GPU컴퓨팅

(공영호 교수님)

Assignment4

학번: 2019202103

학과: 컴퓨터정보공학부

이름: 이은비

**Assignment4**

**<Introduction>**

1개의 thread block을 사용하면 최대 1024개의 threads만 사용 가능합니다. 따라서 ent3에서 실습한 내용을 심화하여 여러 개의 thread block을 사용하여서 Matrix multiplication을 실습합니다. 정방행렬이라 가정하고 전체 행렬은 WIDTH, blockDim은 TILE\_WIDTH, GridDim은 WIDTH/TILE\_WIDTH로 구분하고 Assignment3에서 구현한 Matrix multiplication의 global index를 수정하여 matrix multiplication을 진행합니다.

<the structure of grid/block/thread reflecting TILE\_WIDTH>

<grid 2x2(WIDTH/TILE\_WIDTH)>

WIDTH=16

TILE\_WIDTH=8

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | Block  (0,0) |  |  |  |  |  |  |  | Block  (0,1) |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | Block  (1,0) |  |  |  |  |  |  |  | Block  (1,1) |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

<block(0,0)은 TILE\_WIDTHxTILE\_WIDTH(8x8)>

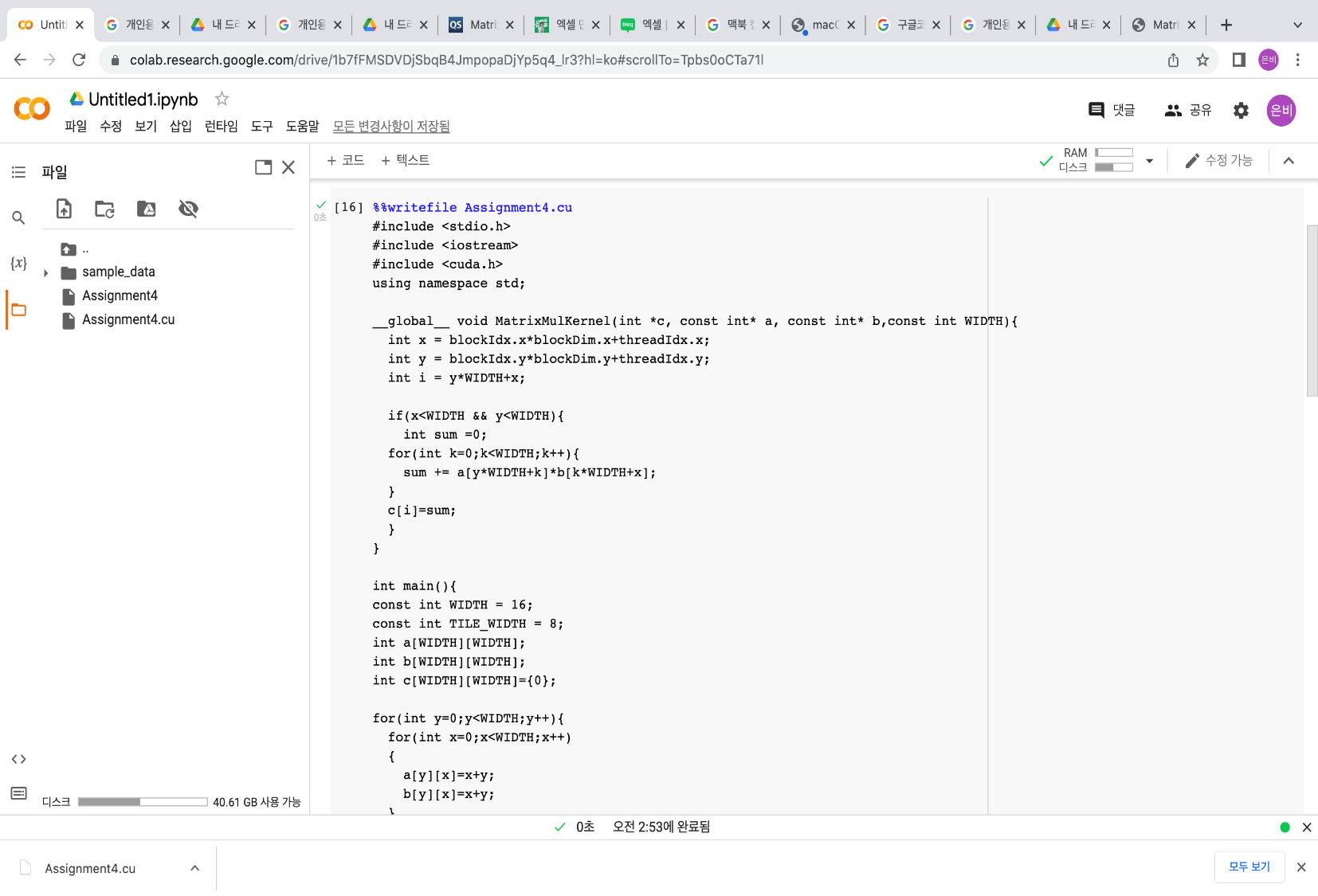
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Thread  (0,0) | Thread  (0,1) | Thread  (0,2) | Thread  (0,3) | Thread  (0,4) | Thread  (0,5) | Thread  (0,6) | Thread  (0,7) |
