GPU컴퓨팅

(공영호 교수님)

Assignment5

학번: 2019202103

학과: 컴퓨터정보공학부

이름: 이은비

**Assignment5**

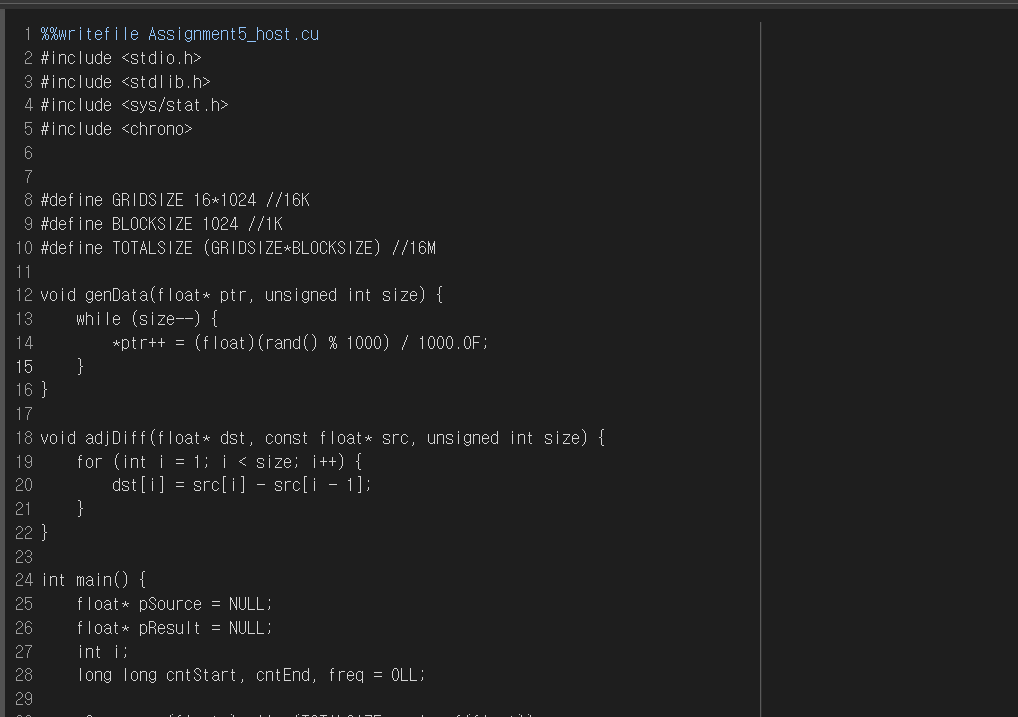
**<Introduction>**

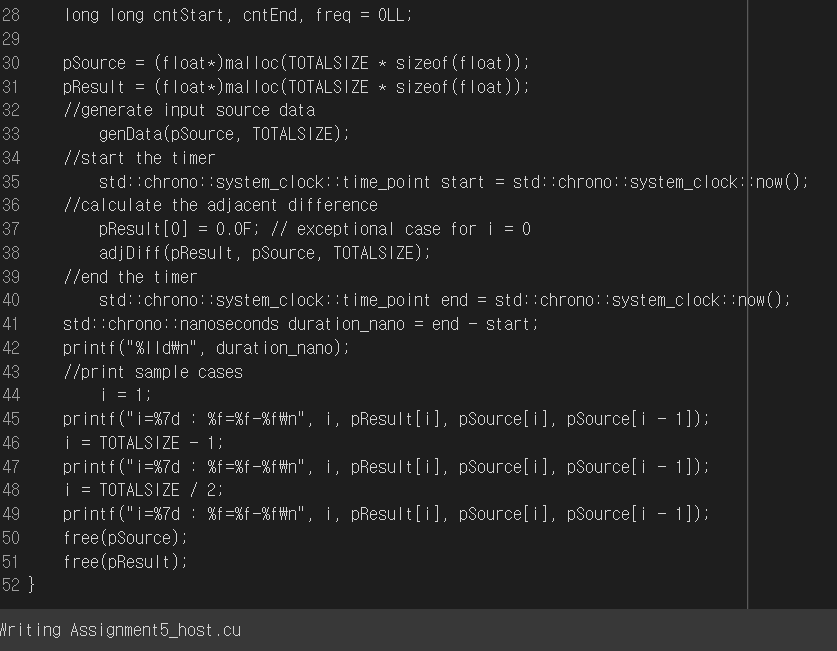
임의의 instruction를 수행한 시간을 측정함을 통해 CPU,GPU의 시간적 측면에서의 성능을 비교합니다. 과제에서 수행하는 instuction은 Adjacent difference로 인접한 수를 빼서 결과 변수에 저장하는 작업이며, Adjacent difference in Host version, Adjacent difference in CUDA Global memory version , Adjacent difference in CUDA Shared memory version으로 3가지 로 나눠서 하며, 시간측정은 c++library의 chrono를 이용합니다. 과제에서의 코드 진행은 다음과 같습니다. (cuda)Malloc the memory , Generate input source (using rand() function) (at cpu), Too many input sources to declare each input values manually(성능 차이를 확실히 비교하기 위해) Start the timer(시각을 저장) , Adjacent difference function call(instruction 수행), End the timer(시각을 저장), Calculate the differences(end time – start time) , Write the data on the disk , Print some sample cases 순으로 진행합니다.

