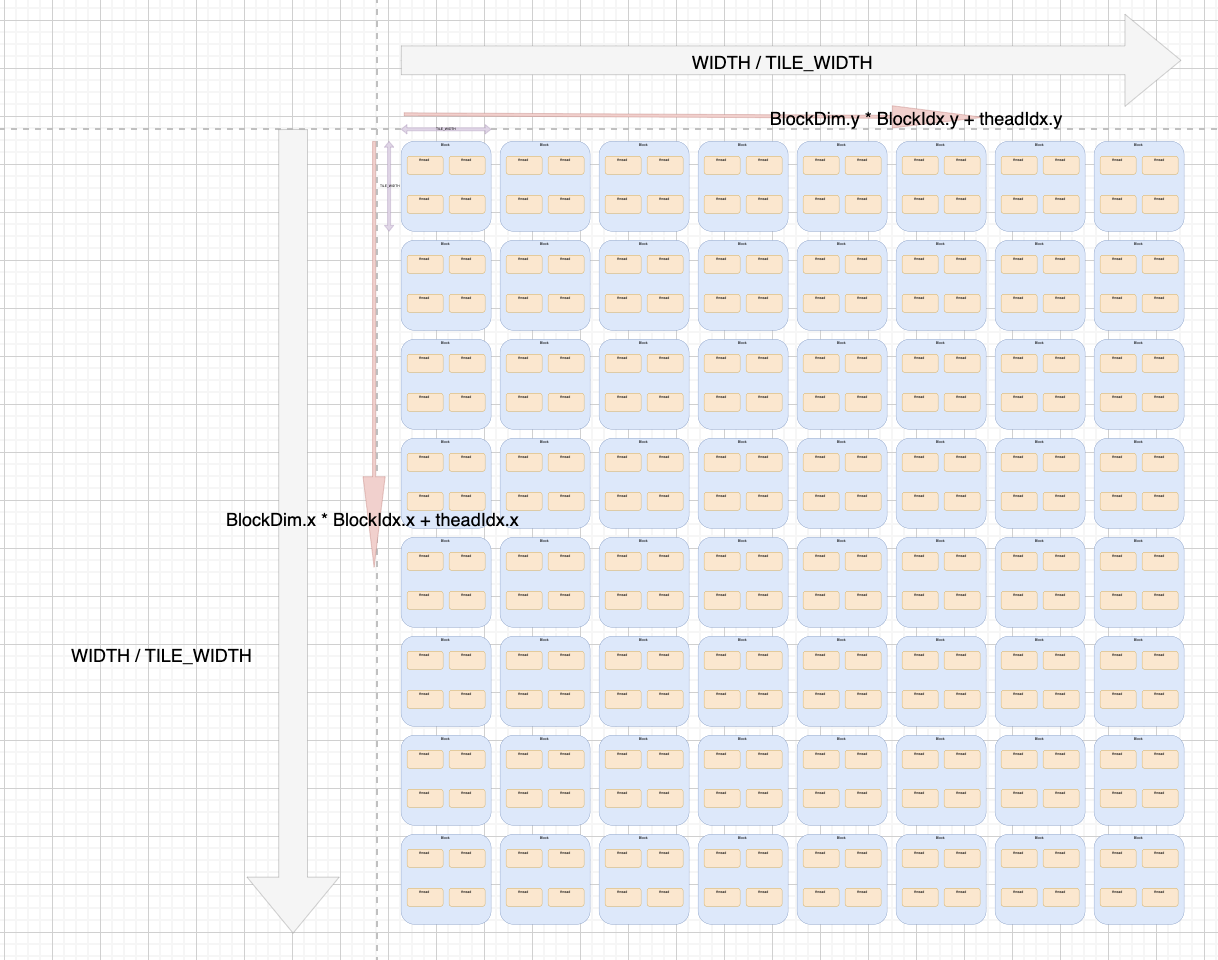
해당 과제는 다음 조건에 맞는 코드를 구현한다. 행렬 A와 B간의 행렬 곱 연산을 수행하여 나온 결과를 C에 저장하는 코드를 작성한다. 이 때 행렬 연산에서 TILE\_WIDTH와 WIDTH를 이용하여 배열의 크기와 thread의 Dimension, block의 Dimension을 가변으로 설정한다. 이를 통해 1 < block 개수 < 16를 만족하도록 코드를 작성한다.

