Density-based spatial clustering of applications with noise

Parallel DBSCAN

MP OpenMP 프로젝트

2016136092 이상민
2017136008 권창덕
2018136153 최원홍

CONTENTS

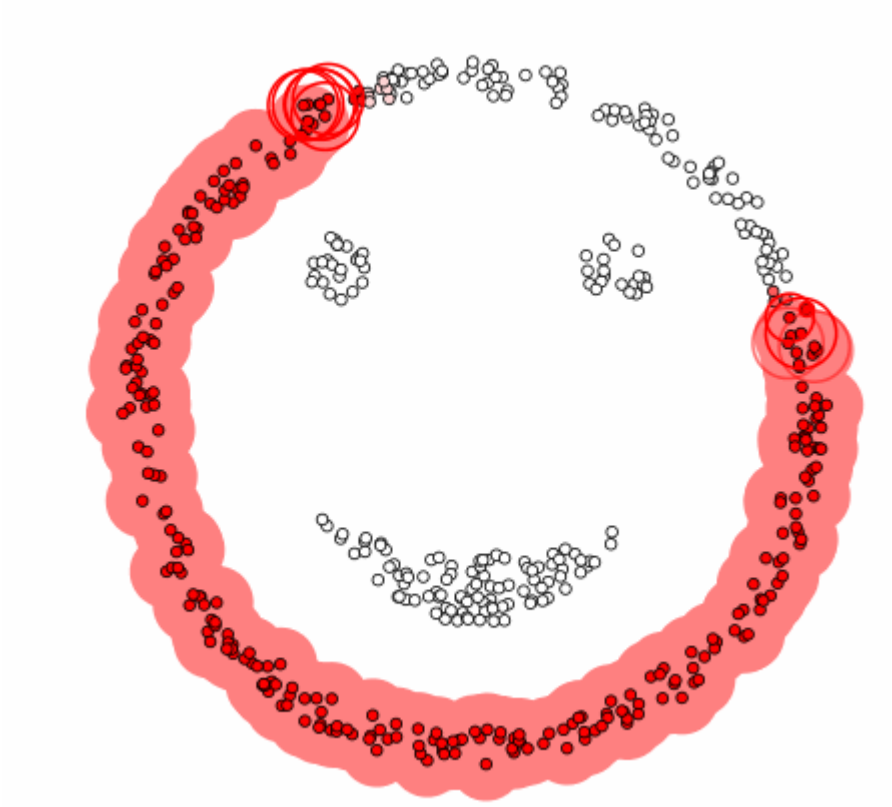
01

DBSCAN

Density-based spatial clustering of applications with noise

DBSCAN

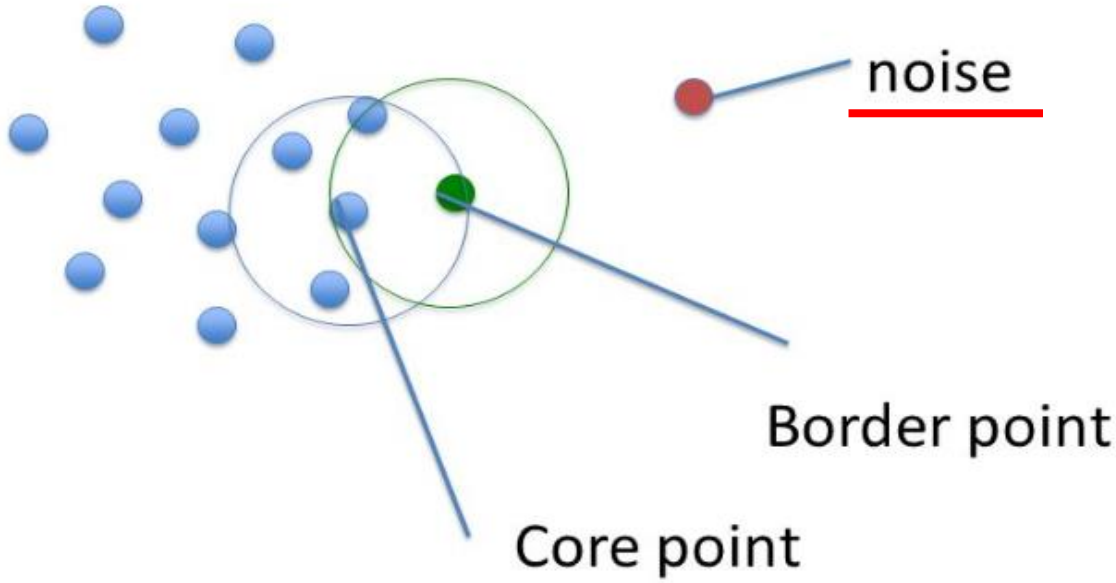
Density-based spatial clustering of applications with noise



1. 가장 널리 알려진 밀도(density) 기반의 군집 알고리즘
2. 비지도 학습
3. 불특정한 형태의 군집이 가능

[IMG] https://medium.com/analytics-vidhya/cluster-analysis-with-dbscan-density-based-spatial-clustering-of-applications-with-noise-6ade1ec23555