**<Conclusion>**

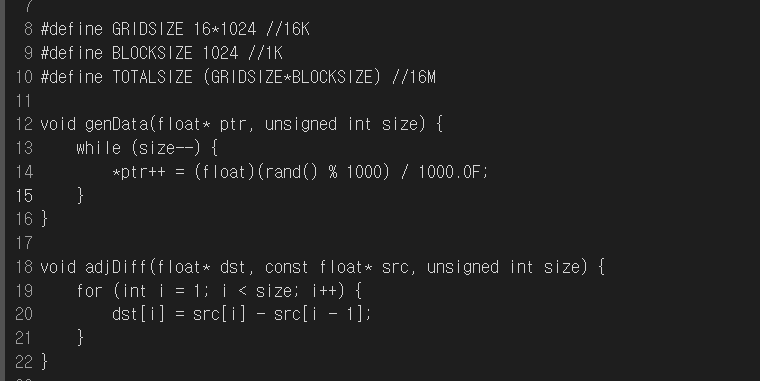
**[Assignment5\_host.cu의 코드내용 및 결과화면]**

**(전체 코드내용)**



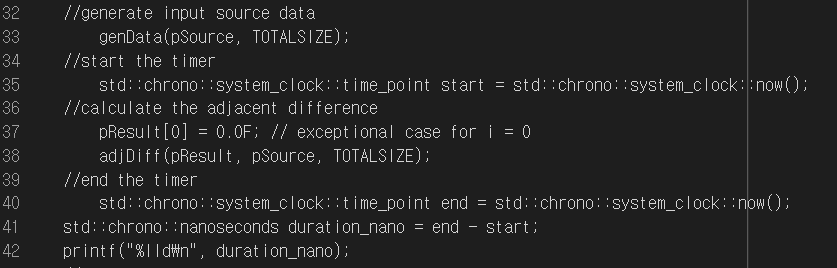


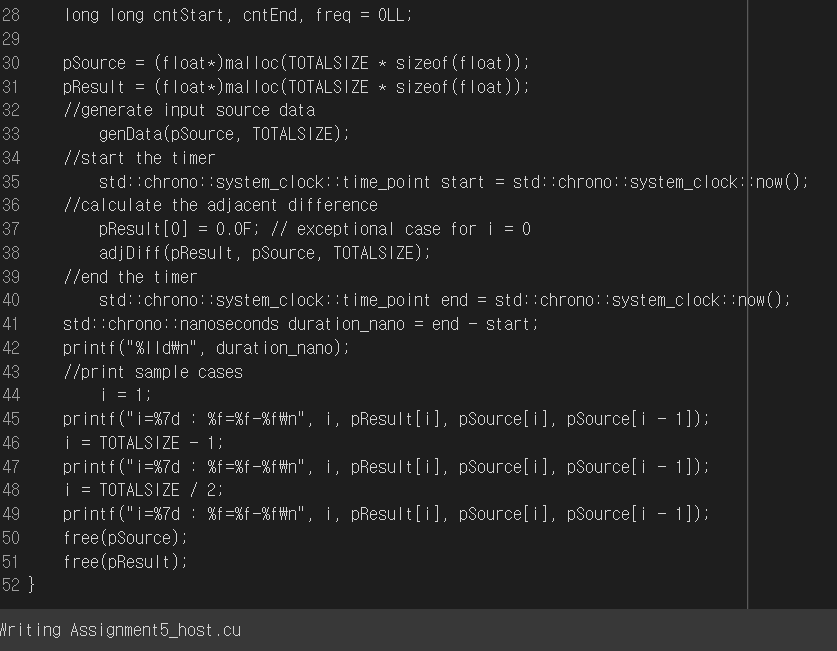
(부분 코드1)