1. Approach



addKernel() 함수는 \_\_global\_\_ 매개변수를 받아 Host에서 Device에 함수를 수행하도록 명령한다. 해당 함수는 \_\_global\_\_를 사용하기 때문에 return은 void를 사용한다. parameter로 저장할 위치 int \*dev\_c와 연산할 값 const int \*dev\_a, \*dev\_b, 그리고 Matrix의 크기 WIDTH를 사용한다.





해당 함수에서는 3가지 idx를 사용한다. dev\_a의 idx 값은 행렬 곱 연산에서 특정 행에서 다음 열로 한 칸씩 이동한다. 이 때 해당 block의 Dimension 또한 고려해야 함으로 기준점을 (blockIdx.y \* blockDim.y + threadIdx.y) \* WIDTH로 잡는다. blockIdx.y \* blockDim.y는 블럭 단위로 우선 이동한다는 의미이며, threadIdx.y를 더함으로서 해당 block 내의 thread 순서로 이동할 수 있다. 다음으로 dev\_b의 idx값은 특정 열을 기준으로 행 단위로 움직이기 때문에 위와 같이 block을 고려하여 blockIdx.x \* blockDim.x + threadIdx.x로 설정한다. 마지막으로 더할 곳 c의 idx는 특정 행과 열임으로 a\_idx와 b\_idx를 더함으로써 구할 수 있다.

이후 for문에서는 WIDTH만큼 반복하는 코드다. 이 때 a\_idx는 반복 시마다 증가, b\_idx는 WIDTH만큼 증가시키며 연산을 수행한다. sum에 dev\_a의 idx위치의 값과 dev\_b의 idx위치의 값을 곱한 값을 더한다. 모든 for문의 연산 후에는 해당 값을 dev\_c의 idx위치의 값에 저장한다.

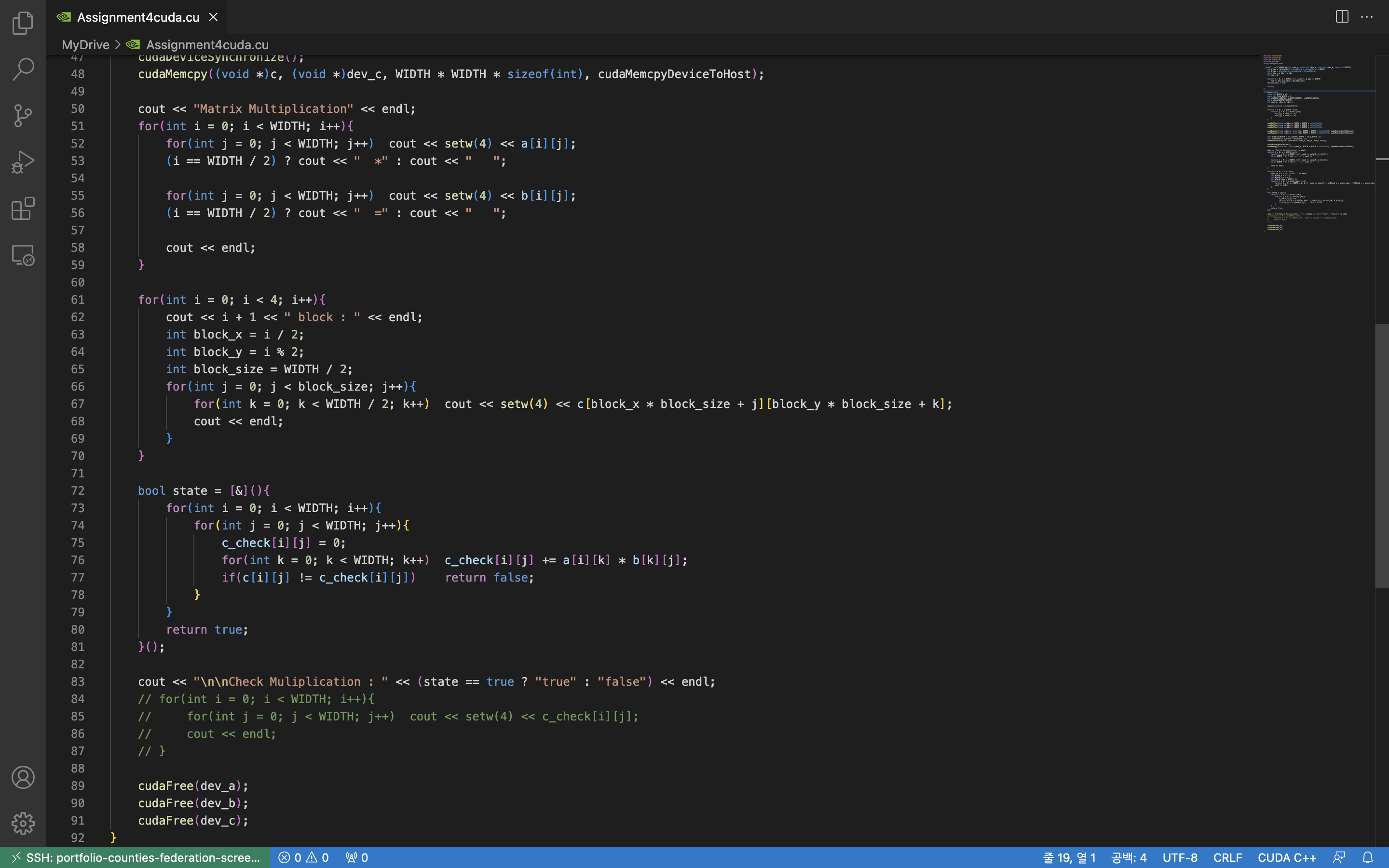
Main 함수는 다음과 같이 진행된다.