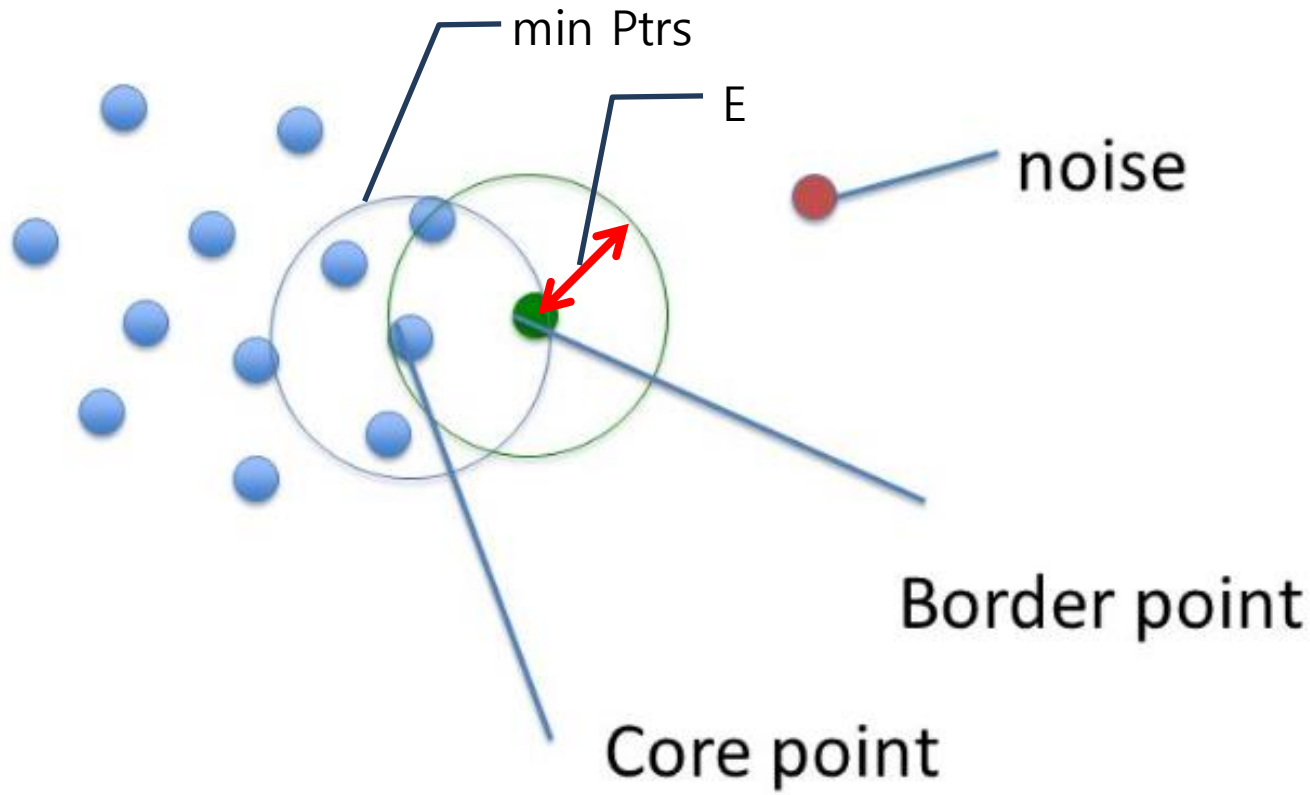
DBSCAN 활용



NOISE 검출이 가능하므로 Outlier를 detection하는 분야에서 사용

DBSCAN

밀도(density) → Epsilon(radius), min # of Points



DBSCAN VS K-means

DBSCAN

k-means

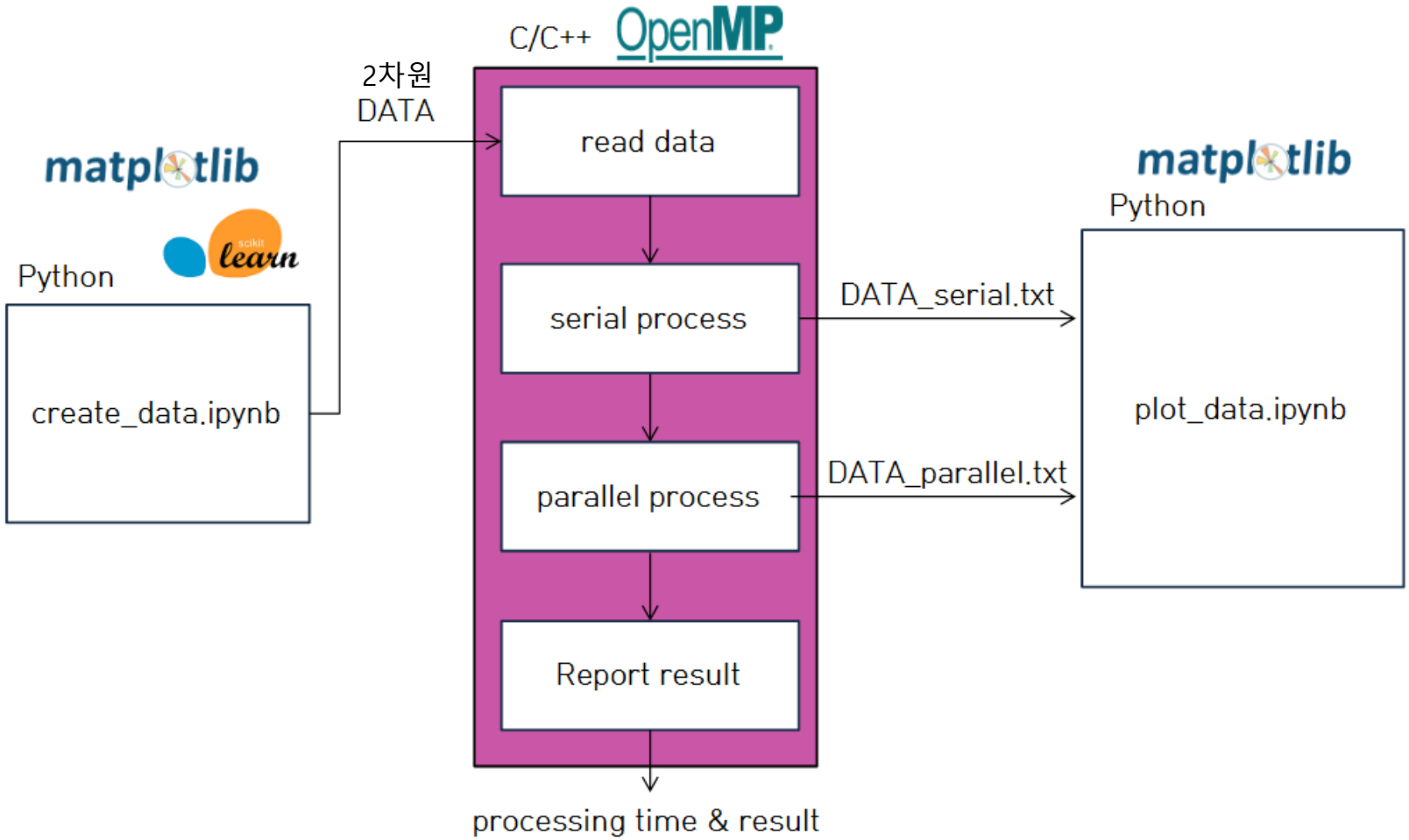
구분	parameter	장점	단점
DBSCAN	E, min_ptrs	노이즈 검출 용이	느린 계산 시간
K-means	K	빠른 계산 시간	노이즈에 민감

[IMG] https://mrinalyadav7.medium.com/dbscan-algorithm-c894701306d5

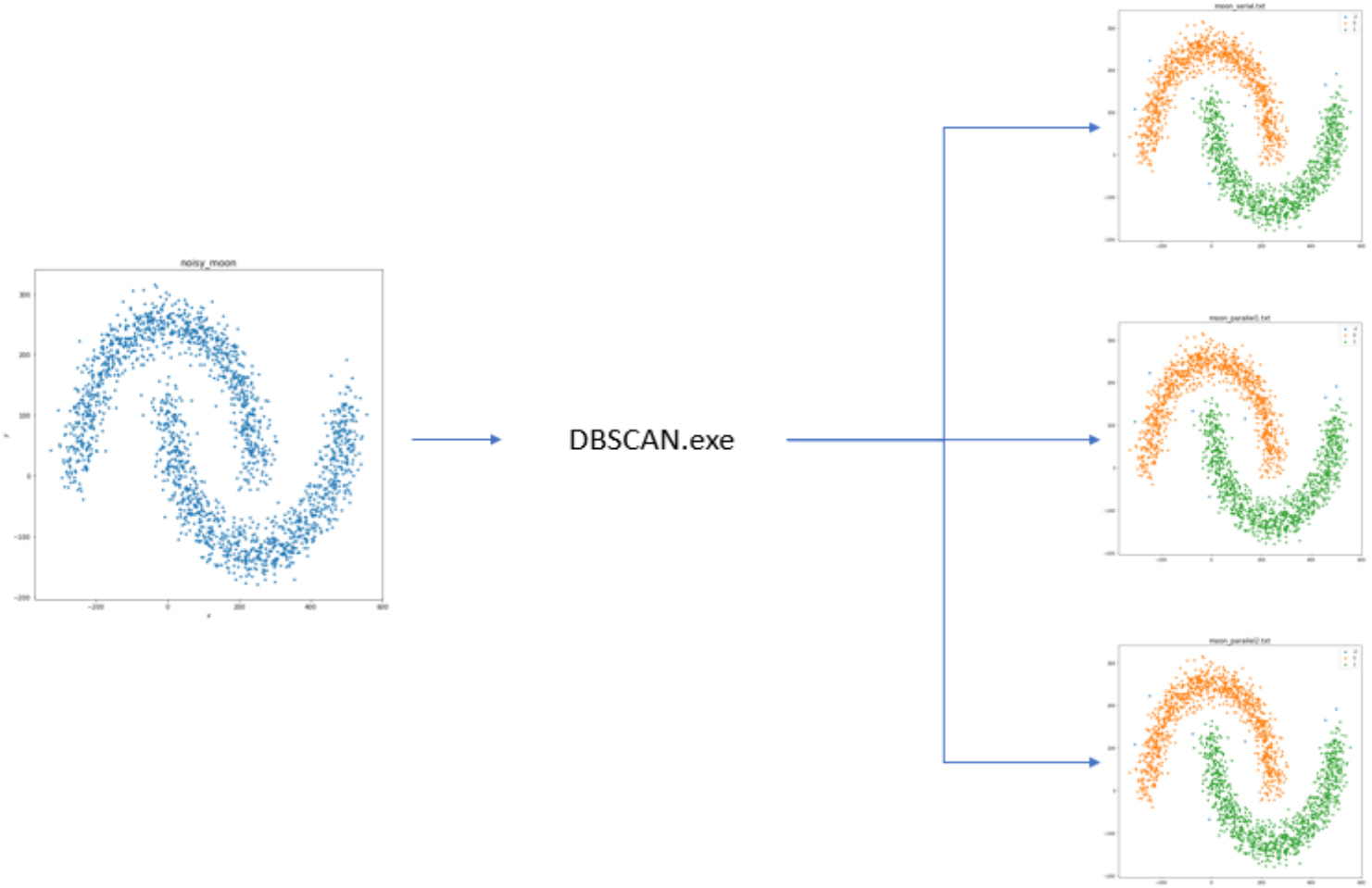
02

구현

전체 구현 과정


C/C++
OpenMP
2차원
DATA
matplotlib
Python
create_data.ipynb
read data
serial process
parallel process
Report result
processing time & result
DATA_serial.txt
DATA_parallel.txt
matplotlib
Python
plot_data.ipynb

