**고급소프트웨어실습 12주차 과제**

**20182204 한찬희**

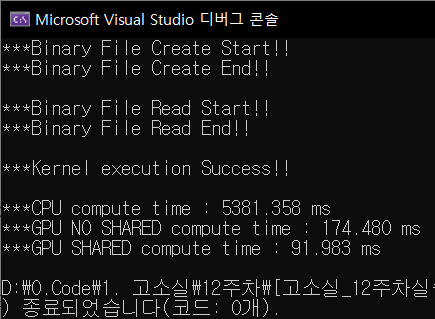
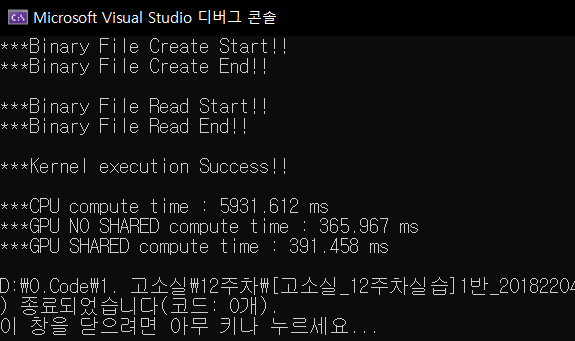
**실습 문제**

CPU, GPU(No shared, Shared memory)를 사용한 주어진 입력 데이터 파일을 읽어들여 수열 S를 계산한 후 출력 파일에 저장하는 프로그램이다.

**실험 결과 (Release 모드)**

Nf = 64에 대하여 블록 크기를 바꾼 경우

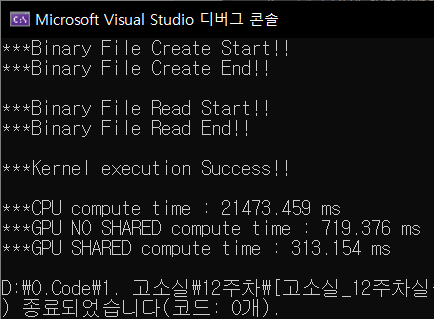
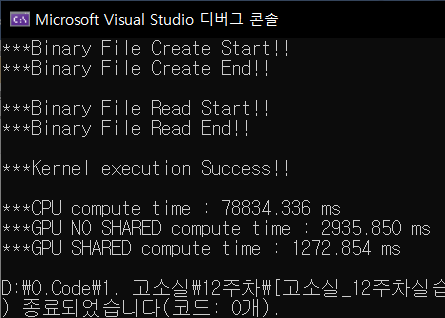
Nf: 64, BLOCK\_SIZE: 1024 / Nf: 64, BLOCK\_SIZE: 16

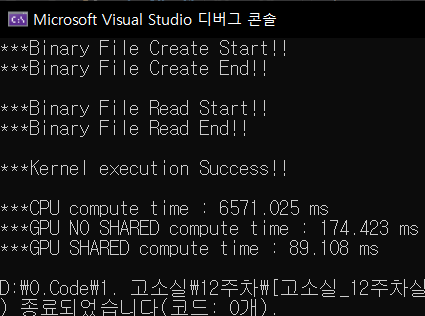
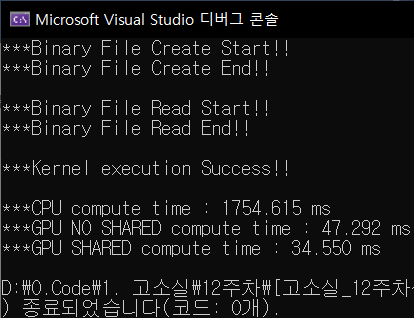
* 블록 사이즈가 클 때 GPU 수행 시간이 2배 가까이 줄어든 것을 확인할 수 있다. 블록 크기가 클수록 작업 분배량이 커지다 보니 GPU의 처리 속도가 빨라진 것이라고 생각한다. 또한, Shared memory를 사용했을 때가 사용하지 않았을 때보다 훨씬 빠른 것을 확인할 수 있다. Shared memory를 사용하게 되면 했던 계산을 또 할 필요없이 shared buffer로 불리는 변수를 이용하여 메모리 접근을 효율적으로 하기 때문에 훨씬 더 빨리 계산을 처리할 수 있게 된다.

