목차

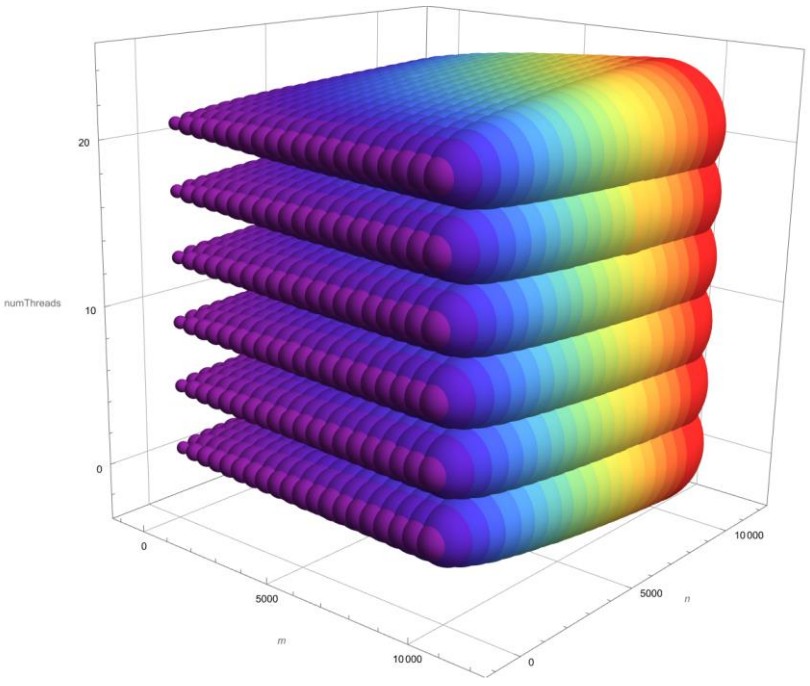
1. 실행 결과

Lab 2-1. Matrix x Vector

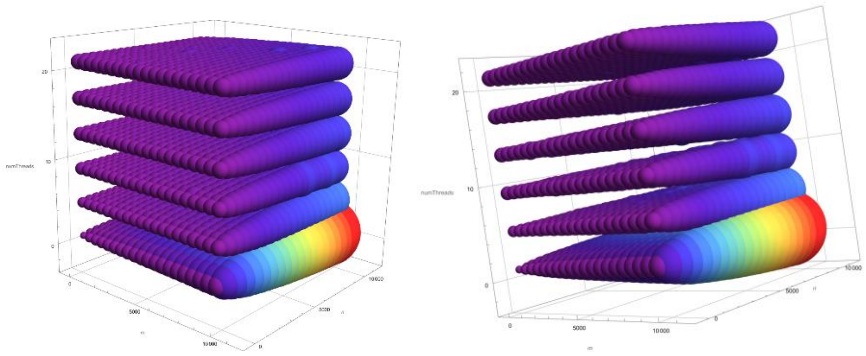
```
m: 10000, n: 10000, numThreads: 24
Results are matched! :)

*         DS_timer Report         *
* The number of timer = 2, counter = 2
**** Timer report ****
Serial : 270.30940 ms (270.30940 ms)
Parallel : 21.62010 ms (21.62010 ms)
**** Counter report ****
*         End of the report       *

Process finished with exit code 0
```

Serial 알고리즘은, 일반적인 행렬곱을 구현하였고, Parallel 알고리즘은 Loop construct를 이용하여, row를 순회하는 반복문을 병렬화하는 방식으로 코드를 구성하였다.

m과 n, 스레드 개수에 따라서 어떤 실행 시간의 차이를 보이는지 확인하기 위해서 명령줄 인자로 해당 매개 변수들을 받을 수 있게한 뒤, Mathematica를 이용하여 가시화하였다. 입체 버블 차트를 이용하였으며, 그래프에서 (m, n, numThreads)에 존재하는 구의 크기는 실행 시간에 비례한 크기를 가진다. (*Ryzen 9 3900X에서 실험됨, 12코어, 24스레드)



(Serial 결과, 최소 실행 시간: 0.0252MS, 최대 실행 시간: 232.409MS)



(Parallel 결과, 최소 실행 시간: 0.0294MS, 최대 실행 시간: 231MS)

Serial 결과에서는, m과 n이 단순히 커질수록 실행 시간이 늘어나는 것을 확인할 수 있었고, Parallel 결과에서는, m과 n이 커질수록 실행 시간이 늘어나지만, 스레드 개수가 늘어날수록 더 빨라진 것을 확인할 수 있었다.

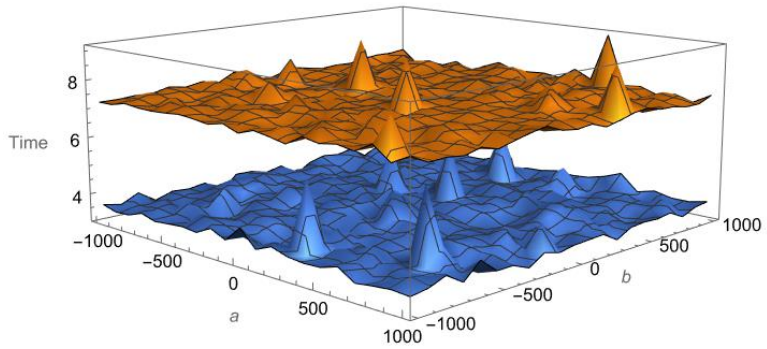
Lab 2-2. Trapezoidal Rule

```
f(x) = x * x
range = (-1.000000, 1.000000), n = 10485768
numThreads: 24
[Serial] area = 0.666667
[Parallel] area = 0.666667
Results are matched! :)

*        DS_timer Report        *
* The number of timer = 2, counter = 2
**** Timer report ****
Serial : 8.05640 ms (8.05640 ms)
Parallel : 3.87600 ms (3.87600 ms)
**** Counter report ****
*        End of the report      *

Process finished with exit code 0
```