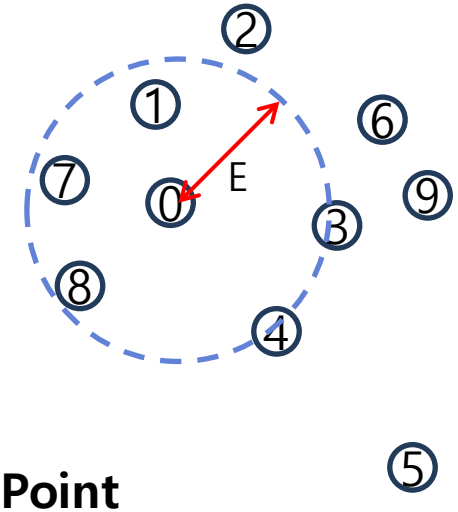
전체 구현 과정

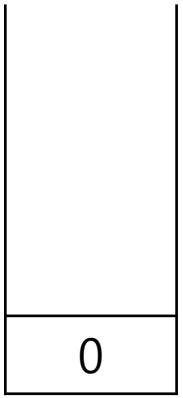


serial 구현

Depth-First-Search(DFS)

데이터를 읽어 드린 순서대로 0, 1, 2, ... → 모든 point에 접근



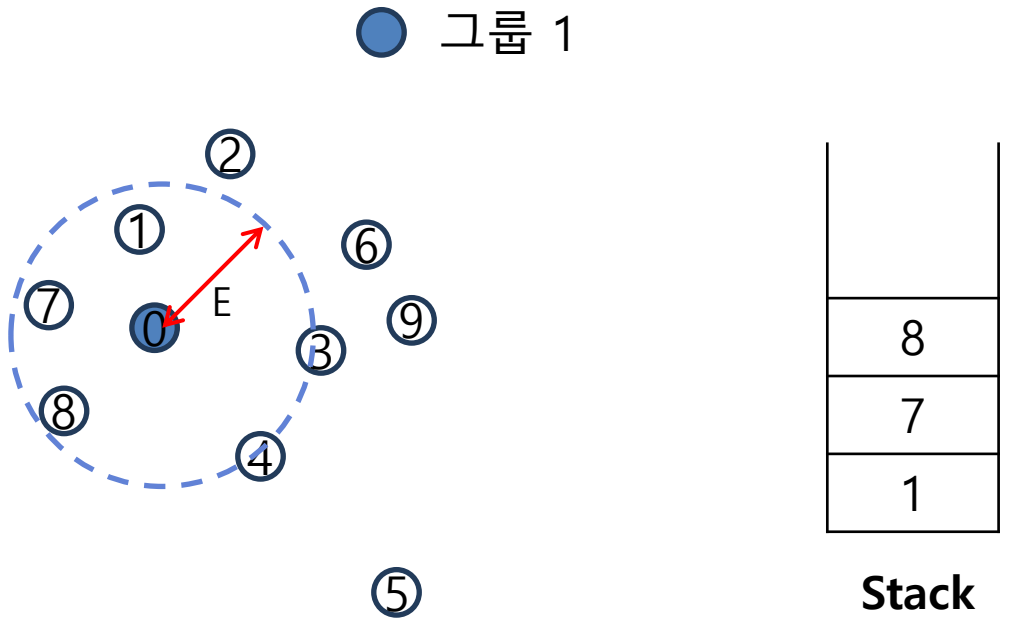


Stack

Struct Point
- **(x,y) 좌표**
- **그룹 ID**
- **dummy**

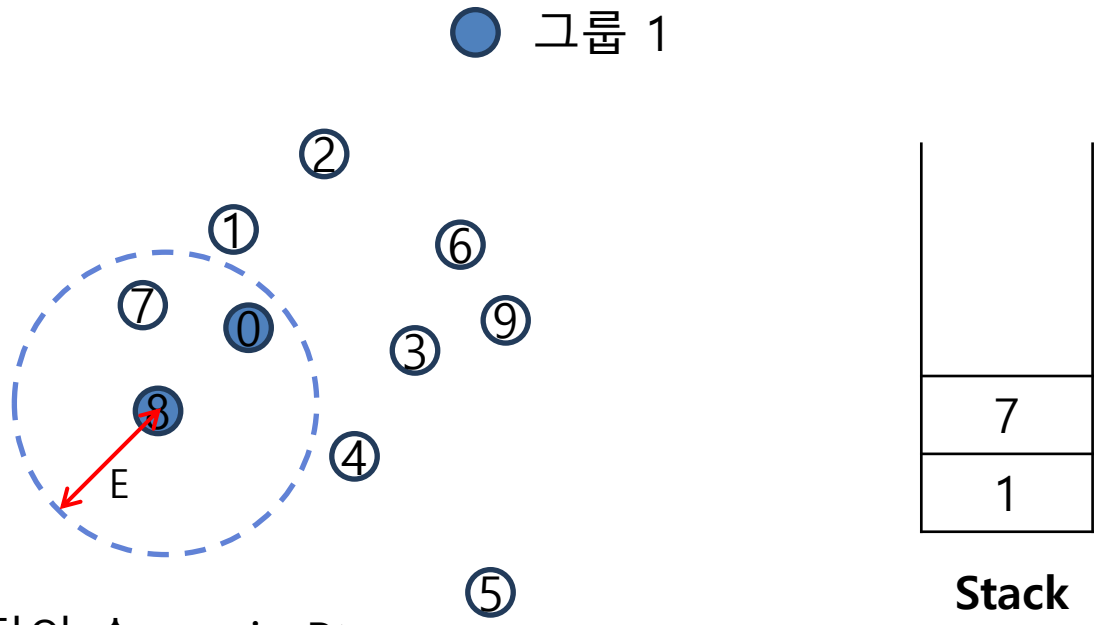
min Ptrs ==3

serial 구현



min Ptrs ==3

serial 구현

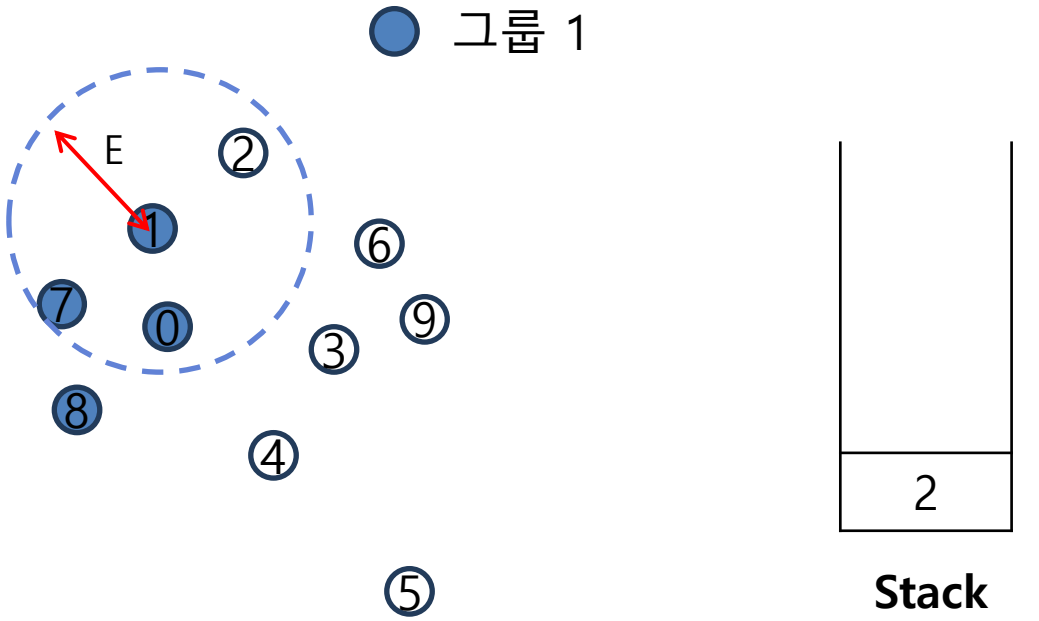


Stack

이웃점의 수 < min Ptrs
이지만, 코어 점 0이 있으므로..