| Thread  (1,0) | Thread  (1,1) | Thread  (1,2) | Thread  (1,3) | Thread  (1,4) | Thread  (1,5) | Thread  (1,6) | Thread  (1,7) |
| Thread  (2,0) | Thread  (2,1) | Thread  (2,2) | Thread  (2,3) | Thread  Block  (0,0)  (2,4) | Thread  (2,5) | Thread  (2,6) | Thread  (2,7) |
| Thread  (3,0) | Thread  (3,1) | Thread  (3,2) | Thread  (3,3) | Thread  (3,4) | Thread  (3,5) | Thread  (3,6) | Thread  (3,7) |
| Thread  (4,0) | Thread  (4,1) | Thread  (4,2) | Thread  (4,3) | Thread  (4,4) | Thread  (4,5) | Thread  (4,6) | Thread  (4,7) |
| Thread  (5,0) | Thread  (5,1) | Thread  (5,2) | Thread  (5,3) | Thread  (5,4) | Thread  (5,5) | Thread  (5,6) | Thread  (5,7) |
| Thread  (6,0) | Thread  (6,1) | Thread  (6,2) | Thread  (6,3) | Thread  (6,4) | Thread  (6,5) | Thread  (6,6) | Thread  (6,7) |
| Thread  (7,0) | Thread  (7,1) | Thread  (7,2) | Thread  (7,3) | Thread  (7,4) | Thread  (7,5) | Thread  (7,6) | Thread  (7,7) |