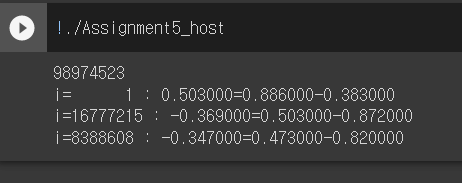


Host에서 adjacent difference function을 수행하고 그때의 시간을 측정하는데 이때 grid size = 16\*1024, block size = 1024로 설정합니다. 그리고 void genData(float \*ptr, unsigned int size)함수를 통해서 ptr로 받아온 값들을 rando한 수로 초기화 해줍니다. 또한 adjDiff(float\* dst, const float\* src, unsinged int size) function을 정의하여 이후 이 function 수행하기 전후의 시간을 측정하도록 합니다.

(부분 코드2, 결과화면)





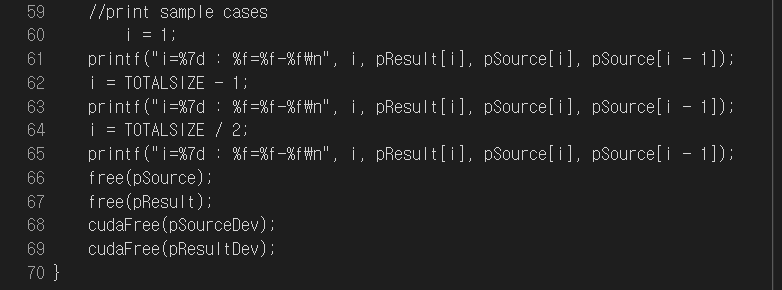
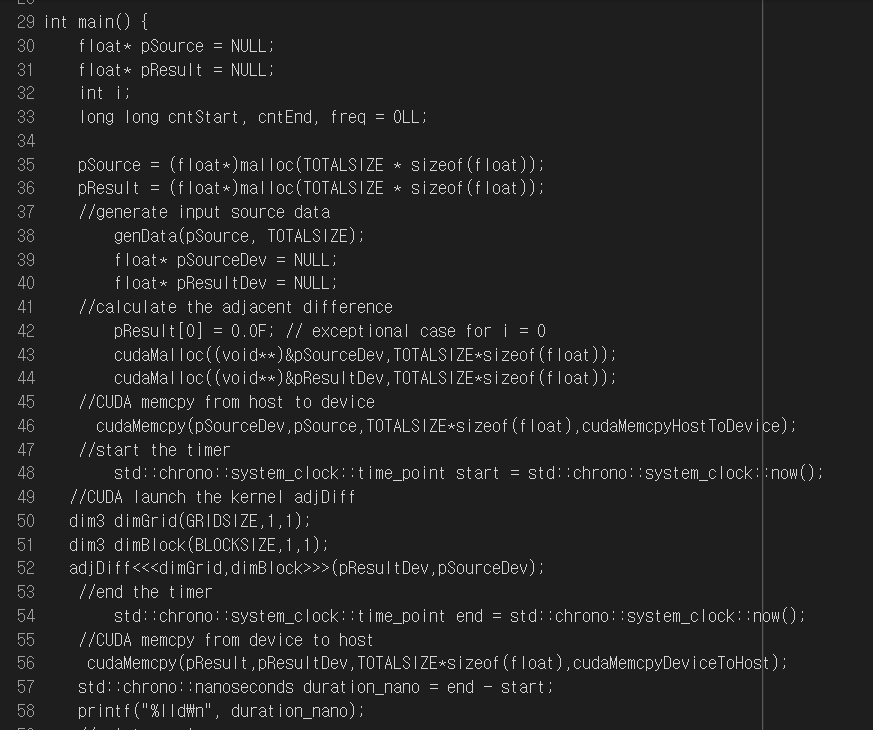
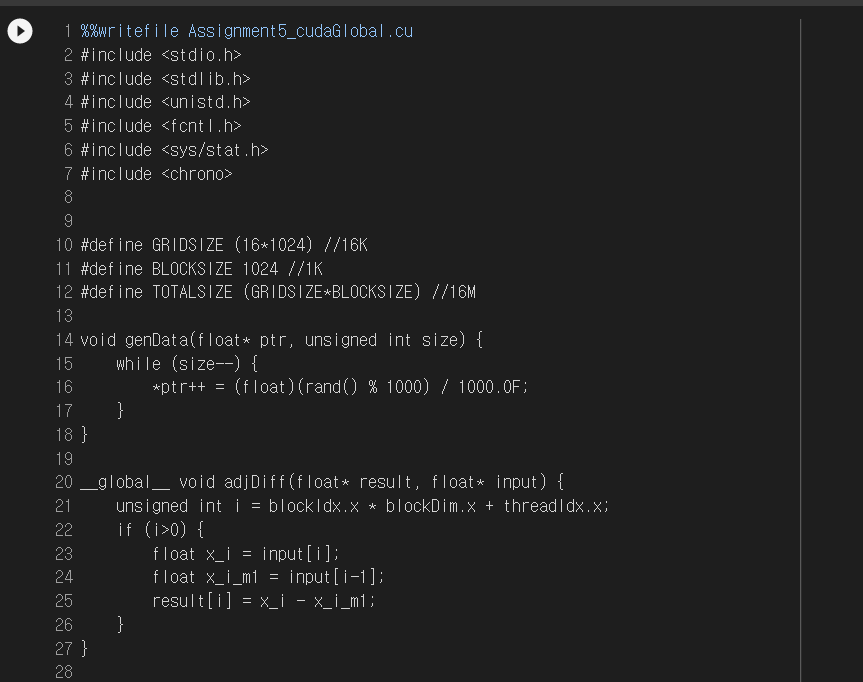


