목차

1. 실행 결과

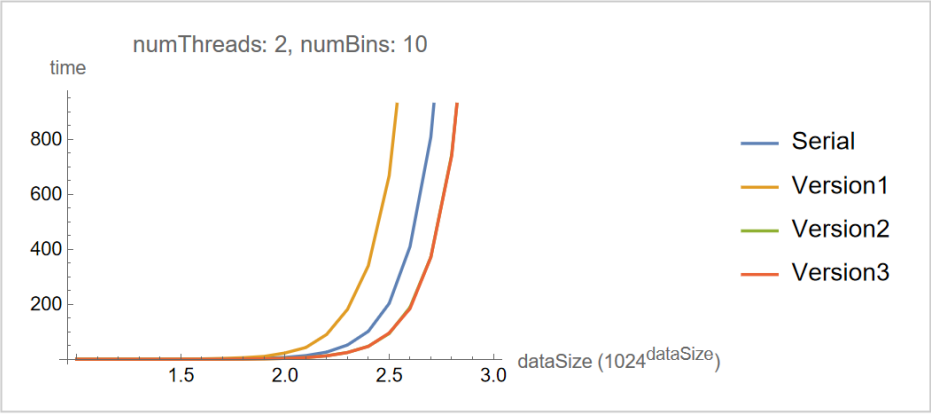
Lab 3. Histogram

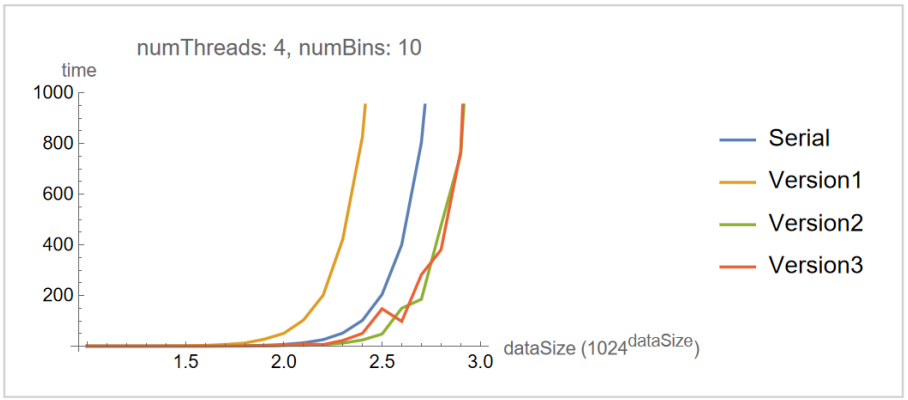
```
*         DS_timer Report         *
* The number of timer = 4, counter = 4
**** Timer report ****
Serial : 6622.40240 ms (6622.40240 ms)
Version1 : 0.00010 ms (0.00010 ms)
Version2 : 1146.89200 ms (1146.89200 ms)
Version3 : 1528.20600 ms (1528.20600 ms)
**** Counter report ****
*         End of the report       *
```

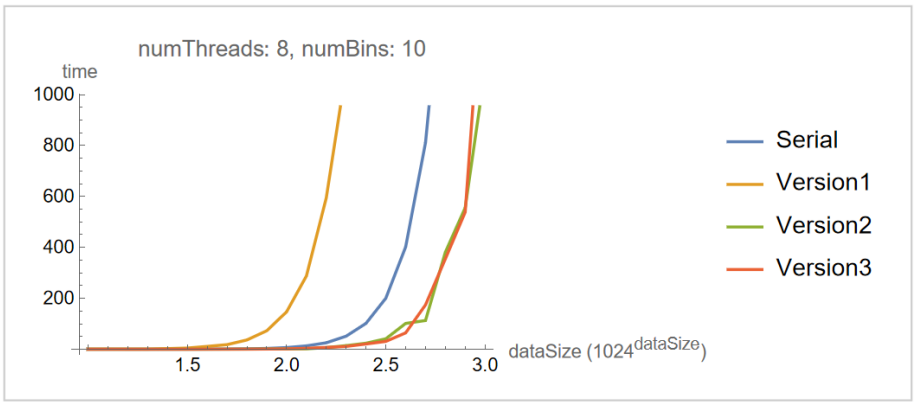
(bin 개수 10개, 8 스레드, 데이터 1024 * 1024 * 1024개에서의 결과)

