**Assignment 5**

(GPU 컴퓨팅)

담당교수: 공영호 교수님

학과: 컴퓨터정보공학부

학년: 3

학번: 2018202074

이름: 김상우

#코드 작성

과제를 들어가기 앞서, 과제 조건에 대하여 살펴보았습니다.

* 해당 코드에선 Grid SIZE는 16\*1024, block size는 1024로 제한된다.
* Adjace difference에 대해 Host / CUDA Global memory / CUDA shared memory version에 대해 이를 진행한다..
* Elapsed time에 대해 Chrono library를 활용해 진행되어야 한다..

위의 조건을 만족하는 코드를 3개 작성하였다. 각 코드는 Host, CUDA Global memory, CUDA shared memory version에서 돌아가게 된다.

**-Host version**

텍스트이(가) 표시된 사진

자동 생성된 설명

Google colab에서 해당 코드를 Assignment5cuda.cu라는 코드로 인식하기 위해 위에 %%writefile Assignment5cuda.cu로 기술하였고 아래 필요한 헤더들을 기술하였다.

또한 측정을 위해 진행하는 함수를 HOST에서 명령하기 위해 function은 void에서 돌아가게 된다. 각 함수는 genData의 경우 rand함수를 이용 들어온 size 길이를 갖는 float 배열에 대해 1000범위에 대한 rand를 진행한 후 1000으로 나눠주는 동장을 모든 contents에 대해 진행하며, adjDiff의 경우 들어온 2개의 float 배열에 대해 진행되며 source로 받아오는 배열의 차를 다른 배열에 저장하는 계산을 진행한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후, main함수에 각각의 값을 받을 배열들을 선언 및 malloc을 통한 공간을 부여하고 genData를 이용해 값을 부여하였다. 이후 시간 측정을 위해 chrono를 이용, system clock내의 time point에 now함수를 이용해 시작시간을 기록했다. 이후 adjDiff를 실행해 시간을 주고 다시 chrono를 이용해 종료시간을 체크한 뒤, 시작시간에서 끝나는 시간을 빼어 전체 elapsed time을 duration\_micro에 저장하고 이를 출력한다. 끝으로 계산 결과를 출력하고 free를 통해 할당된 메모리를 해제하였다.

**-CUDA global memory version**

텍스트이(가) 표시된 사진

자동 생성된 설명

Google colab에서 해당 코드를 Assignment5-2cuda.cu라는 코드로 인식하기 위해 위에 %%writefile Assignment5-2cuda.cu로 기술하였고 아래 필요한 헤더들을 기술하였다.

차를 계산해 저장하는 adjDiff2 함수의 경우 CPU에서 GPU로 명령하기 위해 \_\_global\_\_을 추가하였다. 이후 i에 각 thread에 해당 하는 값들을 넣고 이를 기반으로 array의 차를 계산 후 저장하는 것을 확인할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후, main함수에 각각의 값을 받을 배열들을 선언 및 malloc을 통한 공간을 부여하고 genData를 이용해 값을 부여하였다. 이 배열들에 대해서 Device(GPU)에서 사용하기 위해 추가 배열들을 선언, 할당(cudaMalloc)을 진행하는 것을 확인했다. 이후 cudaMemcpy를 이용해 CPU의 배열을 GPU에서 사용될 변수에 복사하고 시간 측정을 위해 chrono를 이용, system clock내의 time point에 now함수를 이용해 시작시간을 기록했다. 이후 구조에 대한 선언을 위해 dimGrid2(GRIDSIZE, 1, 1)와 dimBlock2(BlockSIZE, 1, 1)을 준비하고 adjDiff2에 해당 구조와 앞서 언급된 배열들을 넣어 실행한다. 해당 작업이 종료되면 cudaMemcpy를 통해 GPU의 계산값을 CPU로 받아온다. 이후 chrono를 이용해 종료시간을 체크한 뒤, 시작시간에서 끝나는 시간을 빼어 전체 elapsed time을 duration\_micro2에 저장하고 이를 출력한다. 끝으로 계산 결과를 출력하고 free와 cudaFree를 통해 CPU와 GPU에 할당된 메모리를 해제하였다. 최종적으로 cudaDeviceSynchronize()를 통해 동기화를 진행했다.

**-CUDA shared memory version**

텍스트이(가) 표시된 사진

자동 생성된 설명

차를 계산해 저장하는 adjDiff3 함수의 경우 CPU에서 GPU로 명령하기 위해 \_\_global\_\_을 추가하였다. 이때 함수 내에서 처리하는 데이터 배열에 대하여 shared memory를 사용하기 위해 \_\_shared\_\_로 s\_data 배열을 선언한다. 이후 i와 tx에 각 thread에 해당 하는 값들을 넣고 이를 기반으로 array의 차를 계산 후 저장하는 것을 확인할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후, main함수에 각각의 값을 받을 배열들을 선언 및 malloc을 통한 공간을 부여하고 genData를 이용해 값을 부여하였다. 이 배열들에 대해서 Device(GPU)에서 사용하기 위해 추가 배열들을 선언, 할당(cudaMalloc)을 진행하는 것을 확인했다. 이후 cudaMemcpy를 이용해 CPU의 배열을 GPU에서 사용될 변수에 복사하고 시간 측정을 위해 chrono를 이용, system clock내의 time point에 now함수를 이용해 시작시간을 기록했다. 이후 구조에 대한 선언을 위해 dimGrid3(GRIDSIZE, 1, 1)와 dimBlock3(BlockSIZE, 1, 1)을 준비하고 adjDiff3에 해당 구조와 앞서 언급된 배열들을 넣어 실행한다. 해당 작업이 종료되면 cudaMemcpy를 통해 GPU의 계산값을 CPU로 받아온다. 이후 chrono를 이용해 종료시간을 체크한 뒤, 시작시간에서 끝나는 시간을 빼어 전체 elapsed time을 duration\_micro3에 저장하고 이를 출력한다. 끝으로 계산 결과를 출력하고 free와 cudaFree를 통해 CPU와 GPU에 할당된 메모리를 해제하였다.cudaDeviceSynchronization()을 통해 동기화하며 코드작성을 완료했다.

**-결과**

텍스트이(가) 표시된 사진

자동 생성된 설명

!nvcc -o를 통해 실행파일로의 변환을 시도해 성공한 것을 확인했다.텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명

-HOST

-CUDA GLOBAL Memory

-CUDA SHARED Memory

위의 결과를 보면 CPU에서 연속적으로 처리하는 HOST Version이 가장 오래 걸리는 것을 확인할 수 있었다. 평행하게 처리하나 GLOBAL Memory를 사용하여 SHARED Memory를 사용할 때보다는 느린 CUDA GLOBAL Memory가 그 뒤를 이었으며, GPU를 이용해 평행하게 처리함과 동시에 SHARED Memory를 사용한 CUDA SHARED Memory의 경우가 가장 적은 Elapsed time을 보였다.

**#고찰**

GPU 컴퓨팅 Assignment5을 진행하면서 google colab의 환경에서 데이터에 따라 접근하는 Memory의 종류를 바꿔가며 테스트를 진행하였습니다. 이때 CPU와 GPU의 차이 또한 비교하며 이전 assignment에서 배운 CPU와 GPU에 차이에 따른 구조적 연산시간 차이를 복습할 수 있었습니다. 이번 과제에서는 Shared memory와 global memory에 대한 접근을 \_\_shared\_\_의 유무에 따라 chrono 헤더를 통한 시간 측정으로 비교하며 data접근 시 사용하는 memory의 종류가 얼마나 시간에 작용하는 지 알 수 있었습니다.