Std::chrono::..::time\_point start=…::now();, Std::chrono::..::time\_point end=…::now(); 를 통해 adjDiff함수의 진행 전 후 의 시각을 측정하고 결과로 std::chrono::nanoseconds duration\_nano = end-start; 를 통해 nanoseconds로 elapsed time을 계산하고 printf문으로 출력합니다.

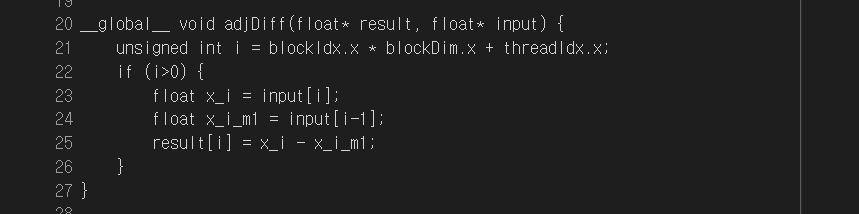
또한 소요된 시간의 결과\n이후 function에 의한 결과 값 몇 개를 출력합니다.

결과는 998974523nanoseconds 입니다.

**[Assignment5\_cudaGlobal.cu의 코드내용 및 결과화면]**

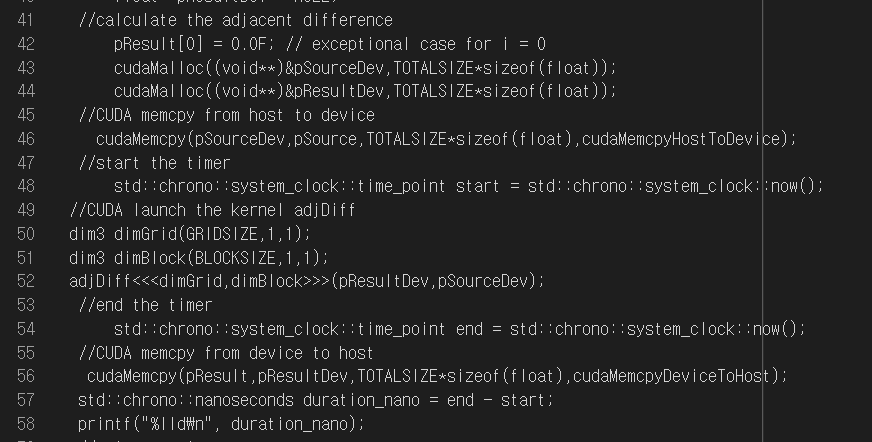


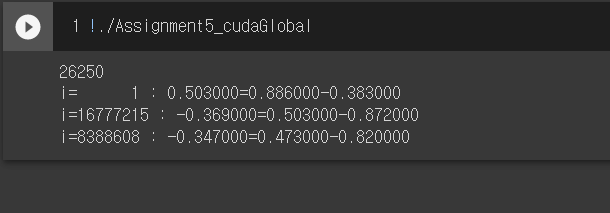
<부분코드1>



cudaGlobal.cu에서는 전의 host와 달리 GPU, kernel을 이용하여 위에서의 과정을 수행하므로 함수 또한 \_\_global\_\_ prefix를 추가 하고 인자의 값도 blockIdx.x, blockDim.x, threadId.x, 와 깉이 내장된 변수를 통해 선언하는 과정을 추가 합니다.

<부분코드2>





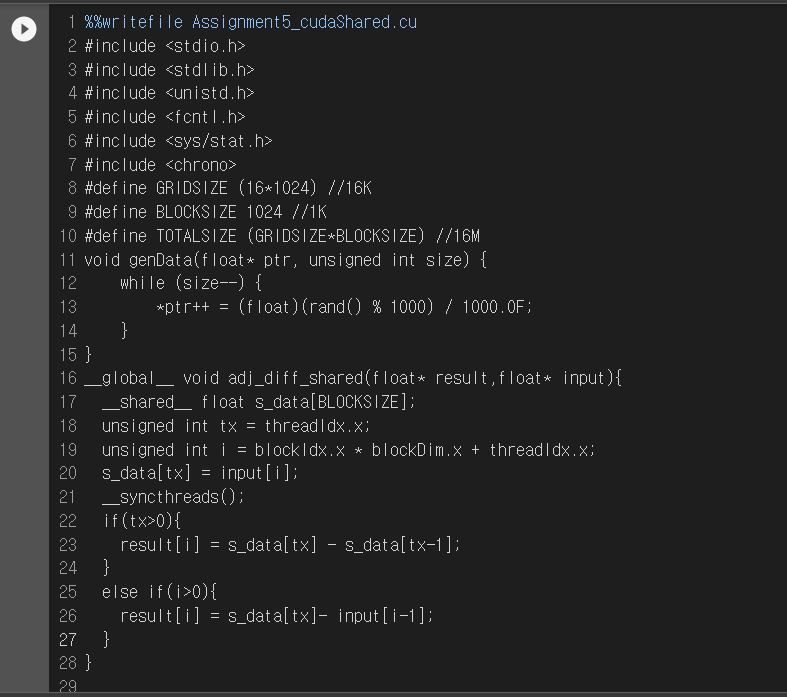
Host에서는 Malloc만 진행하였지만 cuadGlobal.cu에서는 추가로 cudaMalloc function을 이용하여 Kernel의 메모리를 할당합니다.