TILE\_WIDTH=8

**<Conclusion>**

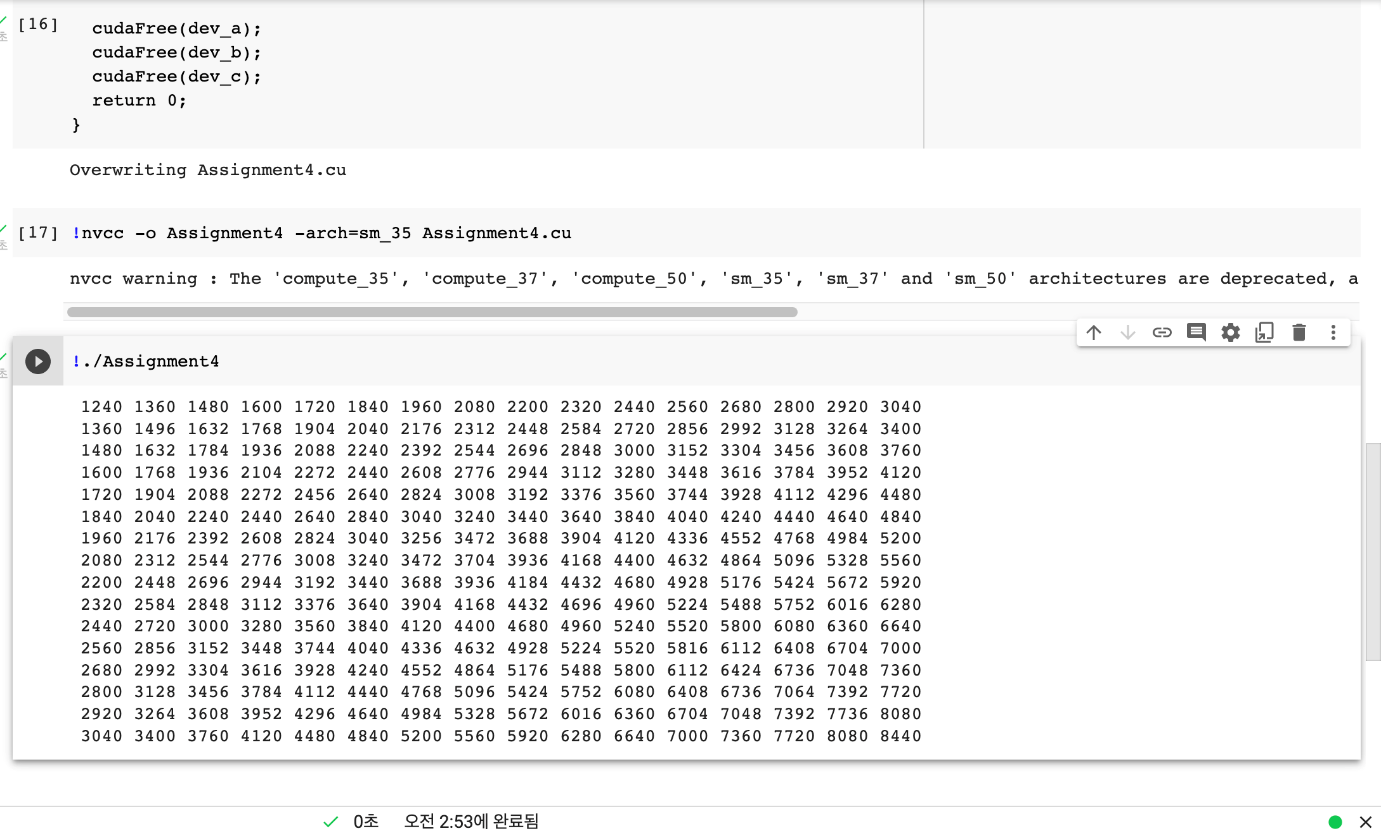
**[Assignment4.cu의 코드내용 및 결과화면]**

**(전체 코드내용)**

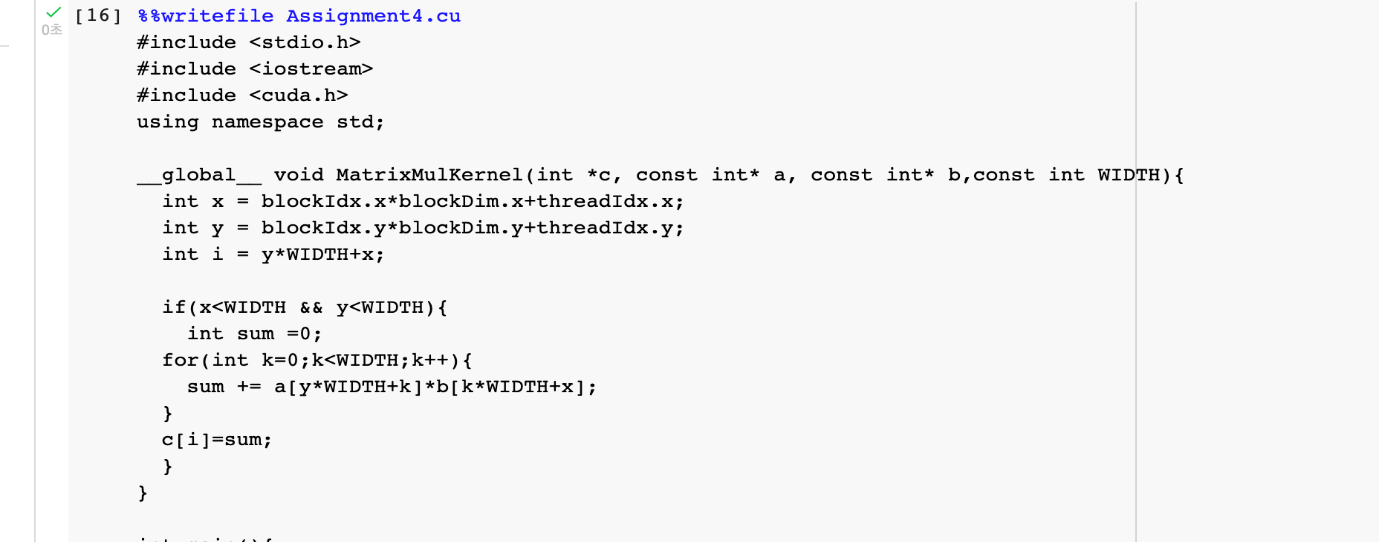






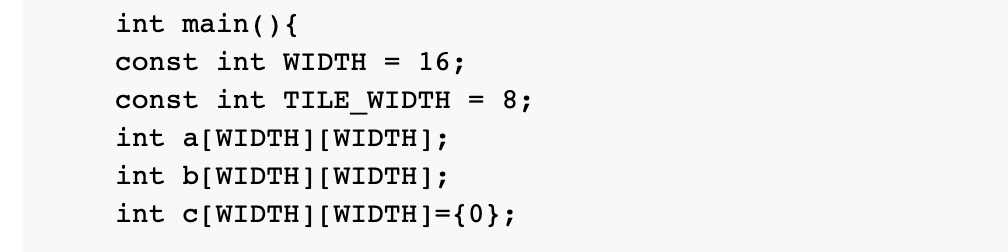


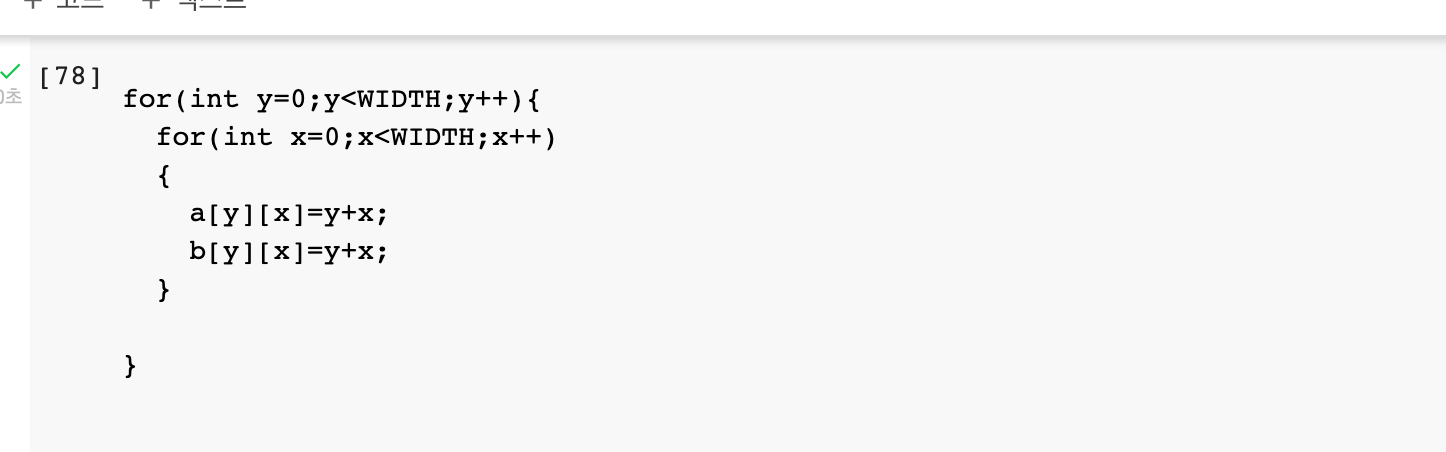
(부분 코드1)



CUDA kernel function을 준비합니다. 이때 \_\_global\_\_은 CPU에서 호출 되어서 GPU에서 실행 될 수 있도록 합니다. 함수의 이름은 MatrixMulKernel이며 인수로는 cpu에서 작성된 배열을 copy받아서 곱해지는 대상인 2개의 Matrix인 const int\* a,b와 Matrix Multiplication의 결과 Matrix를 다시 CPU에 copy해줄 int \*c,그리고 loop계산에서 쓰이게 될 Matrix size 인 WIDTH를 const int로 받아옵니다. 또한 함수의 내용을 보면 x와 y는 Multi blocks을 사용하는 것을 고려하여서 x= blockIdx.x\*blockDim.x+threadIdx.x; y= blockIdx.y\*blockDim.y+threadIdx.y로 설정합니다. 또한 배열의 index i는 y\*(blockDim.x)+x;로 1d로 전체 index에 접근할 수 있도록 합니다. 또한 행과 열이 Matrix size를 넘어가지 않는 조건에서 각각의 thread별로 sum 변수로 a의 행과 b의 열을 곱한 결과를 저장하고 각각의 결과값(sum값)을 c Matrix의 각각의 원소(thread)에 할당합니다.