블록 크기 256에 대해 Nf 값을 바꾼 경우

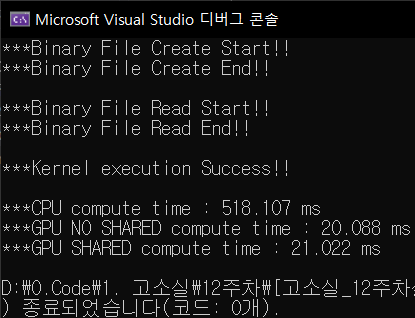
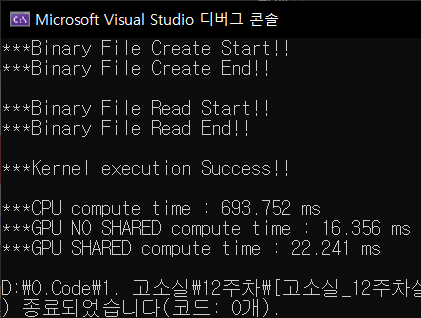
Nf: 256, BLOCK\_SIZE: 256 / Nf: 1024, BLOCK\_SIZE: 256

Nf: 64, BLOCK\_SIZE: 256 / Nf: 16, BLOCK\_SIZE: 256

Nf: 4, BLOCK\_SIZE: 256 / Nf: 1, BLOCK\_SIZE: 256

* Nf가 작아질수록 GPU 수행시간이 빨라진 것을 확인할 수 있다. Nf가 커질수록 합을 구하는 계산량도 늘어나기 때문에 처리하는 시간도 오래 걸리는 것으로 보인다.

**숙제**

CPU, GPU를 이용하여 입력받은 이미지를 Gaussian Filtering 처리하여 출력 이미지에 할당하는 프로그램이다.

**B1**

Do\_Gaussian\_on\_GPU() – cuda\_code.cu

* Shared memory를 사용하지 않고 가우시안 필터링을 수행하는 GPU 함수이다.
* 입력 이미지와 출력 이미지 배열을 위해 d\_bitmaps, d\_Gaussian을 device memory에 할당한다. 병렬 계산을 위해 1차원 그리드를 이용한다. BLOCK\_SIZE는 32로 지정하였다.
* Set\_Gaussian\_kernel() 함수를 호출해 가우시안 필 터링을 위한 커널을 생성한 후, Shared memory를 사용하지 않으므로 Gaussian\_kernel\_no\_shared() 함수를 호출한다. 이 함수에서는 커널이 이미지 바깥에 있는지 판별한 후 커널에 적용하여 mean에 더해준다. 이후 최종 mean은 해당 id의 출력이미지에 할당한다.

**B2**

Do\_Gaussian\_on\_GPU() – cuda\_code.cu

* Shared memory를 사용하므로 Gaussian\_kernel\_shared()를 호출한다. sharedBuffer의 사이즈는 BLOCK\_SIZE + 2\*w 만큼 할당한다.
* 이 함수에서는 중복 계산을 피하여 메모리 접근을 효율적으로 하기 위해 thread\_id가 0일 때와 BLOCK\_SIZE -1일 때로 나누어 준다. 각각의 유효 범위 내에 sharedBuffer 인덱스 자리에 d\_bitmaps 인덱스 자리를 할당한다. 유효 범위 내가 아니라면 sharedBuffer을 0으로 처리해 준다. 이후 sharedBuffer의 thread\_id+w 자리에 해당 줄 d\_bitmaps를 width만큼 할당해 준다. 이때 모든 스레드들이 그 지점에 도달할 때까지 기다리며 동기화해 주는 작업을 위해서 \_\_syncthread() 함수를 호출한다. 이후 sharedBuffer를 이용하여 커널에 적용하고 결과물을 d\_Gaussian에 할당한다.

**B3**

GaussianFilteringCPU() – FilteringView.cpp

* 입력 이미지를 불러온 후 m\_imageBits에 저장하고 가우시안 필터링을 위한 c\_gaussian\_kernel를 생성한다. 이후 커널이 이미지 내에 없으면 mean을 0으로, 커널이 이미지 내에 있으면 입력 이미지에 커널을 적용한 값을 mean에 넣어 준다. 이후 결과 이미지 배열 m\_imageBitsFiltered에 최종 mean을 넣는다.

**실행 결과 (Release모드)**

B1 B2 B3 (BLOCK\_SIZE = 2)

|  |  |
| --- | --- |
|  |  |
|  |  |

B1 B2 B3 (BLOCK\_SIZE = 8)

|  |  |
| --- | --- |
|  |  |
|  |  |

B1 B2 B3 (BLOCK\_SIZE = 32)

|  |  |
| --- | --- |
|  |  |
|  |  |

* 블락 크기가 작아질수록 GPU(shared, noshared 모두) 수행 시간이 늘어나는 것을 확인할 수 있다. 미묘한 차이지만 위 세 경우 모두 shared memory를 이용했을 때가 사용하지 않았을 때보다 시간이 빨랐고, 이것은 shared memory를 사용할 때 if 조건 비교하는 시간을 줄이기 위해 sharedBuffer에서 할당할 때 미리 처리해 줬으므로 메모리 접근을 효율적으로 하기 때문이다.