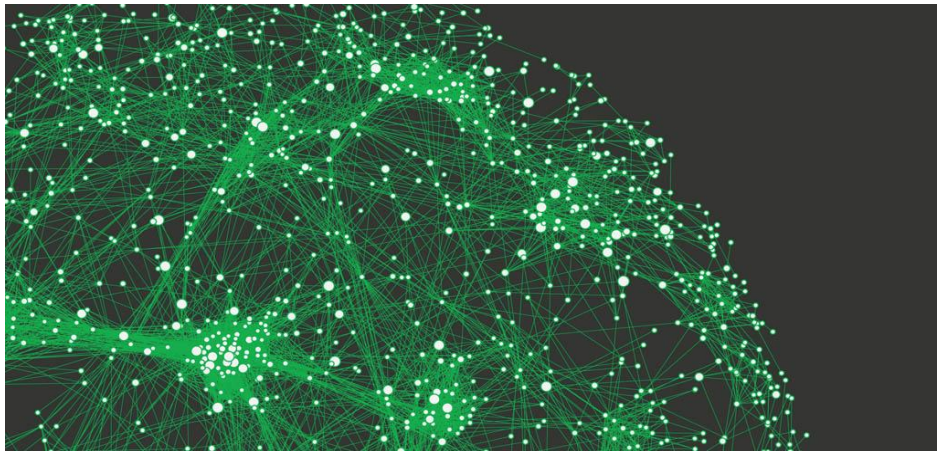
Algoritmo K-Means in MPI



Simone Contini

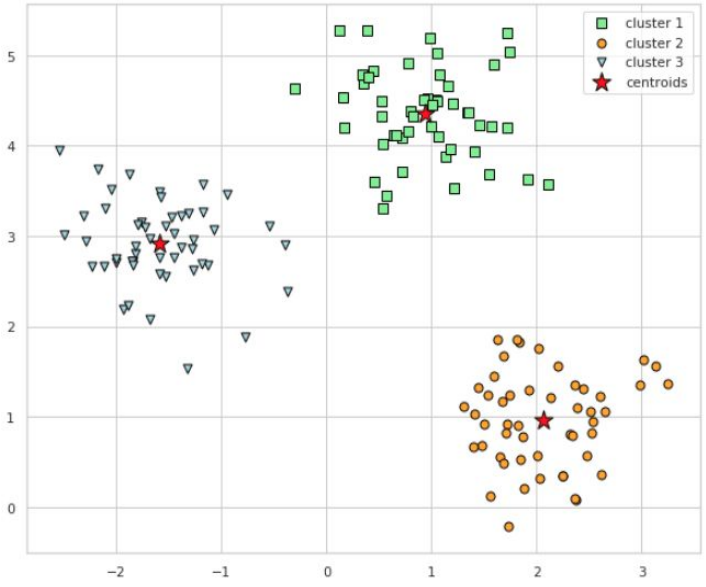
Universitá degli Studi di Palermo

Cos'è il K-Means

Il **K-Means** è un algoritmo di apprendimento non supervisionato per il raggruppamento di data point simili in un dato numero *K* di cluster.

I **cluster** rappresentano i gruppi che dividono gli oggetti a seconda della presenza o meno di una certa somiglianza tra di loro.

Per ogni cluster si definisce un **centroide**, ossia un data point situato nella posizione centrale di un cluster



Implementazione seriale

1. Scelta del numero di data point e del numero *K* di cluster in cui si vuole suddividere il dataset stesso;
2. Selezione casuale di *K* centroidi appartenenti allo spazio delle features;
3. Calcolo della distanza di ogni data point rispetto ad ogni centroide;
4. Associazione di ciascun data point al cluster collegato al centroide più vicino;
5. Update della posizione di ogni centroide in base alla media delle posizioni di tutti i data point del cluster associato;
6. Iterazione dal punto 3 fino a quando non ci sarà più alcuna modifica dei centroidi.

Misura delle distanze

Per la misura delle distanze è stata utilizzata una matrice delle distanze, in cui in ciascuna cella viene calcolata la **distanza euclidea** tra un data point ed un centroide.