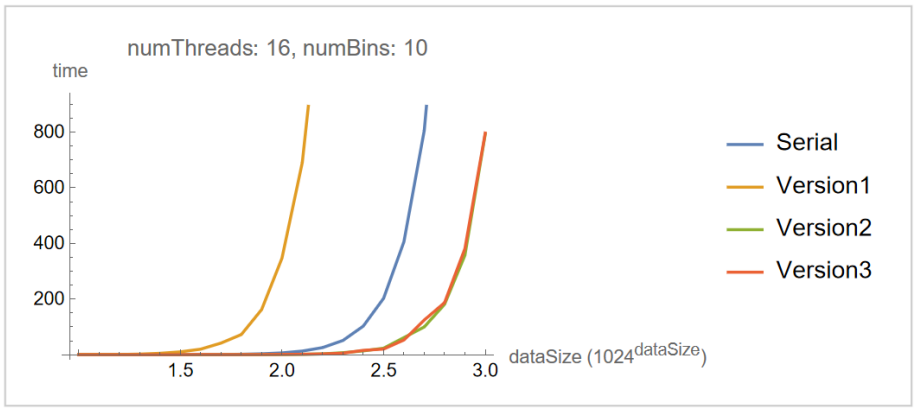
*. Version1의 경우 시간이 너무 많이 걸려서 위 결과에서는 미포함하였음

Serial 버전은 데이터에 따른 인덱스를 계산한 뒤, 카운트 하는 방식으로 구현하였고, Version1의 경우, 병렬 for문을 이용하여 카운트를 진행하되, 빈에 접근할 때 lock을 이용하여 임계 영역으로 보호받도록 구현하였다. Version2는 병렬 for를 이용하여 localBins을 구성한 뒤, atomic 연산을 이용하여 최종 빈 배열에 localBins를 병합하였다. Version3은 각 스레드별 루프 횟수를 구한 뒤, 두 스레드씩을 묶어서 Reduction sum을 구현하였다.







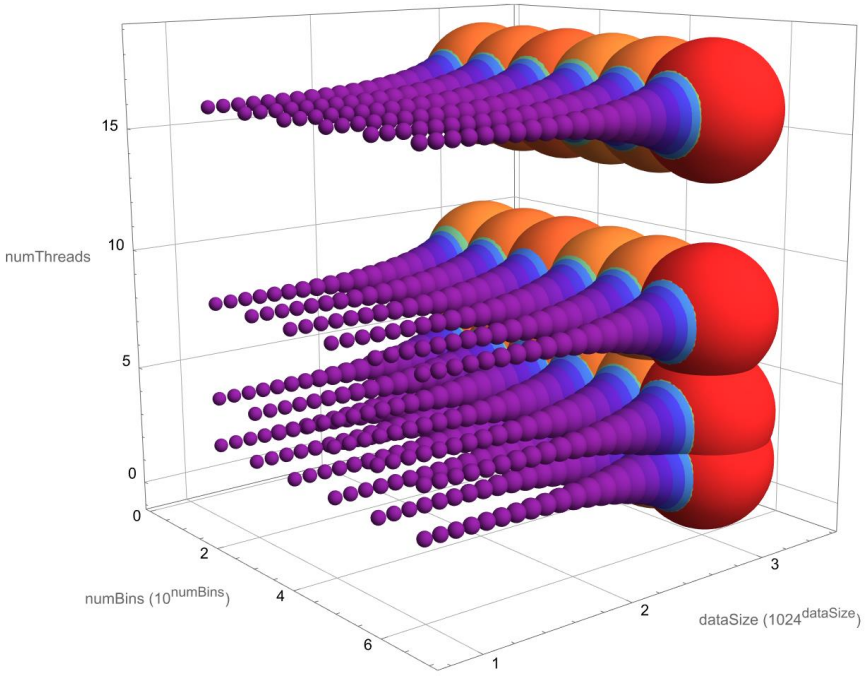


(각 스레드 조건에서의 속도 비교)

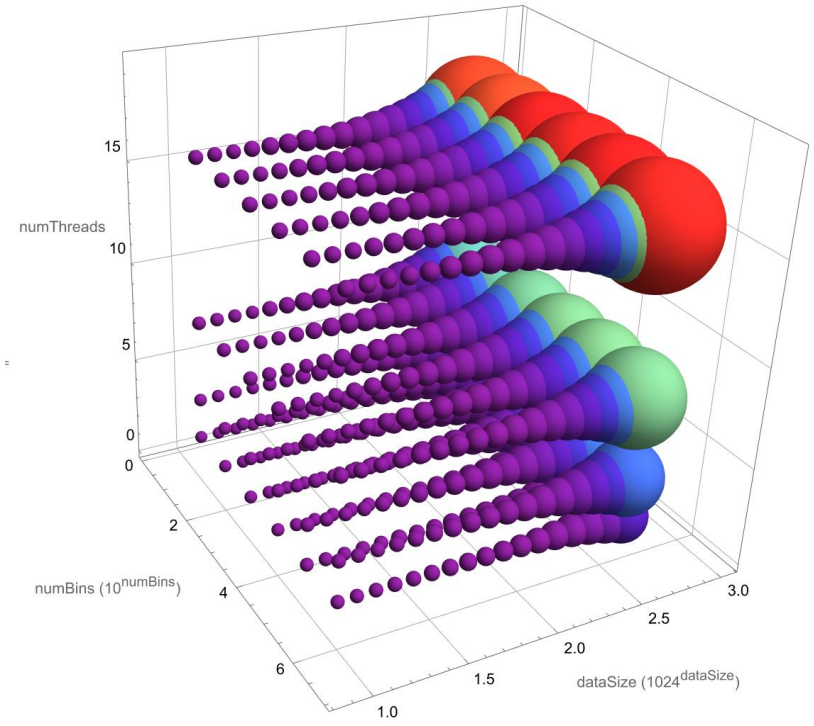
일부 조건에서는 Version3이 빠른 경우가 있었으나, 전체적으로 Version2와 Version3의 성능은 비슷하고, 스레드가 개수가 늘어날수록 Version1의 성능은 하락, Version2, Version3의 성능은 증가한 것을 확인할 수 있었다.

