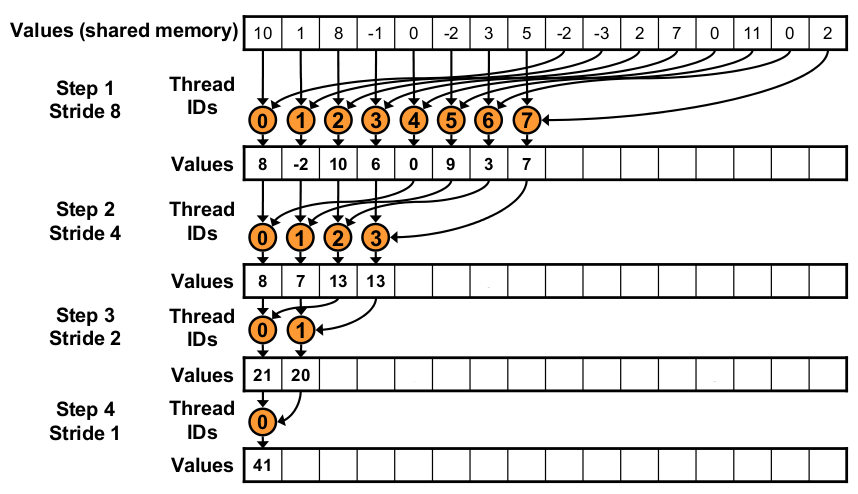
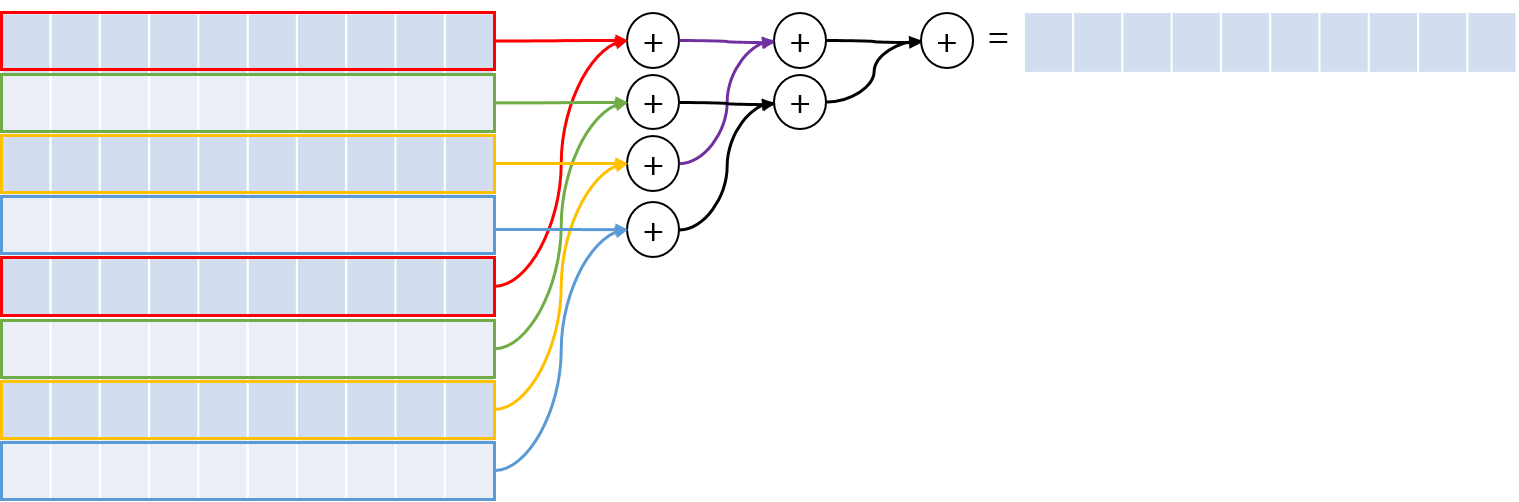
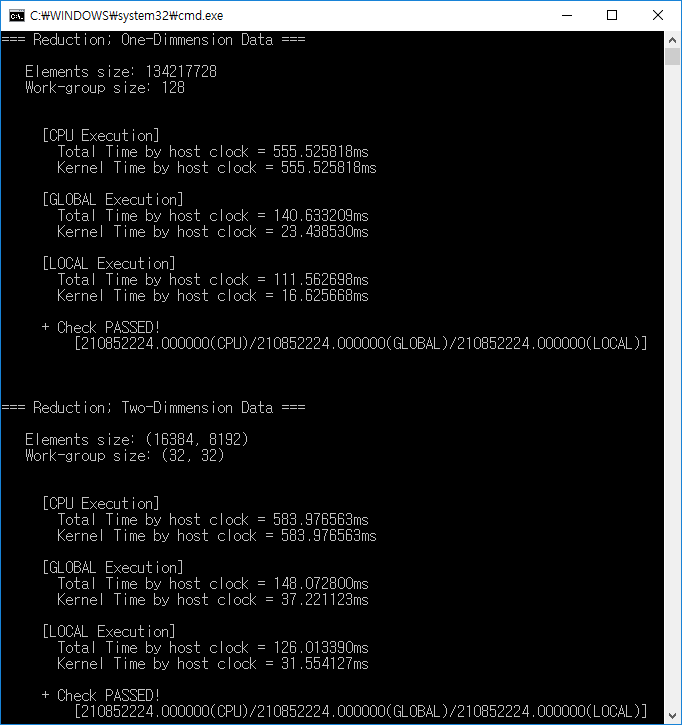
***수치 컴퓨팅 및 GPU 프로그래밍 – 숙제 1 (Ver. 1.0)***