min Ptrs ==3

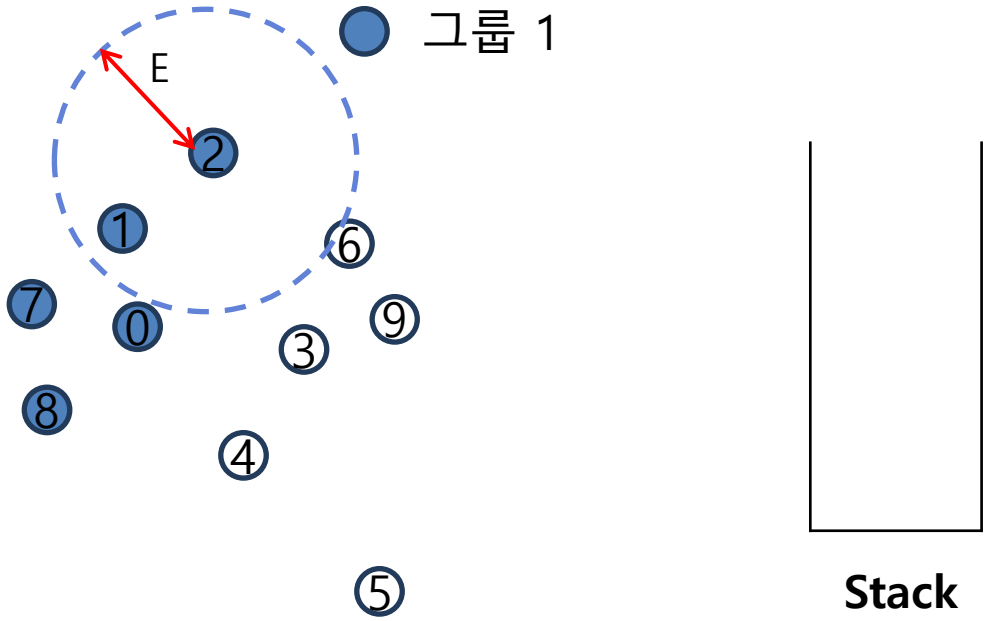
serial 구현



core점이지만, 이미 0번(코어점)
영역에 포함되므로 같은 그룹

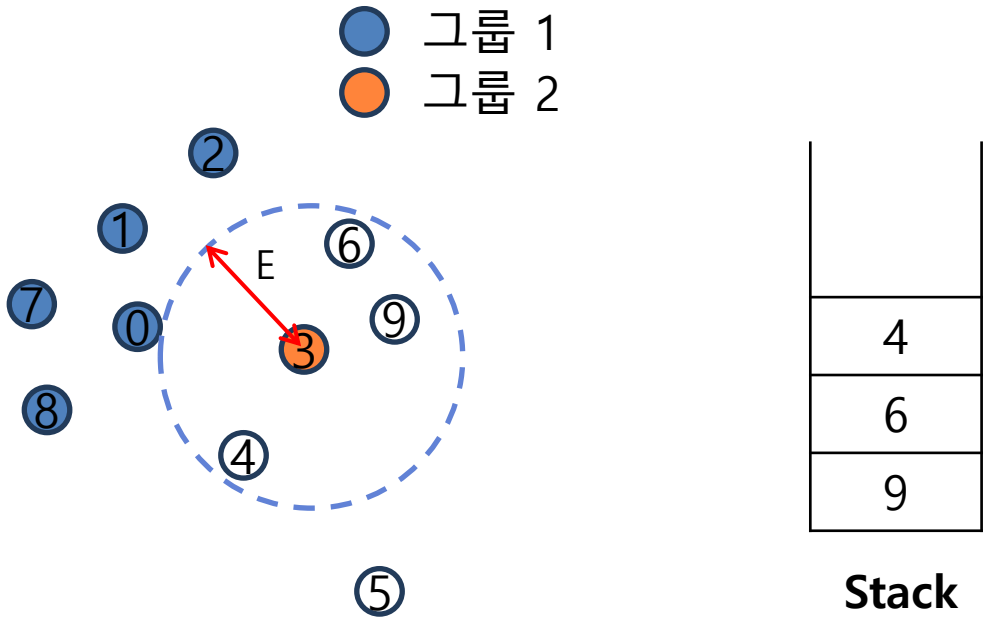
min Ptrs ==3

serial 구현



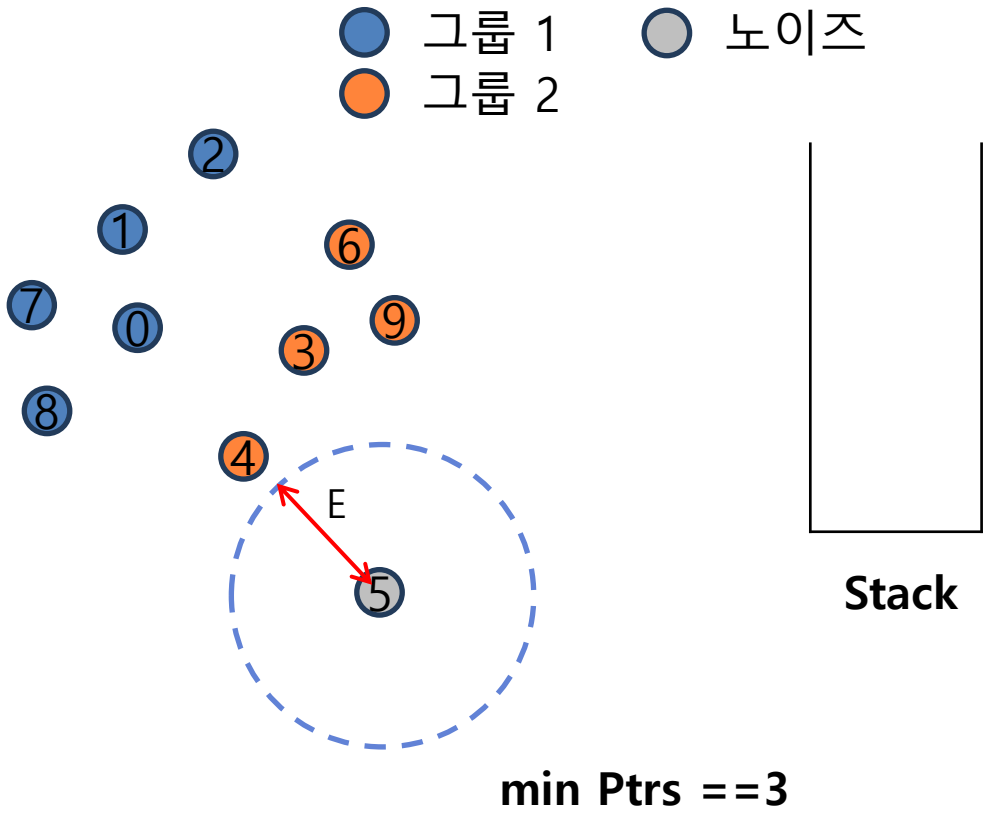
min Ptrs ==3

serial 구현



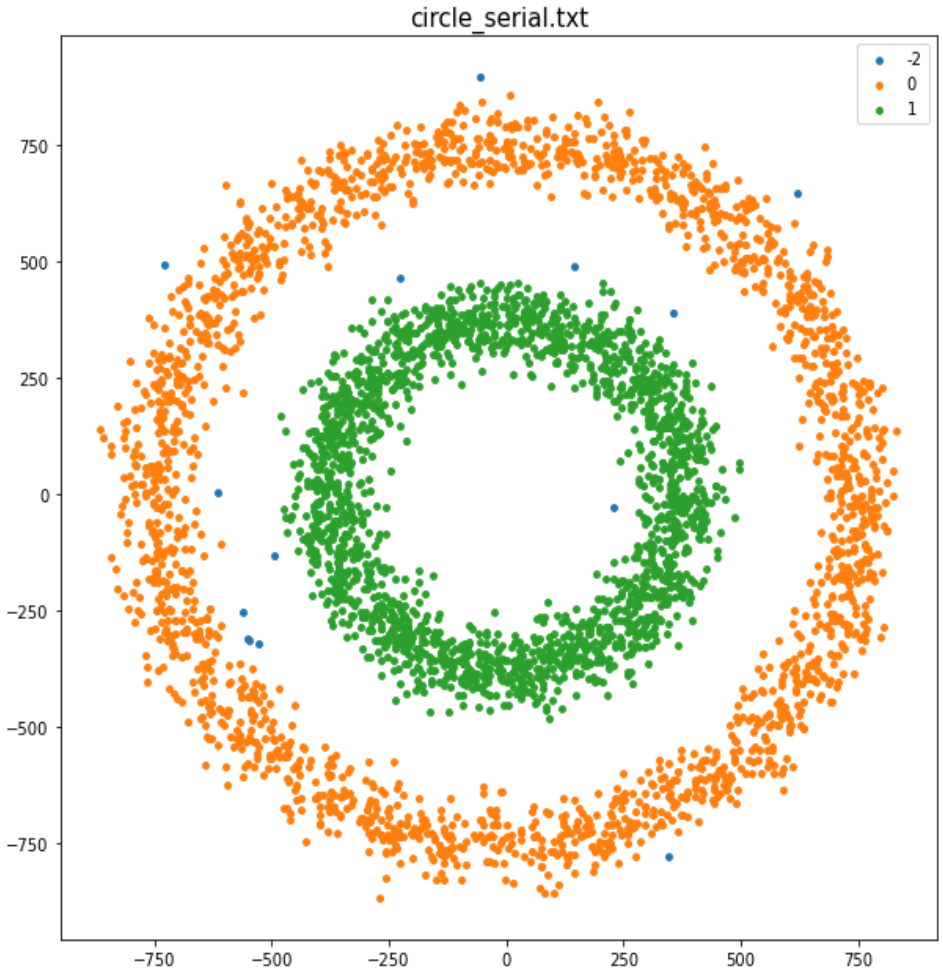
min Ptrs ==3

serial 구현



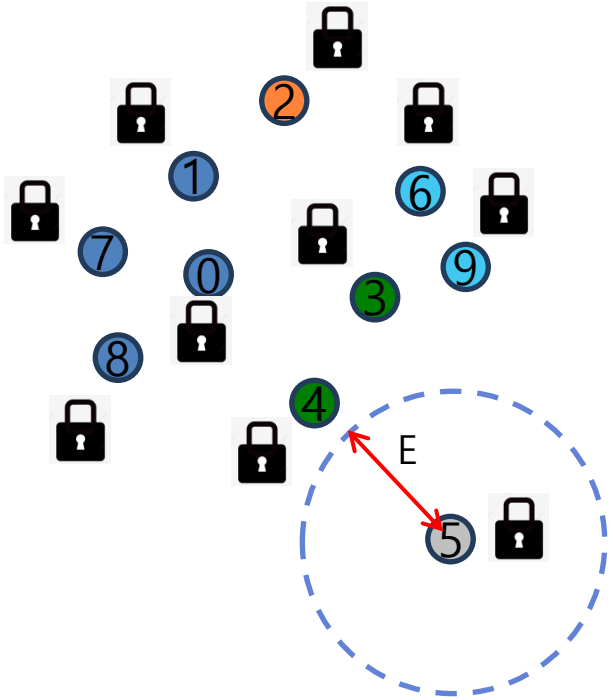
min Ptrs ==3

serial 구현