빈 개수와 데이터 수, 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 (numBins, dataSize, numThreads)에 존재하는 구의 크기는 (10^numBins, 1024^dataSize,

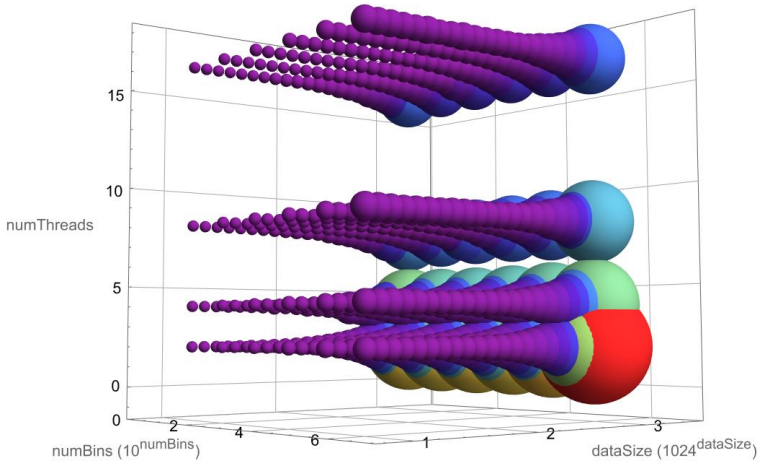
numThreads) 조건에서 실행한 프로그램의 실행 시간에 비례한 크기를 가진다. (*Ryzen 9 3900X에서 실험됨, 12코어, 24스레드)



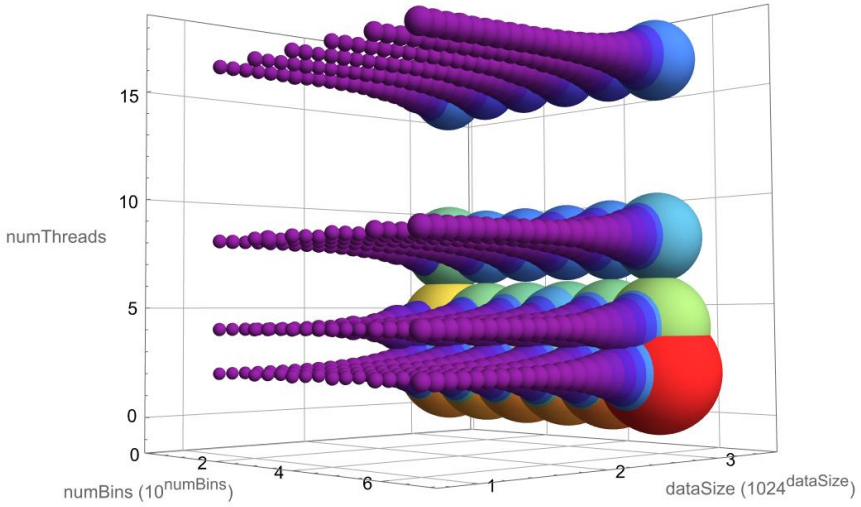
(Serial 결과)



(Version 1 결과)



(Version 2 결과)



(Version 3 결과)

전체 결과 모두 데이터 사이즈가 커지고, 빈 개수가 늘어날수록 실행 시간이 늘어난 것을 확인할 수 있었다. Version1의 경우에는 스레드 개수가 커질수록 실행 시간이 늘어났고, Version2와 Version3의 경우에는 스레드 개수가 커질수록 실행시간의 감소를 확인할 수 있었다.

2. 소감

이번 병렬처리 과제를 통해 여러 가지 병렬처리 방법에 대해 배울 수 있었으며, 이를 실제 프로그램에 적용해보면서 각 방법의 성능 차이와 특징을 이해하는 데 도움이 많이 되었다. 각 버전 별로 구현해본 결과, 최적화된 병렬처리 방법을 사용하면 많은 성능 향상을 얻을 수 있다는 것을 깨달았다.