**고급소프트웨어실습1**

**11주차 과제**

**20161663 허재성**

**실습 2 결과**

실습 2에서는 220(1,048,576)개의 float 원소를 가진 두 배열 A, B에 저장된 값을 이용하여 값을 계산 후 C에 저장하는 프로그램을 작성하였다. 이 때 CPU로 계산을 수행하는 프로그램과 CUDA 프로그래밍으로 GPU로 계산하는 프로그램을 작성하여 실행 시간을 비교한다. CUDA 프로그래밍의 경우 BLOCK\_SIZE를 다양하게 하여 BLOCK\_SIZE에 따른 계산 속도를 비교한다. 이 때 BLOCK\_SIZE는 warp 크기인 32(=25)의 배수로 설정한다. (추가적`으로 warp 크기보다 작은 크기로도 설정해서 비교해본다.)

한 thread block에는 최대 1024(=210)개의 thread만 존재 가능하므로 그 이상으로 BLOCK\_SIZE를 늘리는 것은 의미가 없다.

|  |  |  |
| --- | --- | --- |
| **BLOCK\_SIZE** | **CPU 계산 결과(ms)** | **GPU 계산 결과(ms)** |
| **24** | 32.741199 | 0.247808 |
| **25** | 32.699699 | 0.153984 |
| **26** | 33.617699 | 0.089952 |
| **27** | 32.928902 | 0.092608 |
| **28** | 34.057201 | 0.090048 |
| **29** | 34.565201 | 0.090656 |
| **210** | 34.379101 | 0.096000 |

Block의 행 크기와 열 크기를 조절하여 각각 23일 때, 즉 BLOCK\_SIZE가 26일 때 GPU 계산 속도가 가장 빨랐다. 그 이후로는 block 크기를 더 크게 해도 유의미한 속도 증가를 보이지 못했다.

GPU 계산 시간이 가장 빠른 BLOCK\_SIZE인 26으로 설정하여 계산 시간을 비교해보 GPU 계산이 약 374배 더 빠른 것을 알 수 있다.

**실습 3 결과**

32행 32열 행렬 A와 32행 n열 행렬 X를 곱한 32행 n열 행렬 Y를 구하는 프로그램이다. 대략적으로 322n의 반복 연산이 필요하므로 n의 크기를 1024(=210)으로 설정하여 계산했다.

|  |  |  |
| --- | --- | --- |
| **BLOCK\_SIZE** | **CPU 계산 결과(ms)** | **GPU 계산 결과(ms)** |
| **24** | 0.639400 | 0.080992 |
| **25** | 0.652500 | 0.079456 |
| **26** | 0.639500 | 0.102176 |
| **27** | 0.650800 | 0.104160 |
| **28** | 0.64100 | 0.198656 |

BLOCK\_SIZE가 warp와 같은 25일 때 가장 빠른 속도로 계산되었다. 28에서 BLOCK\_SIZE를 2배씩 증가시켜 실험한 경우 계산이 정상적으로 수행되지 않아 생략하였다. GPU 수행시간이 가장 짧은 BLOCK\_SIZE인 25으로 설정하여 계산 시간을 비교해보 GPU 계산이 약 8배 더 빠른 것을 알 수 있다.

**숙제 1 결과**

N\_EQUATIONS개의 실수 계수 2차 방정식을 생성하여 근의 공식을 이용하여 N\_EQUATIONS 쌍의 근을 구하는 프로그램이다. 편의 상 실근이 2개 존재하는 경우의 방정식만을 생성하며 계산의 편의를 위해 모든 실수 계수는 [-100, 100]의 값을 가지도록 하였다. 방정식의 계수들이 저장될 binary file A.bin, B.bin, C.bin을 생성하기 위해 init\_bin\_file 함수를 작성하였다. 이 때 하나의 방정식의 계수를 정할 때 b2-4ac > 0이 되어 서로 다른 두 근이 존재하는 경우에 방정식이 완성되도록 하였다. CPU와 GPU로 각각 근을 구하고 CPU로 구한 두 근 x0, x1(x0 < x1)을 각각 배열 h\_X0, h\_X1에 저장, 그 근을 해당 방정식에 대입한 값을 각각 배열 h\_FX0, h\_FX1에 저장하였고 마찬가지로 GPU로 구한 두 근을 각각 배열 d\_X0\_GPU, d\_X1\_GPU에, 근을 대입한 값을 d\_FX0\_GPU, d\_FX1\_GPU에 저장하였다. 이 때 CPU로 구한 근과 GPU로 구한 근의 차이를 확인하여 그 차이가 EPSILON 이하일 경우 두 방법이 같은 근을 구했다고 판단한다.