circle_serial.txt
-2
0
1
750
500
250
0
−250
−500
−750
−750
−500
−250
0
250
500
750

parallel 구현

1) 각 Thread별 local clustering 수행
2) 인접한 local 그룹을 merge



여러 쓰레드가 동일 Point 그룹할당 방지
→ 각 Point별 Lock

0번 그룹(by Thread0)
1번 그룹(by Thread1)

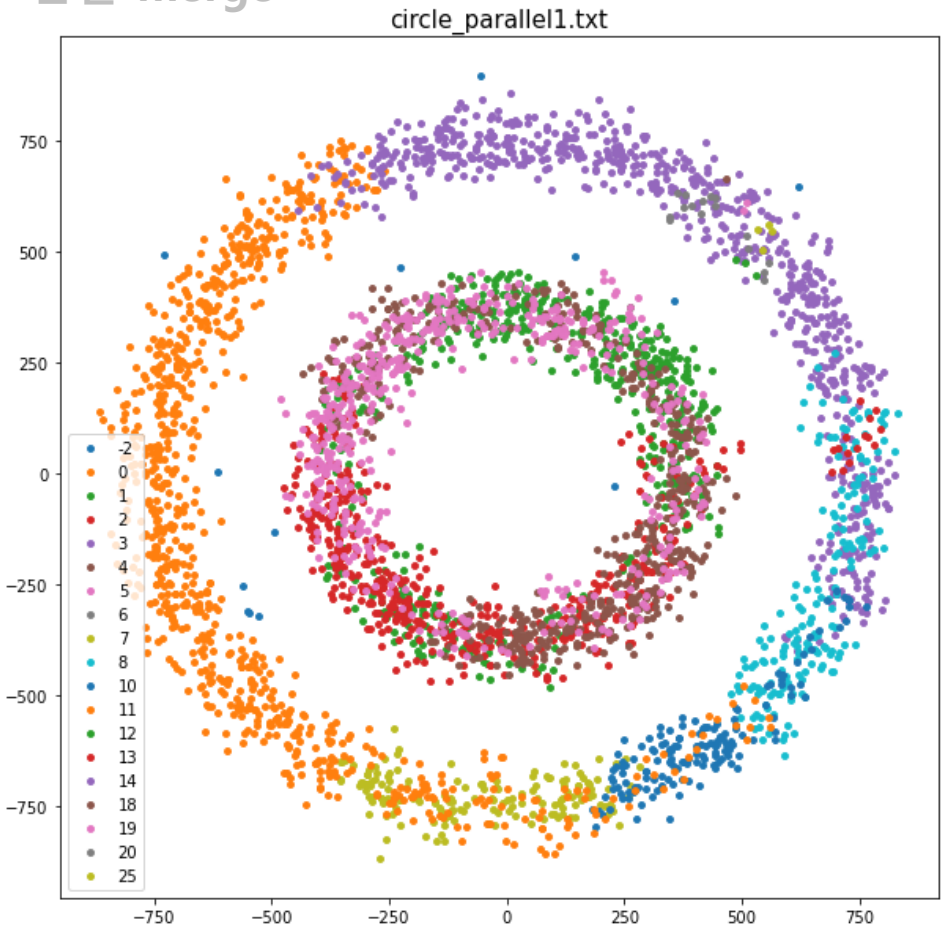
2번 그룹(by Thread0)
3번 그룹(by Thread1)

Thread : 그룹 번호
Thread0 : 0 → 2 → 4 → ...
Thread1 : 1 → 3 → 5 → ...