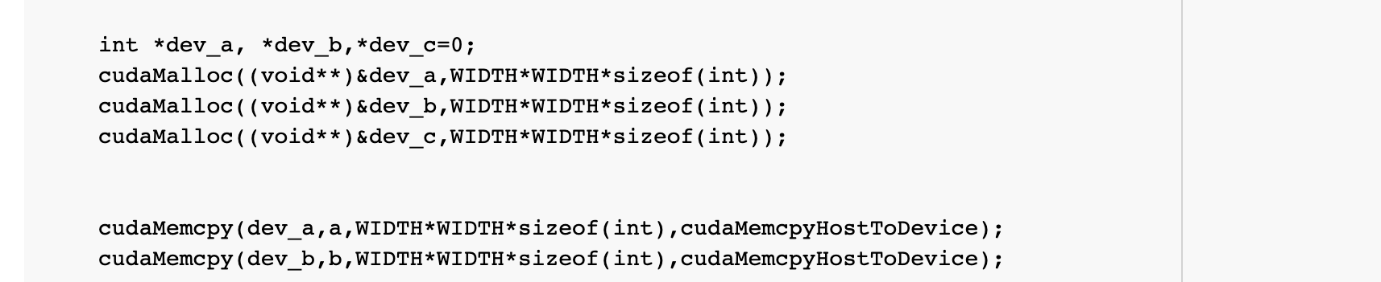
(부분 코드2)

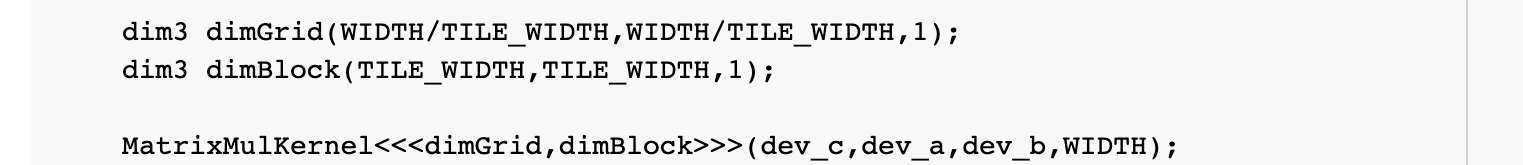




배열의 size인 WIDTH는 16, block의 size가 될 TILE\_WIDTH는 8로 설정합니다. Kernel 힘수 mulkerenl에 전달 하게 될 Matrix a,b의 원소를 initialize합니다. Mulkernel의 결과를 전달 받을 c Matrix또한 초기화 합니다..

(부분코드3)





cudaMalloc function을 이용하여 Kernel의 메모리를 할당합니다. 할당되는 사이즈는 전달 받을 사이즈인 matrix의 전체 원소(threads)와 저장될 값인 int형의 size를 각각 곱한 값과 같습니다.

cudaMemcpy function을 이용하여 host의 data를 device로 copy합니다. a,b배열의 값을 각각dev\_a,dav\_b,로 copy합니다.

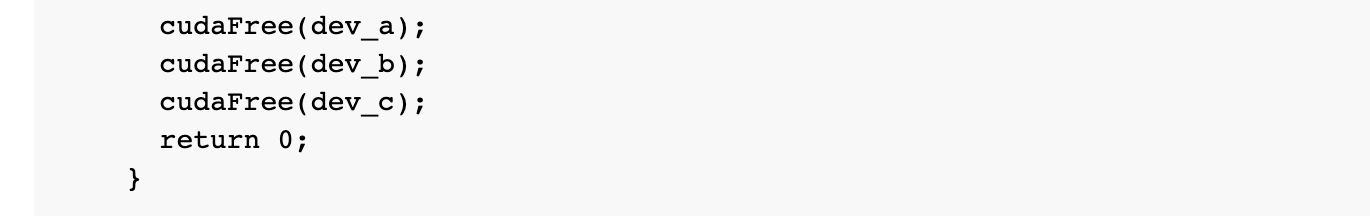
dim3 dimGrid(x,y,z)에 각 차원에 해당하는 값 즉 Grid차원에서의 크기 WIDTH/TILE\_WIDTH를 설정합니다. 실습에서는WIDTH/TILE\_WIDTH는 16/8=2입니다.