Dati i punti $x_1, x_2, ..., x_n$ in uno spazio *k* dimensionale $\mathbb{R}^k$, i valori della matrice *A* delle distanze euclidee sono calcolati dai quadrati delle distanze di questi punti:

$$A = (a_{ij})$$

$$a_{ij} = d_{ij}^2 = ||x_i - x_j||^2$$

$$A = \begin{bmatrix} 0 & d_{12}^2 & d_{13}^2 & \dots & d_{1n}^2 \\ d_{21}^2 & 0 & d_{23}^2 & \dots & d_{2n}^2 \\ d_{31}^2 & d_{32}^2 & 0 & \dots & d_{3n}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{n1}^2 & d_{n2}^2 & d_{n3}^2 & \dots & 0 \end{bmatrix}$$

Ciascun data point sarà, dunque, assegnato al centroide la cui distanza risulti minima:

$$\underset{c_i \in C}{\arg\min}\ dist(c_i,\ x)^2$$

dove:

- c_i è un centroide nell'insieme *C* dei centroidi
- *x* sono i data point

Update dei centroidi

Per l'update dei centroidi è stata calcolata la media di tutti i data point che sono stati assegnati al cluster:

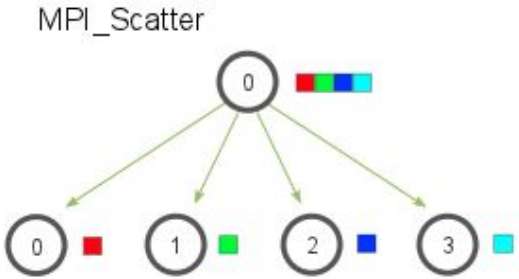
$$c_i = \frac{1}{n_i} \sum_{x_i \epsilon S_i} x_i$$

con S_i l'insieme dei data point assegnati al cluster *i*-esimo e n_i il numero di data point assegnati a questo cluster.

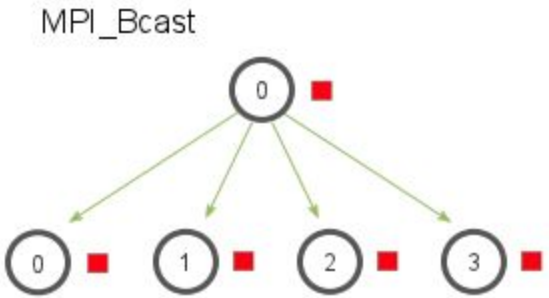
Implementazione parallela

Considerati *n* processori da utilizzare nella parallelizzazione, l'implementazione MPI consiste nei seguenti passaggi:

1. Il Master suddivide il set di data point in *n* chunk, ciascuno dei quali avente dimensione *sizeChunk = numDataPoints / n*, con *numDataPoints* il numero totale di data point nel set di dati. Quindi, distribuisce gli *n* chunk tra i vari rank del comunicatore (**Scattering**)

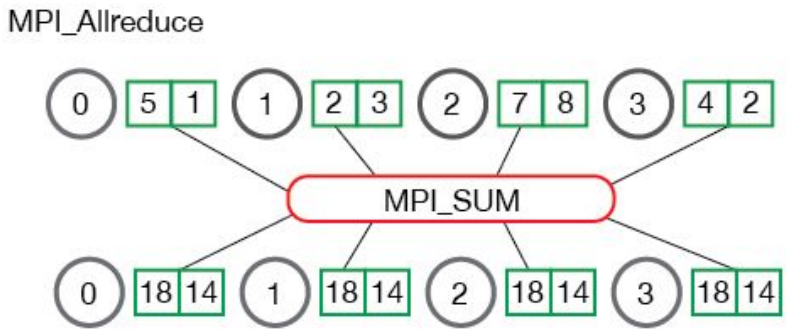


2. Il Master inizializza casualmente *K* centroidi appartenenti allo spazio delle features e manda una copia esatta di ciascuno di essi a tutti i rank del comunicatore (**Broadcasting**)

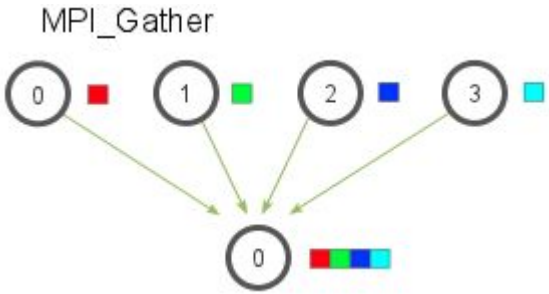


3. Ciascun processo (Master compreso) calcola la matrice delle distanze tra i data point del chunk assegnato ed i *K* centroidi. Ciascun data point, quindi, viene associato al cluster collegato al centroide più vicino e memorizzato in un array *clusters*.