min Ptrs ==3

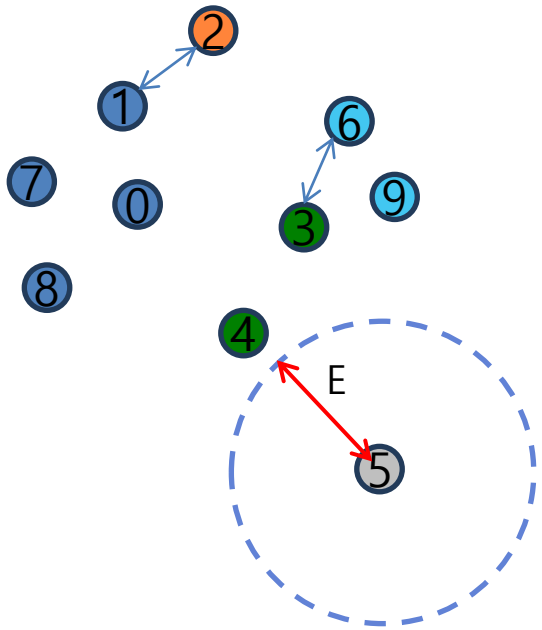
parallel 구현

1) 각 Thread별 local clustering 수행
2) 인접한 local 그룹을 merge

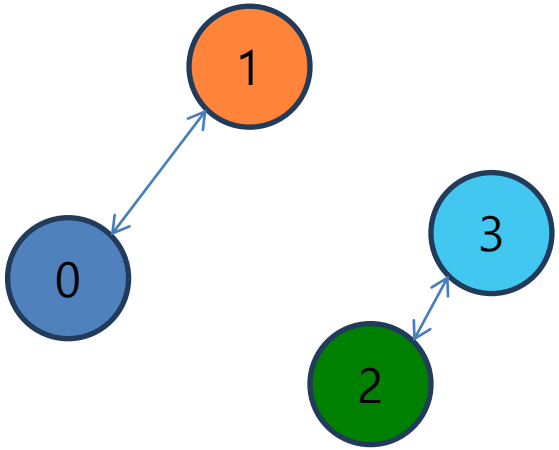


parallel 구현

1) 각 Thread별 local clustering 수행
2) 인접한 local 그룹을 merge



이웃 local 그룹 정보를 담는 그래프 생성



min Ptrs ==3

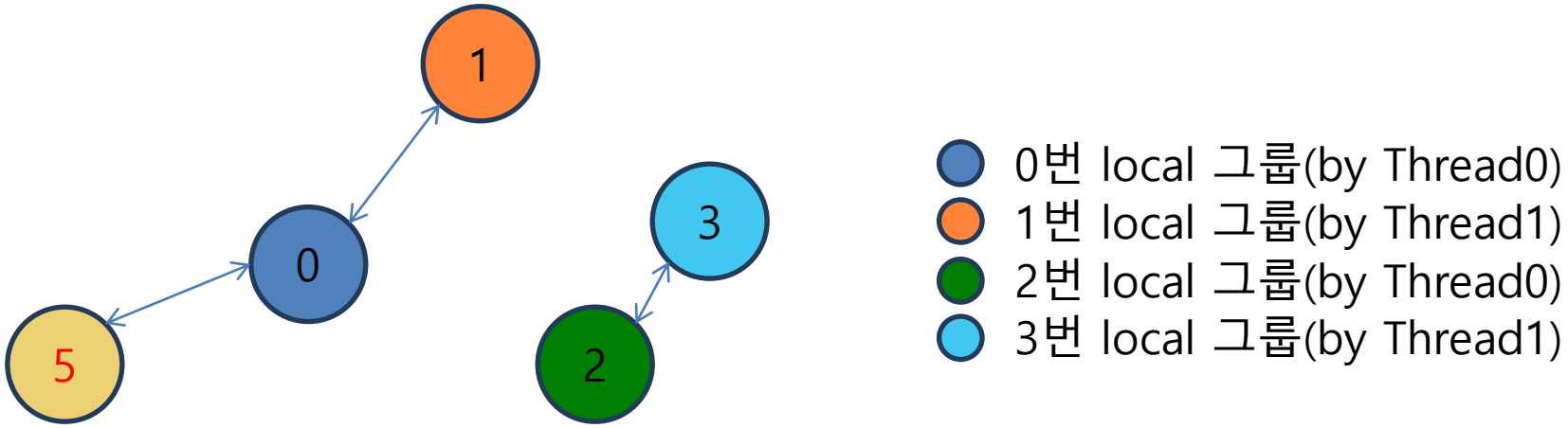
- 0번 그룹(by Thread0)
- 1번 그룹(by Thread1)
- 2번 그룹(by Thread0)
- 3번 그룹(by Thread1)

parallel 구현

1) 각 Thread별 local clustering 수행

2) 인접한 local 그룹을 merge

만약, 5번 그룹이 있을 경우, 5번 그룹과 1번 그룹은 서로 이웃 X

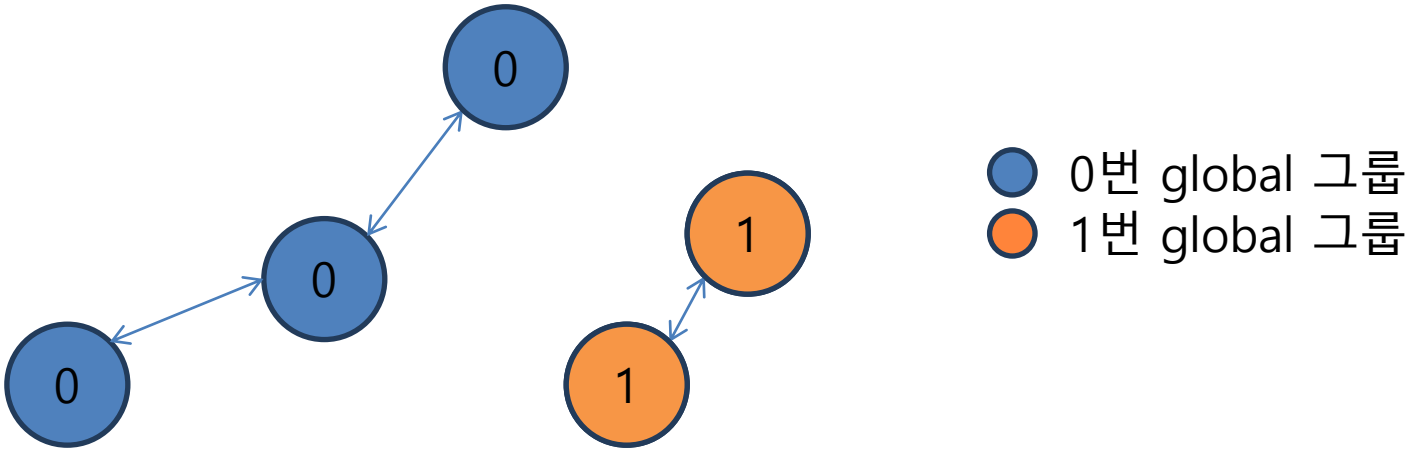


따라서, 이웃 local 그룹 번호를 이용해 다시 DFS 순회 → global 그룹

min Ptrs ==3

parallel 구현

1) 각 Thread별 local clustering 수행
2) 인접한 local 그룹을 merge

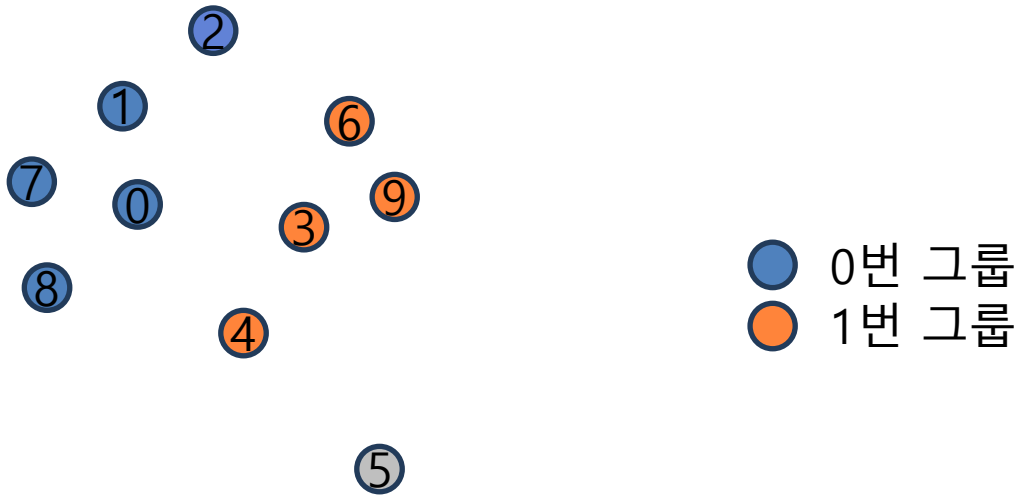


global 그룹으로 치환

min Ptrs ==3

parallel 구현

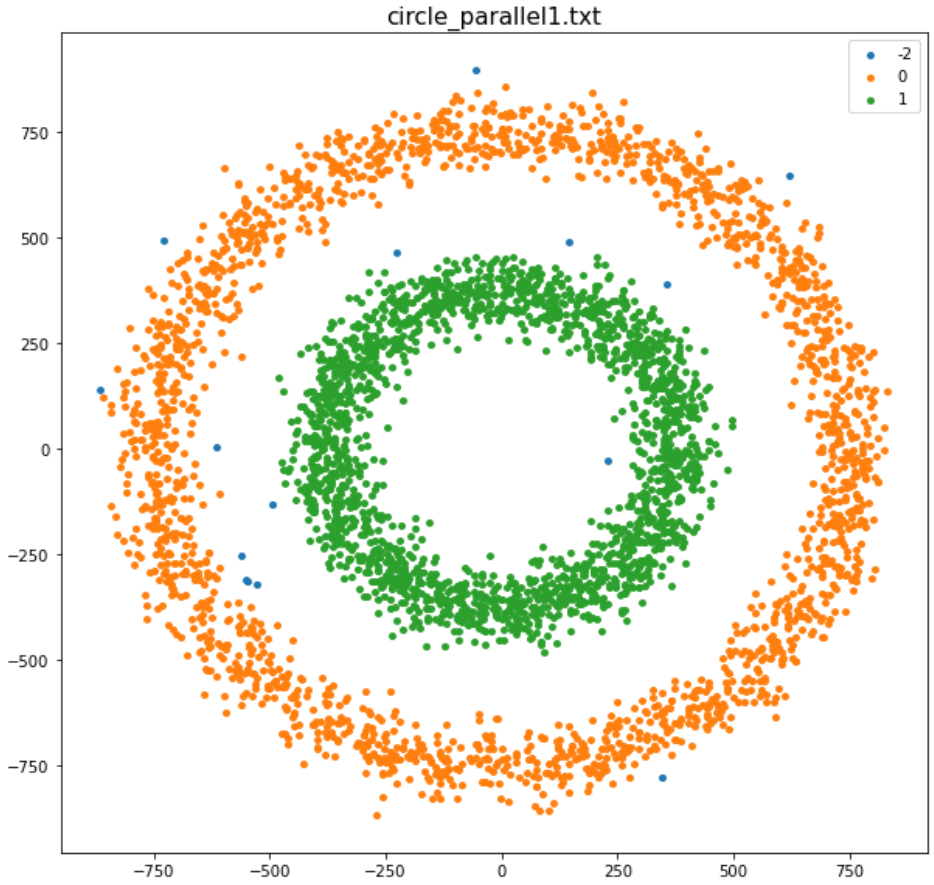
1) 각 Thread별 local clustering 수행
2) 인접한 local 그룹을 merge