***Reduction 문제를 통한 global/local memory의 접근 속도 비교***

120170420 신광수

* OpenCL 프로그램 수행 환경
  + NVIDIA GeForce GTX 1060 6GB
    - OpenCL 1.2 CUDA
    - MAX\_CU: 10
    - MAX\_WORK\_ITEM\_SIZES: 1024 / 1024 / 64
    - MAX\_WORK\_GROUP\_SIZE: 1024
    - MAX\_MEM\_ALLOC\_SIZE: 1536 MB
    - GLOBAL\_MEM\_SIZE: 6144 MB
    - LOCAL\_MEM\_SIZE: 68 KB
* 1차원 데이터의 Reduction 수행
  + 1차원 데이터에서 Reduction을 수행하는 방식을 다음과 같이 구성하였다.  
      
    값의 Index를 통해 반씩 접어 나가면서 덧셈을 진행하였다.
  + Global Memory의 경우, Global Memory에 담긴 데이터를 더한 후, 해당 Global Memory 위치에 그 값을 저장하였고, Local Memory의 경우, Global Memory에서 Local Memory로 값을 복사한 후, 덧셈을 진행하여 그 결과를 저장하였다.
  + 이렇게 구해진 Partial Sum은 CPU 코드를 활용하여 덧셈을 진행하였다.
  + 그 때의 CPU, Global Memory, Local Memory의 각각의 수행 속도는 다음과 같았다. 이 때, Work-Group 사이즈를 1부터 2의 배수로 늘려나가면서 실험을 진행하였다. 데이터 사이즈는 128 \* 1024 \* 1024로 설정하였고, 그 때 Random하게 값을 만들었다.
  + 수행 결과는 CPU 코드와 비교하여 float.h의 FLT\_EPSILON 값과 그 차이를 비교하여 값을 계산하였다. 그 때, 전반적으로 그 값이 오차 이내에 존재함을 알 수 있었다. 하지만 몇몇 Work-Group의 경우 오차가 벌어지는 경우가 존재하는데, 이는 float 합을 구하는데 있어 수치적 오류가 나는 것으로 이해가 가능하다.
  + Max Work-group Size가 1024였으므로, 그 이상 Work-Group 값을 설정하는 것은 불가능하였다.
* 2차원 데이터의 Reduction 수행
  + 2차원 데이터에서 Reduction을 수행하는 방식을 다음과 같이 구성하였다.  
    2차원 데이터를 행을 기준으로 반씩 접어서 열 마다 값을 더하면서 저장하였다.
  + 나머지 사항은 1차원 데이터의 Reduction 수행과 동일하게 진행하였다.
  + 그 때의 CPU, Global Memory, Local Memory의 각각의 수행 속도는 다음과 같았다. 이 때, Work-Group 사이즈를 1부터 2의 배수로 늘려나가면서 실험을 진행하였다. 데이터 사이즈는 128 \* 1024 \* 1024로 설정하였고, 그 때 Random하게 값을 만들었다.
  + Max Work-group Size가 1024였으므로, 32 \* 32 크기 이상의 Work-Group 값을 설정하는 것은 불가능하였다.
  + 위의 그래프와 다르게 Work-Group의 크기를 정사각형이 아닌, 직사각형의 모습으로 배열을 진행하였을 때의 결과는 다음과 같다.  
    위에서 2차원 행렬을 행을 기준으로 접었기 때문에, 계산은 열을 기준으로 이루어진다. 따라서 열의 크기를 키우면 키울수록 그 계산의 속도가 점차 빨라지는 것을 볼 수 있다. 따라서 같은 커널 사이즈라면, 가능한 열의 크기를 늘리는 것이 계산 결과를 빠르게 하는 방법이 될 것으로 보인다.
* Synchronization 관련 issue
  + 수행 환경은 NVIDIA GPU였으므로, Warp의 크기가 32이다. 따라서 한 번에 동시에 수행 가능한 thread의 수는 32번이다. 따라서 이에 맞추어 Work-group의 사이즈를 조절하고, OpenCL 코드에서 Synchronization 함수를 사용하지 않고 수행할 시의 결과를 확인하였다.
  + 1차원 데이터의 경우 Work-group을 32로 설정한 결과, 데이터의 덧셈의 결과는 동일하였다. 따라서 Warp와 Work-group을 맞춰준다면 그 값은 변하지 않음을 알 수 있다.
  + 2차원 데이터의 경우 Work-group을 (4, 8)으로 설정한 결과, 데이터의 덧셈의 결과는 동일하였다. 역시나 Warp와 Work-group을 맞춰준다면 그 값은 변하지 않음을 알 수 있다.
  + 따라서, GPU의 특성에 따라 NVIDIA의 Warp 또는 AMD의 Wavefront와 Work-group을 맞춘다면, Synchronization과 관련 없이 그 값이 크게 변하지 않을 것임을 알 수 있었다.
* 수행 결과
  + 결과에서 Total Time은 Host의 데이터를 버퍼에 담을 때부터 최종적으로 값을 낼 때까지 실행 시간을, Kernel Time은 Kernel 실행부터 최종적으로 값을 낼 때까지 실행 시간을 Host의 Clock으로 측정한 결과이다.
  + CPU로 Reduction 하는 코드는 수업시간에서 설명한 CPU “reduction\_on\_the\_cpu\_reduction” 함수를 사용하여 구성하였다.
  + 본 Readme 파일에서의 그래프 상 수행 속도는 Kernel Time을 기준으로 기술하였다.