Parallel/Distributed Computing (CSEG414/CSE5414) Assignment #2

120220172 박규리

**1.**

|  |  |
| --- | --- |
| cores | times |
| 1 | 2.940025 |
| 2 | 1.514977 |
| 4 | 0.911928 |
| 8 | 0.513713 |
| 16 | 0.458025 |
| 32 | 0.397498 |
| 64 | 0.398804 |

Palindrome의 실행 결과는 다음과 같다.

표와 그래프에서 확인할 수 있듯이, 사용하는 core의 개수를 1부터 64까지 늘어남에 따라 실행 속도가 점점 빨라지는 것을 볼 수 있다. 그러나 코어의 개수가 32개가 넘어가는 순간부터는 32개나 64개나 성능 차이가 없는 것을 알 수 있다. 이는 아무리 사용하는 코어의 개수를 늘리더라도 sequential 하게 돌아가는 구간 때문에 일정 수행 속도보다는 더 빨라질 수 없는 것으로 보인다.

실제로, 프로그램에서 sequential 하게 수행되는 구간은 다음과 같다.

- Input file의 크기 즉, 단어의 개수를 구하는 구간으로 만약 단어의 개수가 N개라면 N번 만큼의 loop를 수행하게 된다.

- 각 단어가 palindrome인지 확인하기 위해서는 N 만큼의 loop를 각 thread 내에서 돌아야 한다.

Parallel 하게 수행되는 구간은 다음과 같다.

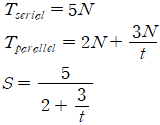
-단어의 개수를 읽는 N번만큼의 loop

-단어마다 palindrome인지 확인하는 함수를 호출하는 N번만큼의 loop

-단어마다 할당해줬던 메모리를 해제해주는 N번 만큼의 loop

따라서, sequential 한 구간은 2N, parallel 한 구간은 3N/t (t는 thread의 개수)가 될 것이다.

Speedup을 구해보면,

 가 성립되고, 이에 대한 값을 표로 나타내면 아래와 같다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Threads | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| Speedup | 1.0 | 1.43 | 1.81 | 2.11 | 2.29 | 2.39 | 2.44 |

표를 보았을 때, sequential 하게 돌아가는 구간 (2N)에 의하여 코어의 개수를 무한대로 늘려도 2N만큼 loop를 수행하게 되며, speeduo은 5/2 즉, 2.5를 넘을 수 없게 된다.

**2.**

(1)

1) **Naïve Prefix Sum Algorithm**

첫 번째로 가장 naïve한 알고리즘은 다음과 같다. Node N(1 ~ N)개가 있다고 가정할 때, Node 1에서 node2로 data를 보낸다. Node 2는 Node 1에게 데이터를 받은 다음, prefix sum을 계산해 Node 3로 해당 prefix sum을 보낸다. Node3는 Node2에게 Prefix sum을 받아 해당 prefix sum을 계산한 후, Node 4로 prefix sum을 보낸다. 이렇게 1부터 N까지 prefix sum이 차례대로 Propagation 된다고 보면 된다.

2) **More Parallel Prefix Sum Algorithm**

각 Level에 대해 서로 다른 offset의 node들에게 prefix sum을 병렬적으로 전송하는 것이다. Level 1에선 Rank 차이만큼 전송하고, Level 2에선 Rank 차이만큼 전송하고, Level 3에선 차이만큼 전송하는 꼴이다. 위와 같이, Parallel Algorithm을 구현하면 장점은 각 Level 별로는 Parallel하게 Data를 계산하므로 Parallelism을 최대한 활용할 수 있다.

(2)

Scan, blocking, non-blocking 에 대한 구현은 두번째 알고리즘을 채택하였다.