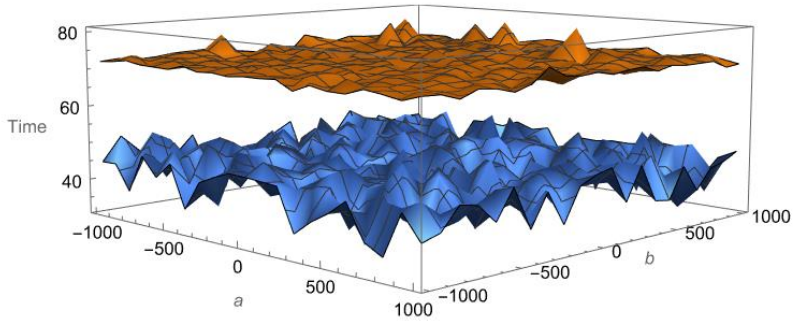
Serial 알고리즘은 순차적으로 구간을 순회하며 구간합을 구하는 방식을 사용했다. Parallel 알고리즘은, 합할 구간을 스레드 개수별로 다시 나눈 뒤, 각 구간합을 스레드 번호를 인덱스로 가지는 배열을 통해 계산하고, 이후 병렬 계산이 끝나면 배열의 값을 다시 합산하여, 최종 구간합을 산출하는 방식을 사용했다.

a과 b, n, 스레드 개수에 따라서 어떤 실행 시간의 차이를 보이는지 확인하기 위해서 명령줄 인자로 해당 매개 변수들을 받을 수 있게한 뒤, Mathematica를 이용하여 가시화하였다.

우선 구간 설정(a, b)은 병렬 처리 시간에 별다른 영향을 끼치지 않을 것이라는 가설을 세우고 a, b를 각각 (-1000, 1000) x (-1000, 1000) 공간에서 100 단위로 샘플링하여 시간을 측정하였다. (주황: Serial, 파랑: Parallel)



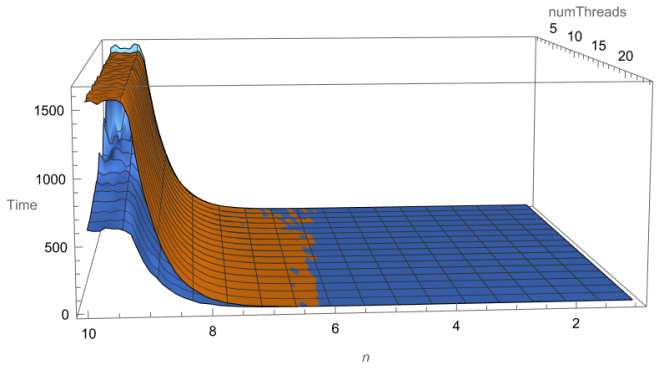
Case A: (n=10^7, numThreads=24)



Case B: (n=10^8, numThreads=12)

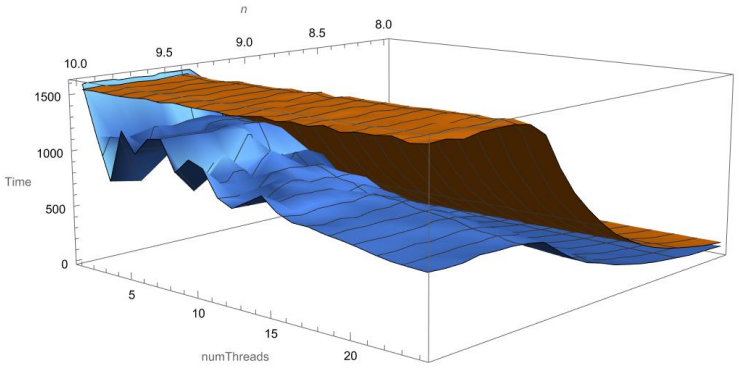
측정 결과, 어느 a, b 값을 사용하여도 오차 범위 내에서 동일한 수준의 시간 성능이 관찰됨을 확인할 수 있었다.

다음은 a=-1, b=-1에서 n=[10^1, 10^10], numThread=[1, 24] 구간에서의 시간을 측정하여 가시화한 그래프이다.



(a=-1, b=1, 주황: Serial, 파랑: Parallel)

*. 그래프에서 n은 10^n에 들어가는 지수임



(n=[10^8, 10^10]으로 구간 확대)

Serial의 경우 n이 증가함에 따라 선형적 시간 증가를 보였고, Parallel의 경우 numThreads가 높을수록 완만한 시간 증가를 보여줬다. 특히 numThreads=24일 때, Serial 대비 절반 정도의 소요 시간이 걸린 것을 확인할 수 있었다.

2. 소감

이번 과제를 통해 병렬 프로그래밍에 대한 이해를 높이고, 강의에서 배운 OpenMP 구문과 개념을 이용하여 프로그래밍에 실제로 적용해볼 수 있었다. 또한, 병렬 처리가 시간 복잡도를 줄이는 효과를 직접 확인함으로써, 병렬 처리의 중요성을 더욱 깊게 느낄 수 있었다.