멀티코어 HW5 보고서

2016-11464 정재훈

1. 서론

본 보고서에서는 OpenMP을 통하여 다수의 thread를 사용하여 thread들이 parallel하게 for loop를 처리하도록 하여 work를 배분하고, MPI를 통하여 다수의 node를 사용하여 node들에 work를 배분한 후 node간의 통신을 통하여 원하는 데이터를 수집하였다.

2. 병렬화 방법

먼저 곱셈의 loop 순서를 i>k>j 순서로 변경하였고, 가장 바깥쪽 loop의 바깥에 OpenMP parallel 커맨드를 통해서 이 loop를 thread들이 parallel하게 처리하도록 하였고, MPI_Send를 통해서 가로 줄을 node 개수만큼 나누어서 master node에서 slave node들에 전송하고, master node와 slave node들이 parallel하게 1/(node 개수) 만큼의 work들을 수행한 후 slave node의 결과들을 MPI_Recv를 통해서 얻어 와서 총 결과를 c matrix에 담았다.

3. 실행 결과

1) thread 16 node 1,2,4 slot 1 mat 4096*4096

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   40.668 sec (40668.266 ms)
```

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   21.364 sec (21364.378 ms)
```

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   10.743 sec (10743.160 ms)
```

차례로 node 1,2,4개를 사용했을 경우 두 matrix의 곱 연산에 걸린 시간이다.

2) node 4, slot 1, thread 1,2,4,8,16,32 mat 4096*4096

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   140.643 sec (140642.884 ms)
```

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   73.109 sec (73109.401 ms)
```

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   37.390 sec (37390.208 ms)
```

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   20.365 sec (20364.864 ms)
```

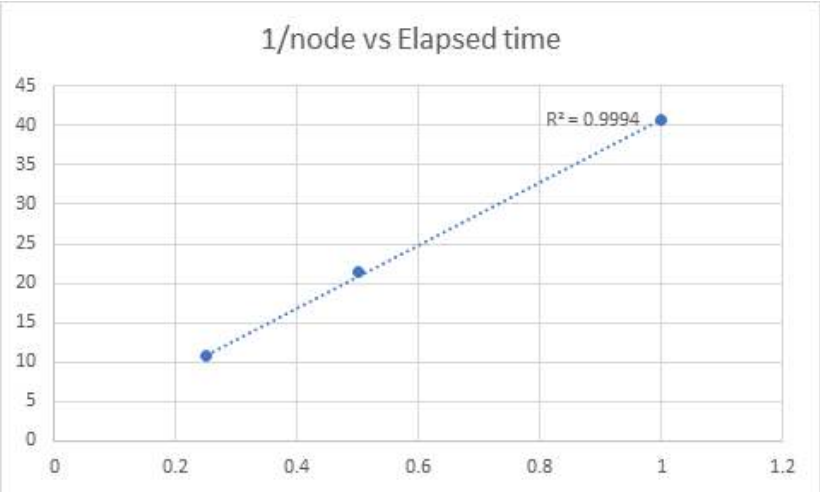
```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   10.743 sec (10743.160 ms)
```

```
Problem size:   4096 x 4096 x 4096

 Initializing ...       done!
 Calculating ...        done!
 Validation off.

Elapsed time:   10.161 sec (10161.363 ms)
```

차례로 thread 1,2,4,8,16,32개를 사용했을 경우 두 matrix의 곱 연산에 걸린 시간이다.

4. 결과 분석

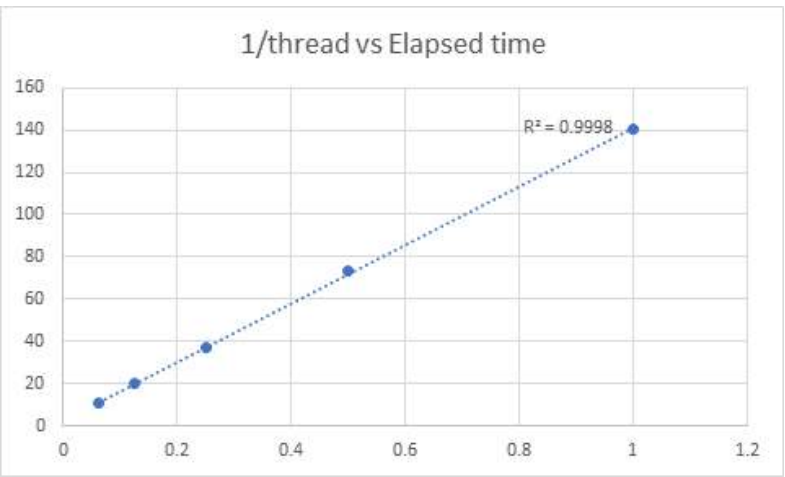
1) node 개수에 따른 수행시간 분석



그래프는 1/(node 개수) 와 Elapsed Time을 두 축으로 하여 그렸다.
일직선을 그리는 것으로 보아 node의 개수에 비례하게 수행 속도가 빨라짐을 알 수 있다. 코드에서 같은 양의 work를 실행시켰으므로 node사이의 연산 처리속도가 비슷함을 알 수 있다.

2) thread 개수에 따른 수행시간 분석

16개일 때의 수행시간과 32개일 때의 수행시간이 거의 비슷한데, 이는 천둥의 코어 개수가 node당 16개이기 때문이라 생각한다.
32를 제외하고 그래프를 그려보면 다음과 같다.



일직선을 그리는 것으로 보아 thread의 개수에 비례하게 수행 속도가 빨라짐을 알 수 있다. 또한, thread들 간의 처리 속도는 과제1번에서

보았듯이 비슷하므로 코드에서 OpenMP의 parallel 커맨드가 thread들 사이의 load-balancing을 잘 수행하고 있다는 것을 알 수 있다.