4. Ciascun processo calcola il numero di data point assegnato in ciascun cluster per quel processo. Viene, dunque, effettuata un'operazione di **allReduce** che permette di ottenere in ciascun processo la somma totale dei data point assegnati a ciascun cluster di tutti i processi

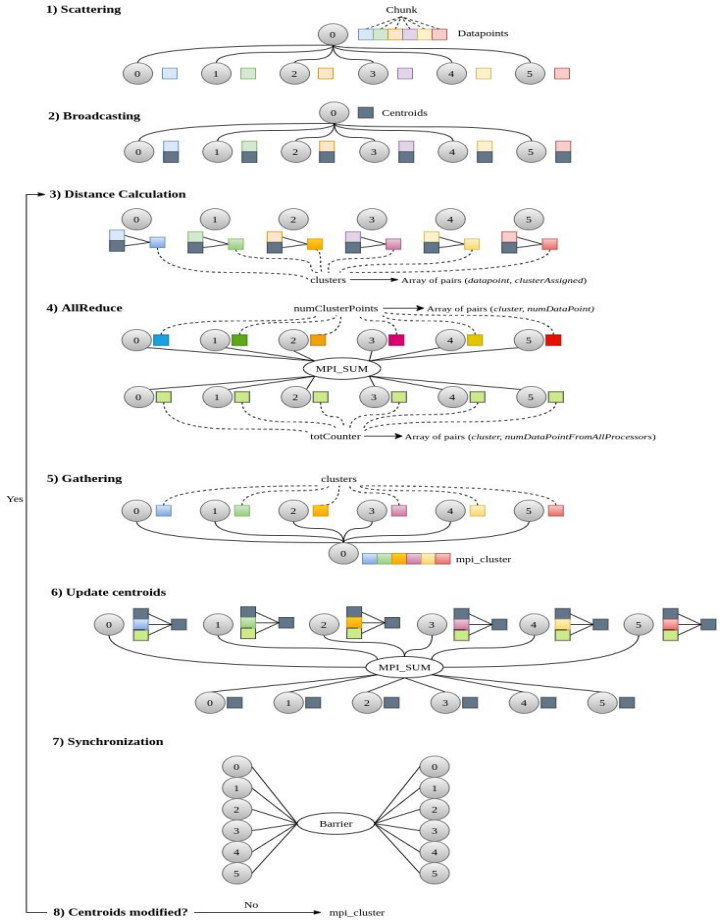


5. Gli array *clusters* di ciascun processo vengono uniti in un unico array *mpi_cluster* nel Master (**Gathering**)



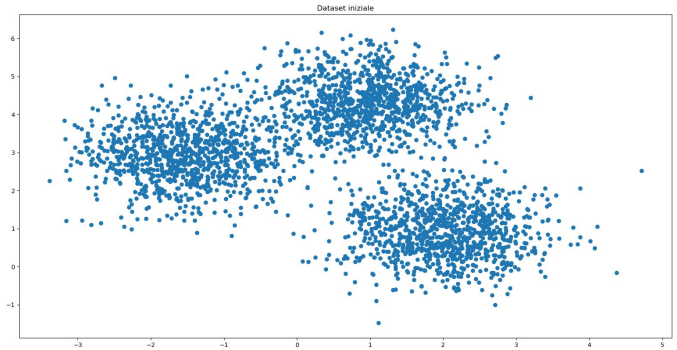
6. Ciascun processo effettua l'update dei centroidi con il proprio chunk di dati
7. Operazione di sincronizzazione dei task (**Barrier**)
8. Ciascun processo itera dal punto 3 fino a quando non ci sarà più alcuna modifica dei centroidi