같은 방법으로 dim3 dimBlock의 Block차원에서의 크기 TILE\_WIDTH를 설정합니다. 실습에서는 TILE\_WIDTH는 8입니다.

MatrixMulKerenl<<<dimGrid,dimBlock>>>(dev\_c,dav\_a,dev\_b,WIDTH); 1부터 WIDTH까지 앞서 prepare된 함수(\_\_global\_\_void MatrixMulKenel)를 호출합니다.

(부분코드4)



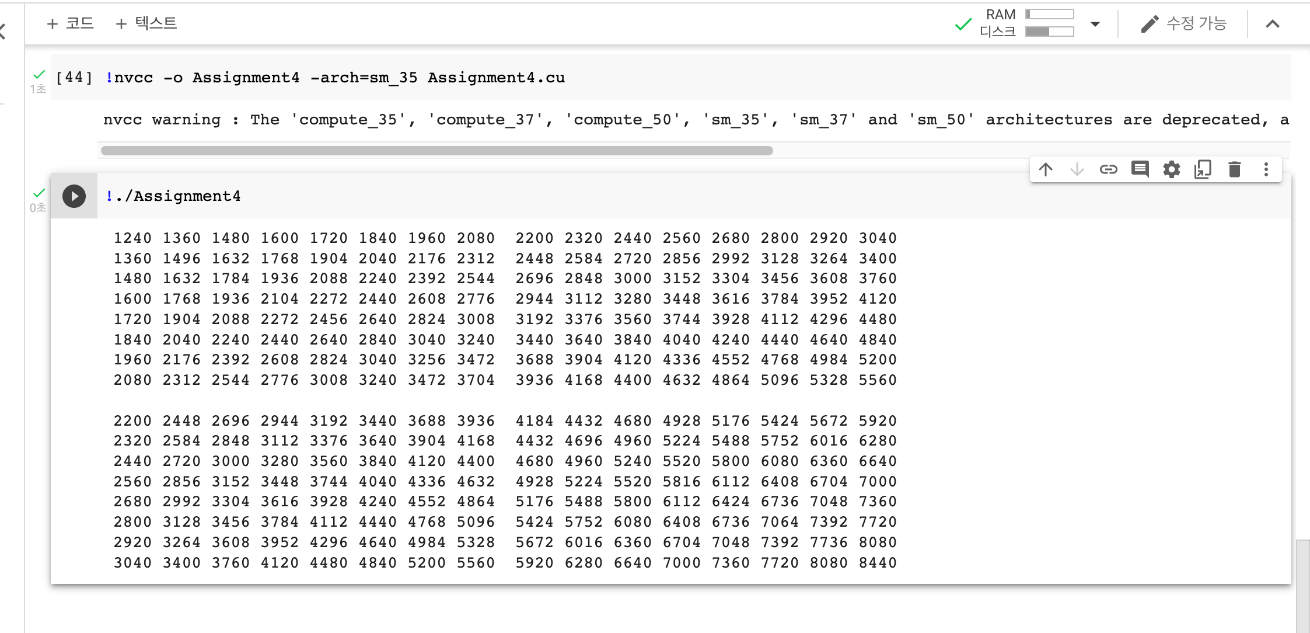


이후 kernel에서 수행된 결과 값인 cMatrix를 cudaMemcpy로 CPU로 전달합니다.

CPU에서 Printf함수로 kernel로부터 전달받은 결과인 c Matrix를 2D형식으로 출력합니다.

연산이 종료된 후에는 cudaFree();함수로 앞서 cudaMalloc으로 할당되었던 변수의 메모리를 해제 합니다.