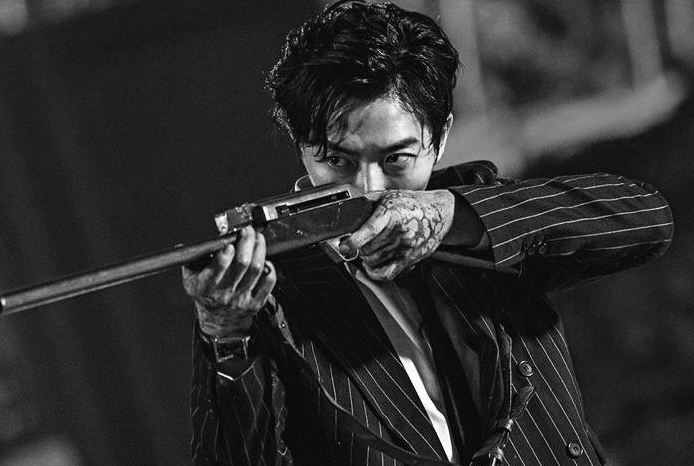
|  |  |  |  |
| --- | --- | --- | --- |
| Node | scan | blocking | Non-blocking |
| 1 | 0.000019 | 0.000025 | 0.000024 |
| 2 | 0.000083 | 0.000088 | 0.000058 |
| 4 | 0.000109 | 0.000238 | 0.000112 |
| 8 | 0.000279 | 0.000103 | 0.000182 |
| 16 | 0.000471 | 0.001368 | 0.000427 |
| 32 | 0.059628 | 0.035739 | 0.012293 |

프로세스의 개수를 1에서 32까지 늘려가면서 실험을 진행하였다. 세가지 방식 모두 노드가 많아질수록 실행 시간이 늘어나는 것을 볼 수 있다. Scan의 경우 전반적으로 MPI에서 기본적으로 제공하는 Collective Operation인 Scan을 이용한 구현은 node의 수가 많아지면 많아질수록 느려지는 것을 볼 수 있는데 이는 아마도 parallel algorithm의 구현에서 온 차이로 보인다. Parallel하게 구현한 prefix sum은 레벨 별로 병렬적으로 돌기 때문에 같은 레벨 내에서는 다른 노드의 연산의 결과를 기다릴 필요하 없는 병렬화를 최대로 사용한 알고리즘이다. non-blocking의 경우 blocking 보다 더 빠른 속도를 내는 것을 볼 수 있는데, 이는 blocking의 경우 다른 노드로 send 및 recv 하는 과정에서 연산이 블락되지만, non-blocking의 경우는 블락되는 delay가 줄어들기 때문으로 보인다.

3.

좌측의 경우 66,887KB로 굉장히 큰 사이즈의 이미지이고, 우측은 969KB로 비교적 작은 사이즈의 이미지이다. 이 두 사진을 가지고 실험을 진행하였다.

Grayscale을 진행한 결과이다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sequential | 2 | 4 | 6 | 8 |
| 969KB | 0.065181 | 0.075859 | 0.085443 | 0.097791 | 0.092882 |
| 66,887KB | 3.762940 | 4.015464 | 4.352579 | 4.356053 | 5.145183 |

flip

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sequential | 2 | 4 | 6 | 8 |
| 969KB | 0.062790 | 0.069623 | 0.088648 | 0.091157 | 0.123153 |
| 66,887KB | 3.311458 | 4.061312 | 4.210455 | 4.627316 | 4.551210 |

smooth

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sequential | 2 | 4 | 6 | 8 |
| 969KB | 0.101996 | 0.121536 | 0.115136 | 0.096354 | 0.119056 |
| 66,887KB | 5.400082 | 5.428732 | 4.951066 | 4.839704 | 5.183709 |

대체로 flip과 grayscale를 진행할 때 process를 하나만 생성하는 serial program이 가장 높은 속도를 보였다. Gray의 경우 코어의 개수가 늘어날수록 오히려 수행 시간이 더 오래 걸렸다. 코어를 여러 개 사용하였을 때 효율이 좋아지는 work는 smooth뿐인 것으로 드러났으며, 수행 시간 감소폭을 봐서는 코어의 개수를 8개 이상으로 늘리면 더 좋은 효율을 보일 것으로 예상된다. 따라서, smooth에 대해서는 weakly scalable하다고 볼 수 있다. 실험을 더 진행해 봐야 확실한 결과를 도출할 수 있을 것이라고 예상된다.