행렬의 크기는 const int 형태로 WIDTH에 16를 TILE\_WIDTH에 2를 할당한다. 또한 해당 값을 바탕으로 int 행렬 a, b, c, c\_check를 생성한다. 그 후 Device에서 사용할 pointer를 위한 int \*dev\_a, \*dev\_b, \*dev\_c를 생성한다.

srand() 함수를 통해 seed값을 현재 시간으로 설정한 후, for문을 통해 a, b 행렬에 random 값을 할당한다. 할당하는 random value의 크기는 10 미만으로 설정한다.

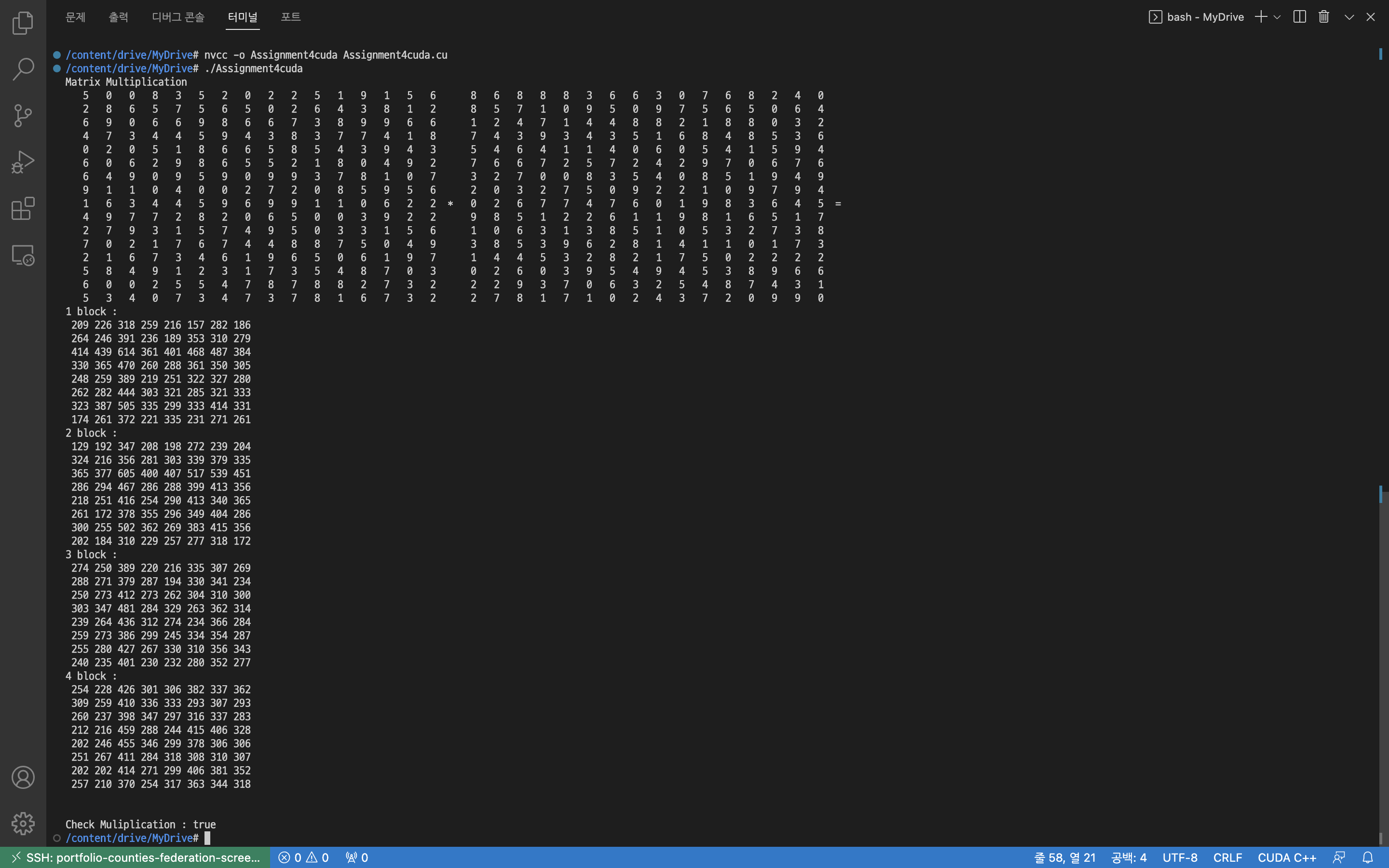
cudaMalloc() 함수에서는 dev\_a, dev\_b, dev\_c 포인터에 WIDTH \* WIDTH \* sizeof(int) 크기의 메모리를 할당한다. 그 후 dev\_a, dev\_b 에 a, b의 값을 복사하는 cudaMemcpy()를 수행한다. 해당 복사는 Host -> Device임으로 cudaMemcpyHostToDevice 옵션을 추가한다.