min Ptrs ==3

parallel 구현

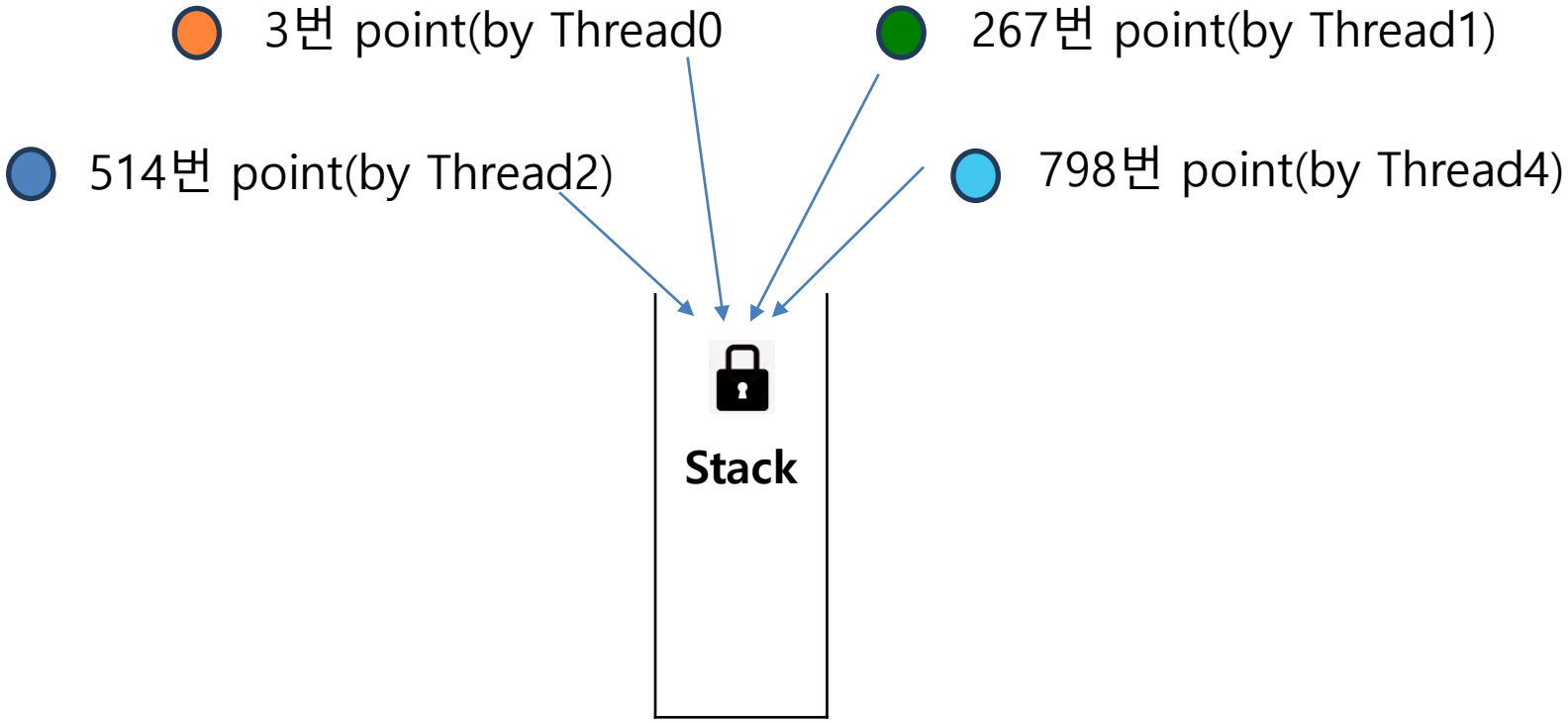
1) 각 Thread별 local clustering 수행
2) 인접한 local 그룹을 merge



