**병렬 프로그래밍 – Assignment 3**

컴퓨터공학과 20151619 최준수

**#1**

|  |  |  |
| --- | --- | --- |
| # of threads | Execution time | Speed up |
| 1 | 51.904162 seconds | 1 |
| 2 | 29.936291 seconds | 1.73 |
| 3 | 21.072244 seconds | 2.47 |
| 4 | 16.799190 seconds | 3.1 |
| 8 | 8.869441 seconds | 5.89 |

처음에 dictionary를 읽으면서 vector에 저장을 하고 앞뒤가 같은 palindrome인지 확인을 한 후, 아니라면 vector를 linear하게 접근하며 반대로 뒤집은 단어가 vector에 있는지를 확인하는 방식으로 진행하였다. 따라서 palindrome인지를 확인하는 과정에서는 모든 thread들이 병렬적으로 수행될 수 있지만. palindrome인 경우, output file에 기록을 하여야 하는데, 이때 하나의 file pointer를 모든 thread들이 공유하기 때문에 #pragma omp critical을 사용하여 race condition을 방지해주었다. #pragma omp critical에서 여러 thread들이 동시에 파일에 기록하려한다면 잠깐씩 대기하는 시간이 있기 때문에 speed up이 # of thread와 비례하기 증가하지는 못한 것 같다. 하지만 대부분의 경우에 speed up이 linear하게 # of threads만큼 나오는 경우는 잘 없다는 것을 가정한다면 이정도 speed up은 꽤나 좋은 결과를 보여준 것 같다.

**#2**

(1)

|  |  |
| --- | --- |
| BLOCK SIZE | Execution Time |
| 8 | 14.485984 |
| 32 | 3.834112 |
| 64 | 3.830656 |
| 128 | 3.790304 |

여러 Block size를 테스트 해본결과 32와 64일 경우에는 거의 비슷한 속도로 수행하였고, 128일 때 미세하지만 조금 더 나은 성능을 보여주었다.

(2)

|  |  |
| --- | --- |
| Global Memory | 3.830656 |
| Shared Memory | 664.487976 |

Shared memory는 matrix를 더 작은 Tile로 쪼개어 각 block들이 자신의 tile을 계산하는 방식으로 계산하였는데, \_\_synchthreads() 때문인지 Global memory를 사용한 것에 비해 훨씬 더 긴 시간이 걸렸다. 하지만 이론상으로는 off-chip memory인 global memory보다 block끼리 공유하는 on-chip memory인 shared memory를 사용하는 것이 더 좋은 성능을 보여준다.

**#3**

(d)

|  |  |  |  |
| --- | --- | --- | --- |
| block size | GPU | GPU no divergence | GPU shared no divergence |
| 32 | 0.080576 | 0.049984 | **0.030624** |
| 64 | 0.175744 | 0.156128 | **0.076608** |
| 128 | 0.112544 | 0.053248 | **0.046976** |

(with array size of 10000)

Reduction을 사용하여 Array에서 max값을 찾는 함수를 처음에 Global로 path divergence를 염두에 두지 않고 작성하였다. 그 뒤 path divergence를 해결한 함수를 작성하였고, 추가적으로 global memory가 아닌 shared memory를 사용하는 함수를 추가했다. 그 뒤 block size를 변경해가며 실험을 진행하였는데, 모든 block size에 대하여 path divergence없이 shared memory를 사용한 경우가 가장 성능이 좋았다. 그리고 그 중에서 32의 block size를 가지는 경우가 가장 성능이 좋았다.

(1)

|  |  |
| --- | --- |
| CPU | 0.219000 |
| GPU | 0.090432 |
| GPU no divergence | 0.088192 |
| GPU shared no divergence | **0.035168** |

CUDA로 GPU를 사용하지 않고 오로지 CPU에서 모든 연산을 처리하는 [CPU]가 가장 성능이 좋지 않았고, path divergence를 염두에 두지 않고 global memory에서 값을 불러와 수행하는 [GPU]가 그 다음으로 성능이 좋았다. 그리고 path divergence를 해결한 결과인 [GPU no divergence]가 근소하게 더 나은 성능을 보여줬으며, 추가적으로 shared memory를 사용한 [GPU shared no divergence]가 가장 뛰어난 성능을 보여주었다.

(2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| data size | CPU | GPU | GPU no divergence | GPU shared no divergence |
| 1000 | 0.005900 | 0.059136 | 0.035104 | **0.031584** |
| 10000 | 0.219000 | 0.090432 | 0.088192 | **0.035168** |
| 100000 | 0.594600 | 0.175744 | 0.156128 | **0.076608** |

data size를 1000, 10000, 10000으로 바꿔가며 실험한 결과이다. 앞에서 언급한 것처럼 path divergence를 해결하고 shared memory를 사용한 함수가 가장 뛰어난 성능을 보여주었으며, 신기하게도 GPU를 사용한 코드들은 array size를 10,000에서 100,000으로 올렸을 때 대략적으로 2배정도 수행시간이 늘어났다면 CPU는 3배가량 늘어나며 data size가 증가함에 있어서 가장 큰 변화폭을 보여주었다.