Schema Implementazione parallela

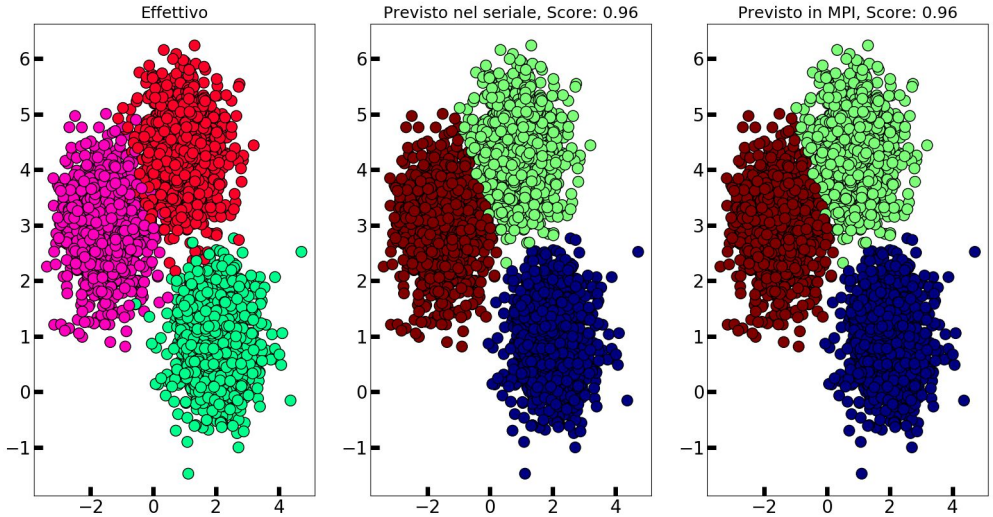


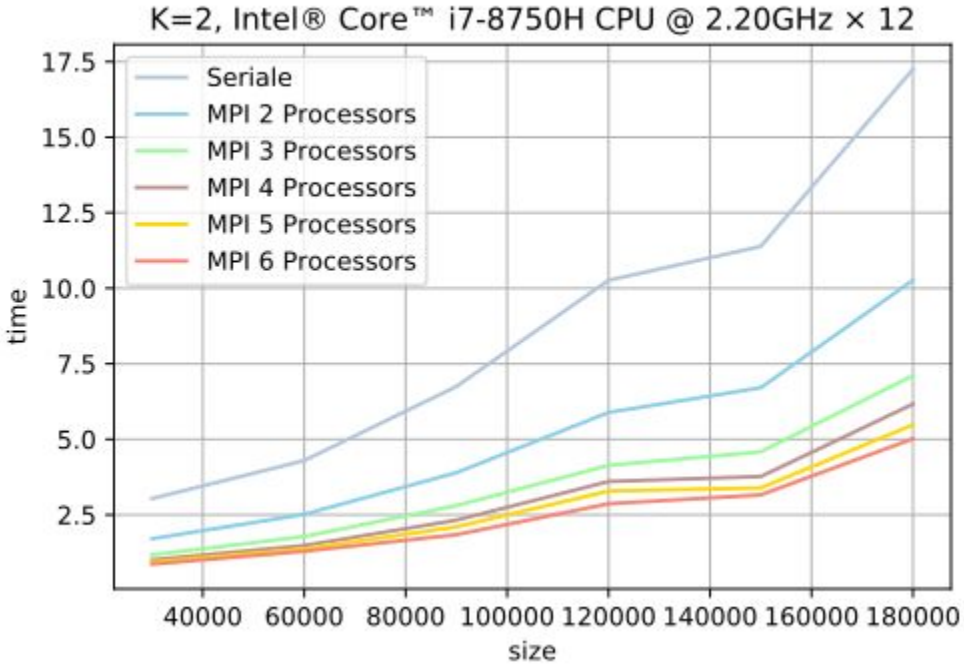
Risultati

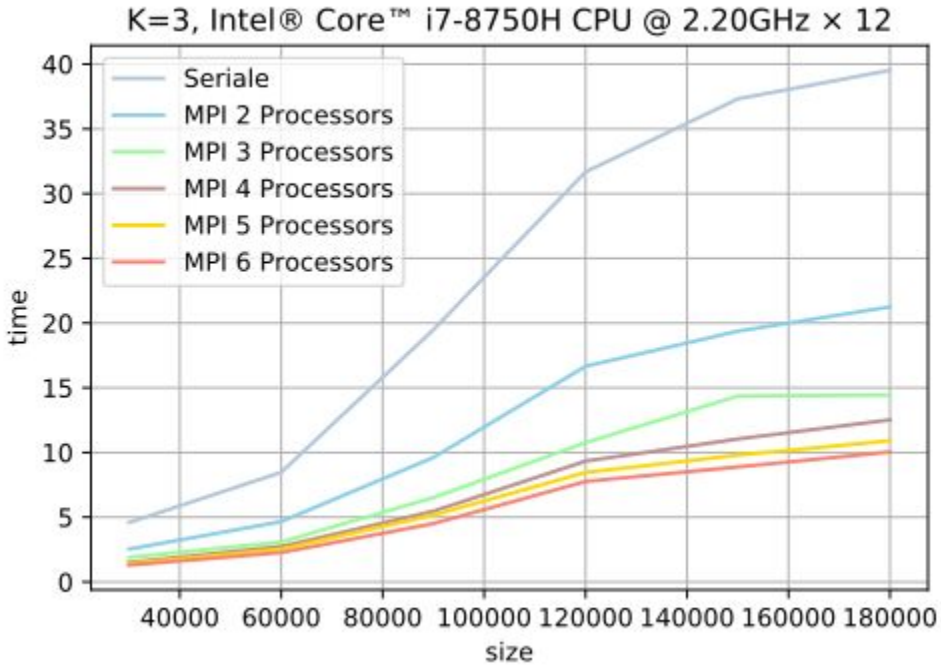
Per ciascun esperimento è stato generato randomicamente un dataset prendendo in input il numero di data point, il numero di cluster e la deviazione standard. Lo stesso dataset generato è stato utilizzato sia nell'implementazione seriale sia nell'implementazione parallela.