Device는 (WIDTH / TILE\_WIDTH) \* (WIDTH / TILE\_WIDTH)개의 block, block 내에서는 TILE\_WIDH \* TILE\_WIDTH 개수의 thread를 사용할 예정임으로 dim3 변수 DimGrid, DimBlock 변수의 값을 각각에 맞게 설정한다. 그 후 addKernel<<< DimGrid, DimBlock >>> (dev\_c, dev\_a, dev\_b, WIDTH)는 2D WIDTH \* WIDTH 크기의 ID를 가진 Thread에서 addKernel 함수를 수행한다. cudaDeviceSynchronize() 함수를 통해 Device 동기화를 수행한 후 결과로 나온 dev\_c의 값을 c로 옮겨주기 위한 cudaMemcpy() 함수를 수행하며, 이 때는 Device -> Host 임으로 cudaMemcpyDeviceToHost 옵션을 활용한다.



결과를 출력한 후 람다 함수를 통해 CPU에서 Multiplication을 수행했을 때의 결과와 비교하여 같은 지를 확인한다. 마지막으로 GPU에서 동적 메모리 할당을 해제하기 위한 cudaFree()함수를 수행하며 이후 프로그램을 종료한다.

1. Result



해당 화면은 Colab을 SSH로 연결하여 해당 프로그램을 컴파일, 수행한 모습이다. 위와 같이 정상적으로 컴파일이 되며, 결과가 출력된 것을 확인할 수 있다. 해당 행렬 연산이 CPU와 같은 결과를 보임으로 정상적으로 구현되었음을 알 수 있다.

1. Consideration

해당 과제를 통해 기존 행렬곱을 일부 수정하여 2D Block과 2D thread를 사용하는 법을 익힐 수 있었다. 이 때 1D index를 찾으면서 blockDim과 blockIdx, 그리고 threadIdx를 통해 접근하는 법을 알 수 있었다. 행렬의 연산이 결과와 같이 빠른 시간 내에 수행 할 수 있다는 사실을 통해 이러한 GPU를 사용하였을 때 유용한 연산 형태를 가늠할 수 있었다. 특히 block과 thread의 수를 수업 때 배운 warp 개념과 SM 개념을 통해 코드를 작성하면 GPU를 더욱 효율적으로 사용할 수 있을 것이라 생각이 들었다.

1. Reference

강의자료만을 참고