N\_EQUATIONS는 226으로 설정하였다.

|  |  |  |
| --- | --- | --- |
| **BLOCK\_SIZE** | **CPU 계산 결과(ms)** | **GPU 계산 결과(ms)** |
| **24** | 328.153 | 13.076 |
| **25** | 349.8760 | 8.022 |
| **26** | 343.236 | 7.974 |
| **27** | 329.345 | 8.595 |
| **28** | 326.361 | 8.511 |
| **29** | 325.921 | 8.356 |

Block의 행 크기와 열 크기를 조절하여 각각 23일 때, 즉 BLOCK\_SIZE가 26일 때 GPU 계산 속도가 가장 빨랐다. 그 이후로는 block 크기를 더 크게 해도 미세하지만 오히려 속도가 느려졌다.

GPU 계산 시간이 가장 빠른 BLOCK\_SIZE인 26으로 설정하여 계산 시간을 비교해보 GPU 계산이 약 44배 더 빠른 것을 알 수 있다. 반대로 BLOCK\_SIZE를 warp 크기보다 작은 24로 했을 경우 warp 크기만큼 설정했을 때보다 확실히 느린 것을 알 수 있다.

**숙제 2 결과**

DATA\_SIZE 개의 0~40의 정수를 임의로 생성하여 해당하는 Fibonacci 수를 구하는 프로그램을 작성한다. 이 때 Fibonacci 수는 잘 알려진 공식을 이용하여 계산한다. 계산 후 CPU로 계산한 값과 GPU로 계산한 값이 맞는지 확인하는 과정을 거친다.

구하려는 Fibonacci 수의 개수 DATA\_SIZE는 226이다.

|  |  |  |
| --- | --- | --- |
| **BLOCK\_SIZE** | **CPU 계산 결과(ms)** | **GPU 계산 결과(ms)** |
| **24** | 1038.360840 | 12.425952 |
| **25** | 1042.018311 | 6.799136 |
| **26** | 1059.309448 | 4.510048 |
| **27** | 1044.344238 | 4.610240 |
| **28** | 1040.404053 | 4.796704 |
| **29** | 1048.723145 | 5.567808 |
| **210** | 1061.071045 | 6.584192 |

Block의 행 크기와 열 크기를 조절하여 각각 23일 때, 즉 BLOCK\_SIZE가 26일 때 GPU 계산 속도가 가장 빨랐다. 그 이후로는 block 크기를 더 크게 해도 미세하지만 오히려 속도가 느려지는 것을 알 수 있다.

GPU 계산 시간이 가장 빠른 BLOCK\_SIZE인 26으로 설정하여 계산 시간을 비교해보 GPU 계산이 약 235배 더 빠른 것을 알 수 있다. 반대로 BLOCK\_SIZE를 warp 크기보다 작은 24로 했을 경우 warp 크기만큼 설정했을 때보다 확실히 느린 것을 알 수 있다.

**숙제 3 결과**

N\_SIZE 개의 -100~100의 정수를 임의로 생성하여 배열에 저장 후 배열의 각 원소 별로 해당 원소와 해당 원소 앞뒤로 Nf개의 원소의 합을 새로운 배열의 해당 원소의 인덱스에 저장하는 프로그램이다. N\_SIZE는 226으로 설정했고 Nf는 64(=26)으로 설정하였다. 마찬가지로 CPU와 GPU를 이용해 계산하며 두 방법으로 계산한 결과가 일치하는 지 확인한다.

|  |  |  |
| --- | --- | --- |
| **BLOCK\_SIZE** | **CPU 계산 결과(ms)** | **GPU 계산 결과(ms)** |
| **24** | 837.697 | 106.366 |
| **25** | 854.130 | 56.913 |
| **26** | 841.140 | 53.961 |
| **27** | 838.822 | 36.754 |
| **28** | 836.212 | 36.345 |
| **29** | 838.167 | 26.548 |
| **210** | 835.730 | 22.495 |

이전 실험들과 달리 warp 크기 또는 그 두배에서 가장 빠른 속도를 보이는 것이 아닌 BLOCK\_SIZE가 더 커질수록 GPU 수행 속도가 더 빨라졌다. 위에서 언급한 것처럼 BLOCK\_SIZE는 최대 210까지 가능하므로 211 로 크기를 증가시킬 경우 kernel code가 제대로 작동하지 않는다.