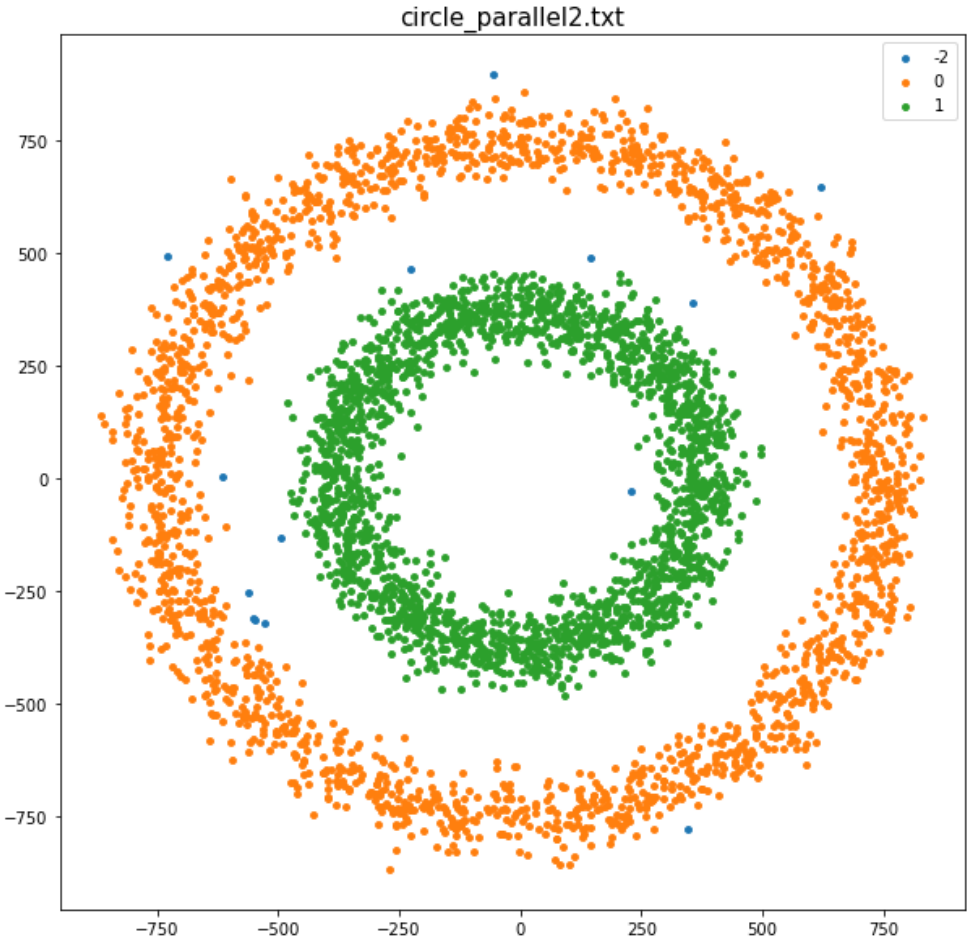
parallel 구현

이웃한 점을 병렬처리로 찾고 Stack을 Lock으로 동기화

stack의 push연산을 병렬처리
stack의 pop연산은 serial



parallel 구현

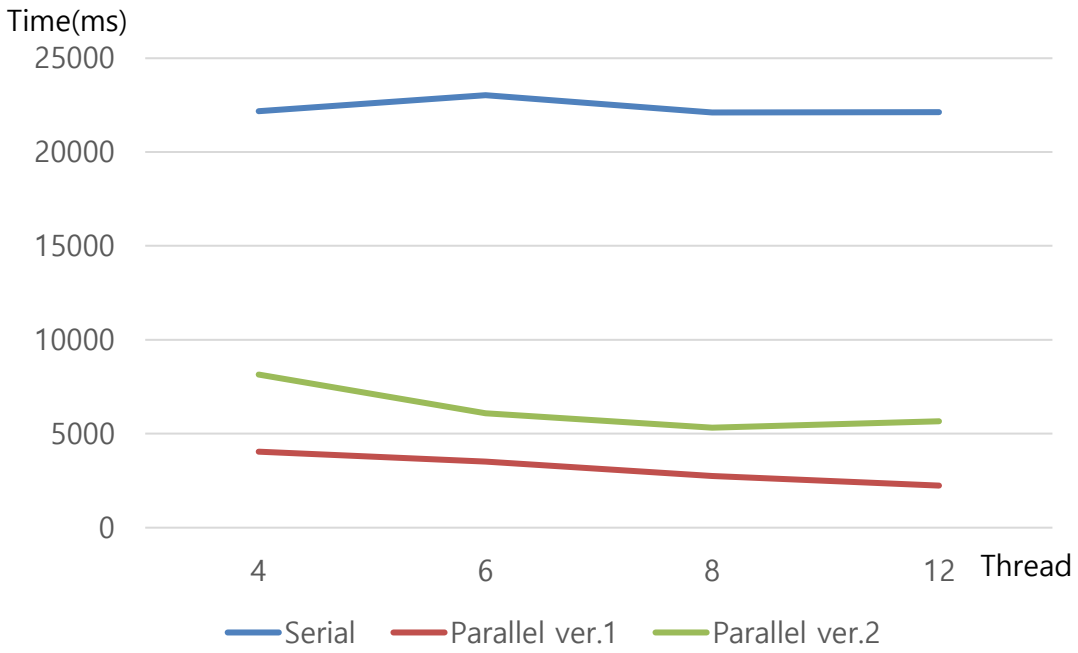

circle_parallel2.txt
-2
0
1
750
500
250
0
−250
−500
−750
−750
−500
−250
0
250
500
750

03

결과 및 분석

결과 분석

-스레드 별 소모 시간



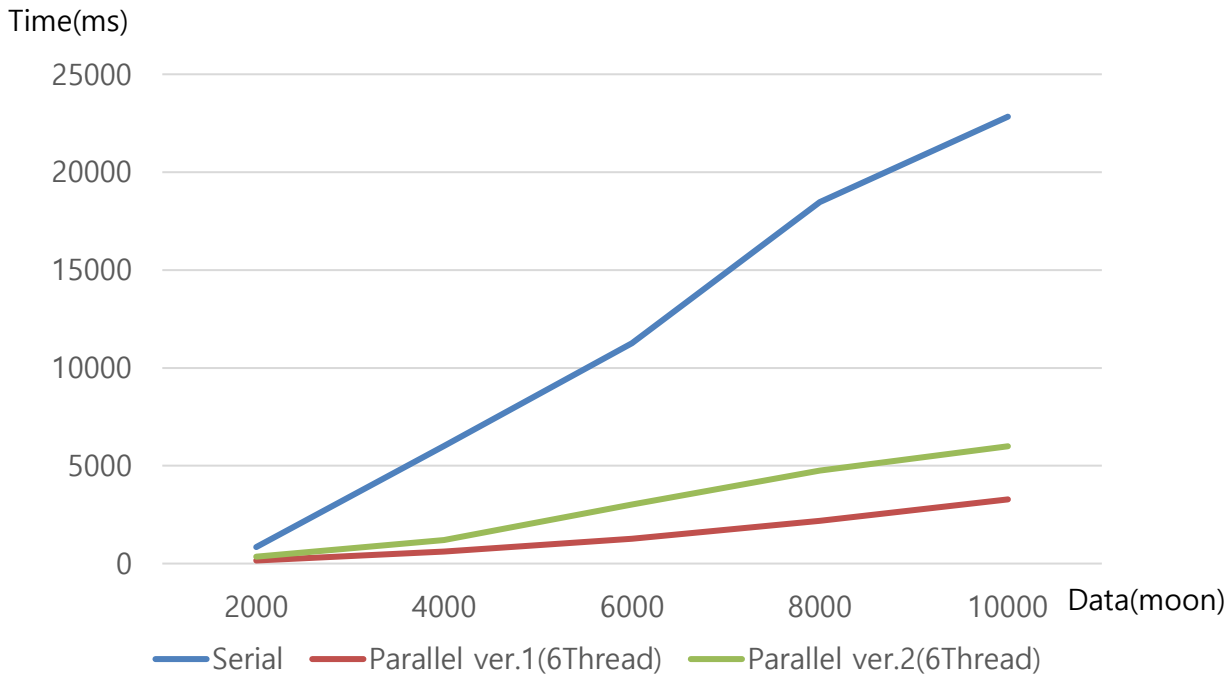
10000개의 데이터를 가진 moon 형식 데이터 처리 소모시간

Parallel ver.1
4 Thread
Serial 대비 성능향상 약 5.487배
12 Thread
Serial 대비 성능향상 약 9.903배

Parallel ver.2
4 Thread
Serial 대비 성능향상 약 2.724배
12 Thread
Serial 대비 성능향상 약 3.909배

결과 분석

- moon형 데이터 량 별 소모 시간

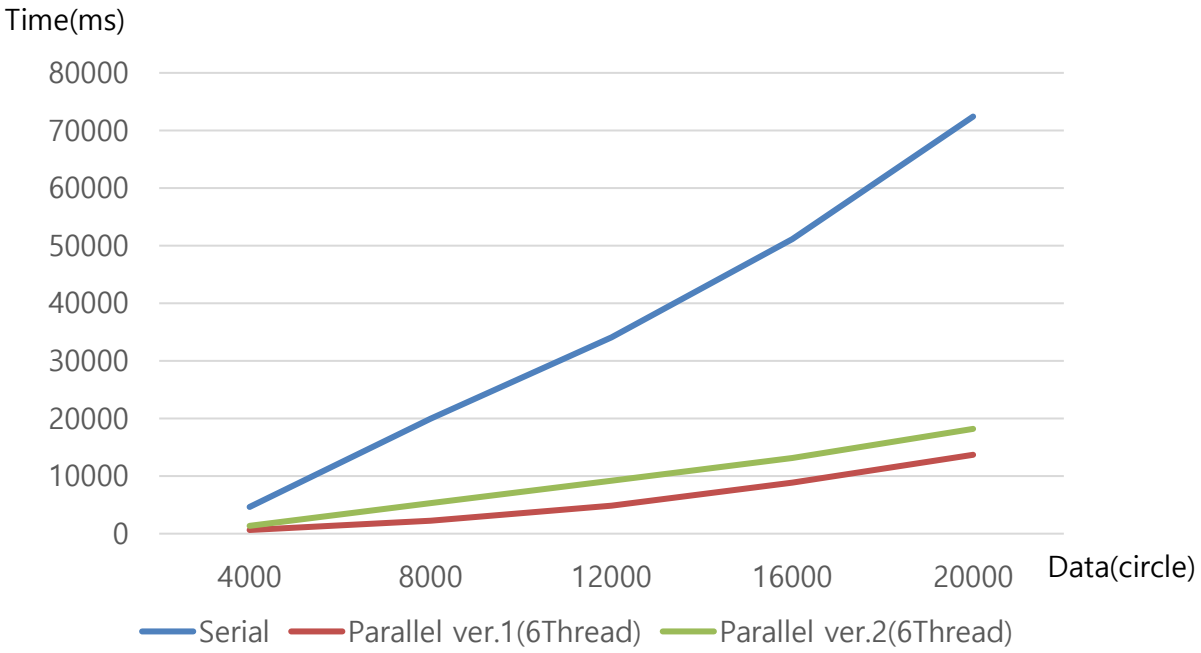


Parallel ver.1(6Thread)
데이터 량 2000
Serial 대비 성능향상 약 5.68배
데이터 량 10000
Serial 대비 성능향상 약 6.976배

Parallel ver.2(6Thread)
데이터 량 2000
Serial 대비 성능향상 약 2.379배
데이터 량 10000
Serial 대비 성능향상 약 3.815배

결과 분석

- circle형 데이터 량 별 소모 시간



Parallel ver.1(6Thread)
데이터 량 4000
Serial 대비 성능향상 약 7.718배
데이터 량 20000
Serial 대비 성능향상 약 6.115배

Parallel ver.2(6Thread)
데이터 량 4000
Serial 대비 성능향상 약 5.283배
데이터 량 20000
Serial 대비 성능향상 약 3.978배

THANK YOU

05

출처