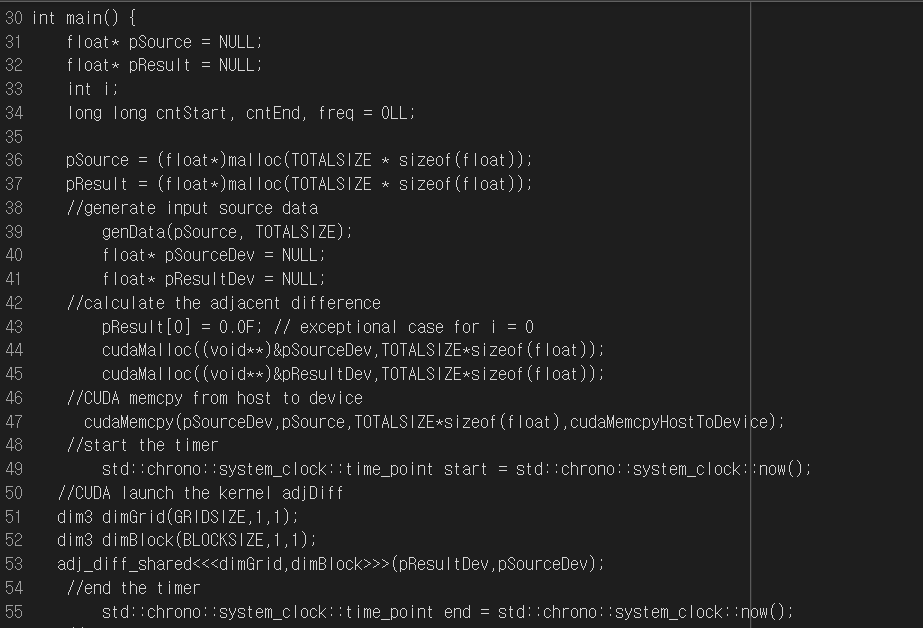
cudaMemcpy function을 이용하여 host의 data를 device로 copy합니다. Chrono 함수를 통해 strat 지점을 찍어준 뒤 function을 호출하기 위한 dimGrid, dim3Block을 선언해줍니다. 이후 앞서 정의 되었던 adjDiff function을 수헹한 뒤 chrono 함수의 end지점을 찍어 줍니다. 이후 function의 결과 값들을 다시 cudaMemcpy를 통해 Device에서 Host로 전달합니다.

이후 소요된 시간을 앞에서와 같이 print하여 확인합니다.

결과적으로 소요된 시간은 26250nanoseconds입니다. 이전의 host의 결과값에 998974523에 비해 약 1/38,056정도 인 것을 확인할 수 있습니다

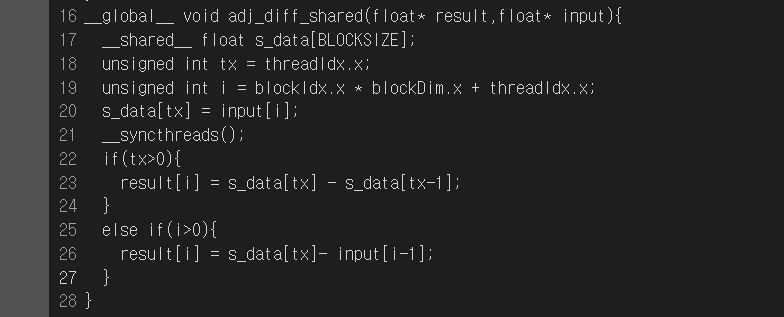
**[Assignment5\_cudaShared.cu의 코드내용 및 결과화면]**







<부분코드1>

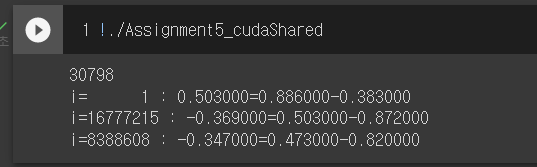


앞서서 진행하였던 cudaGlobal.cu의 예제에서의 \_\_global\_\_ void adj\_diff function에서 전달받은 인자의 값을 받아와서 sharedmemory로 load하고 이후에 그 sharedmemory에 있는 data에 접근하여서 adjacent differeremce 함수의 내용을 수행하는 과정을 거치게 됩니다. \_\_shared\_\_prefix를 통해 float형의 s\_data가 shared\_memory의 변수임을 선언하여 주고, unsigned int tx = threadIdx.x;를 통해 shared memory를 전달 받을 index변수도 선언 합니다.

또한 s\_data[tx]= input[i];를 통해서 globalmemory의 값을 sharedmemory로 전달 받고, \_\_syncthreads();를 통해 전체 threads가 모두 완료되기까지 기다립니다. 이후 s\_data의 값들로 if문 else if문의 조건에 따라 함수의 내용을 수행하고 result[i]에 저장합니다.