(결과화면)



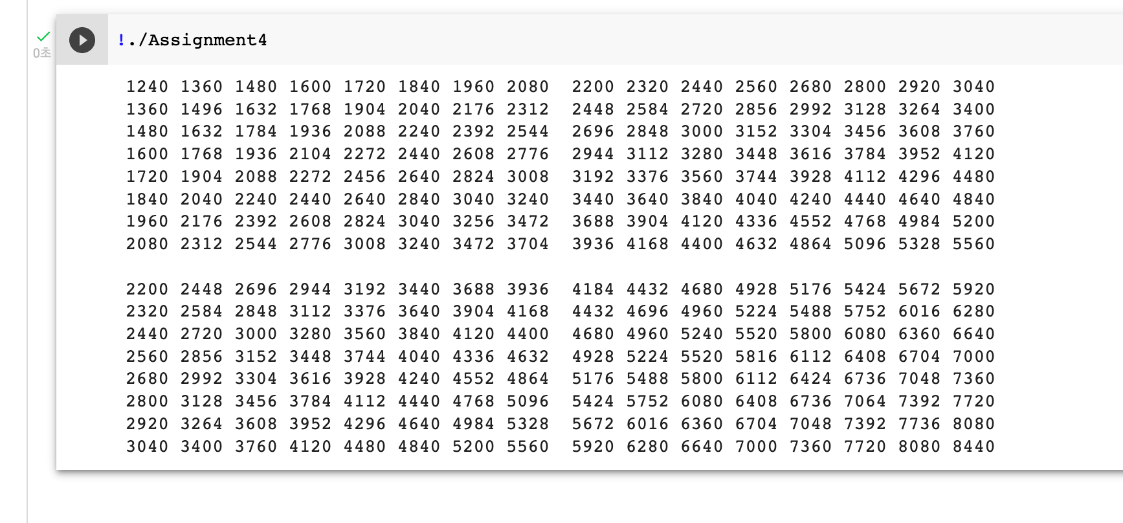
!nvcc -o Assignment4 -arch\_35 Assignment4.cu

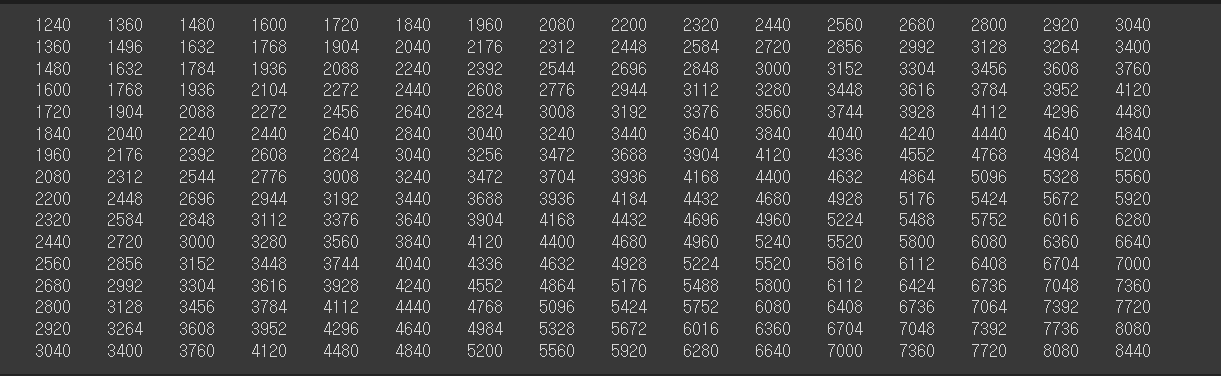
앞서 작성되었던 Assignment4.cu파일을 Assignment4라는 이름의 실행파일로 컴파일합니다.

!./Assignment3

실행파일을 실행시켜서 printf로 작성된 c의 Matrix의 결과가 2d형식으로 출력됩니다.

(결과 matrix값 확인)





위의 결과화면은 위의 내용과 같이 CUDA language를 이용하여 나온 matrix multiplication결과 이고, 아래 결과는 cpu에서 c language를 이용하여 (3개의 loop를 사용) matrix multiplication결과 입니다. 동일한 결과임을 확인 하였습니다. Block단위의 matrix로 구분하여 출력하였습니다.