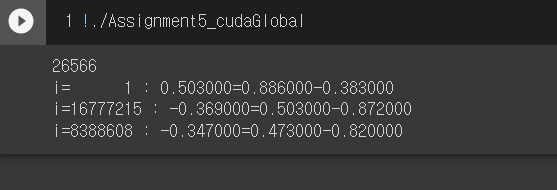
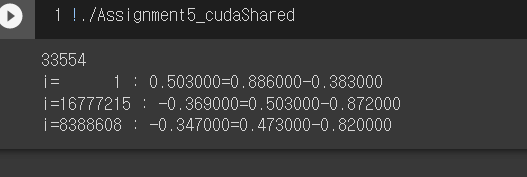
<부분코드2>



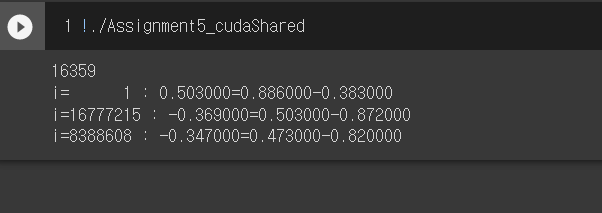
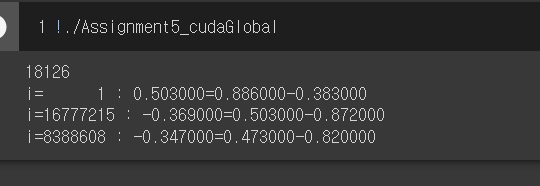


그외의 내용은 앞서서 했던 cudaGrobal.cu와 같이 cudaMalloc,cudaMemcpy,cudaFree와 같이 메모리할당, chrono힘수 start지점 찍기, HostToDevice로 Memcpy하여 function수행,

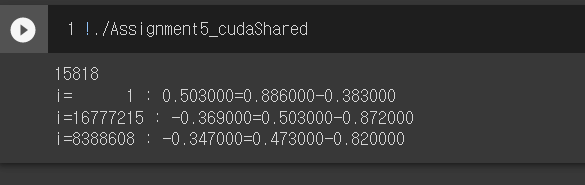
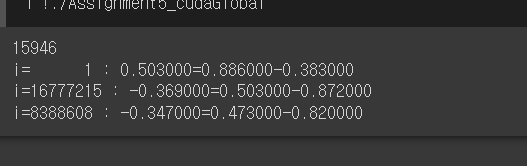
Chrono함수 end 지점 찍기, elapsed시간 nanosecond로 저장, DeviceToHost로 Memcpy하여 결과 출력, 메모리해제 순으로 진행합니다. 결과적으로 소요된 시간은 30798nanosecnond이며, 앞서서 진행한 cuadGlobal의 소요된 시간은 26250nanoseconds에 비해 약간 느리고, 이전의 host의 결과값에는 998974523에 비해 약 1/32,436정도 인 것을 확인할 수 있습니다

<추가>(cuadGlobal.cu 와 cudaShared.cu)반복 컴파일 후 결과 비교)

각각 첫번째 컴파일 후 실행하였을 때 결과 입니다. cudaGlobal.cu가 cudaShared.cu보다 미세하게 빠른 것을 확인 하였습니다.



각각 두번째 컴파일 후 실행하였을 때 결과 입니다. 앞선 결과와 다르게 cudaShared.cu가 cudaGlobal.cu보다 미세하게 빠른 것을 확인 하였습니다.



각각 세번째 컴파일 후 실행하였을 때 결과 입니다. 128nanoseconds정도의 매우 작은차이로cudaShared.cu가 cudaGlobal.cu보다 미세하게 빠른 것을 확인 하였습니다.

<결과 정리 및 고찰>

시간적인 성능으로는 cpu에 비해 gpu가 월등히 좋은 것을 확인하였습니다. 또한 이론적으로 배웠을 때 GPU에서도 globalmemory에서 위와 같은 instruction을 수행하는 것 보다 sharedmemory를 사용하는 것이 성능적으로 유리하지만 다루는 data의 사이즈, sharedmoemory로 옮기는데에 걸리는 overhead등의 영향으로 첫 컴파일애 대한 결과는 globalmemory가 더 좋게 나온 것 이지 않을까 생각 되었습니다. 또한 CPU에 비한 GPU의 시간측면에서의 성능을 직접적으로 수치로 비교하면서 차이를 체감 할 수 있게 된 것 같습니다.