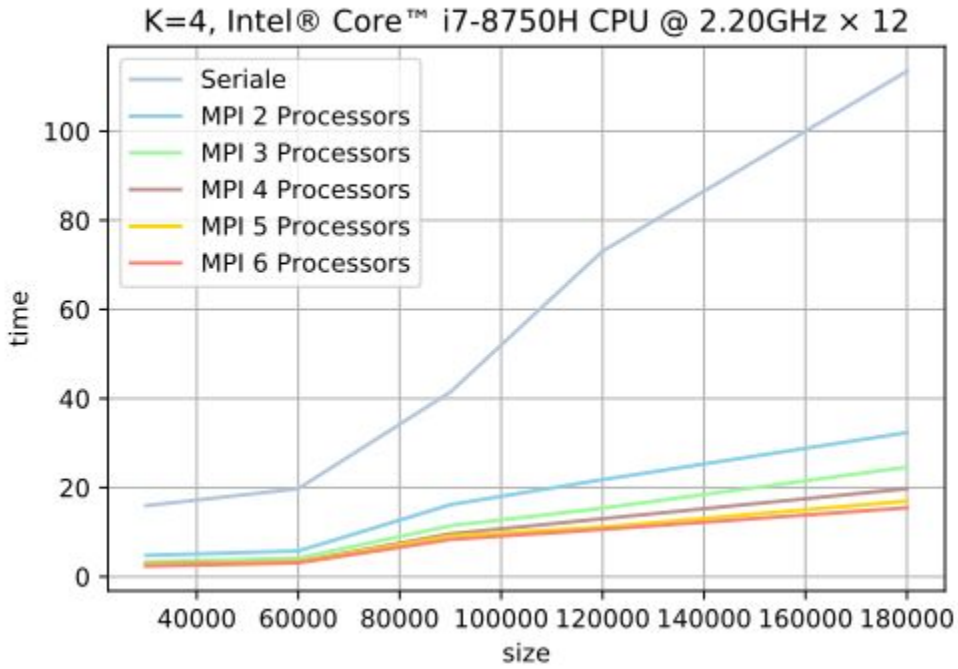


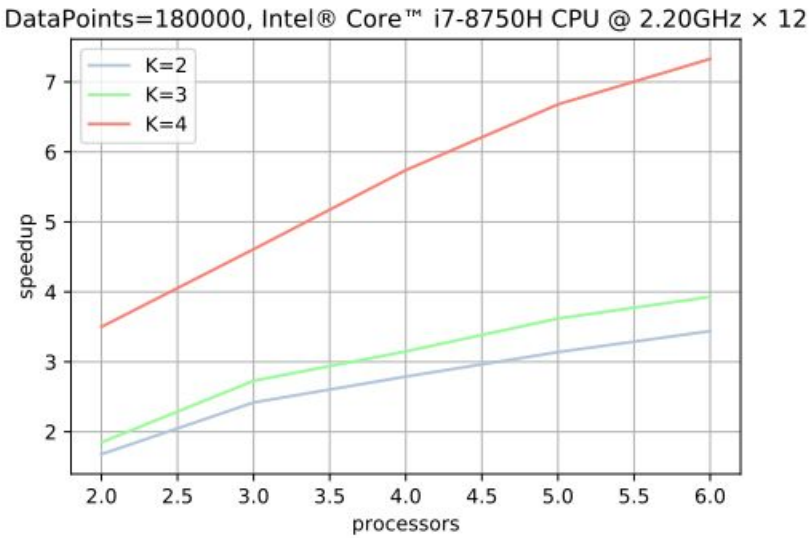
Numero data point: 3000

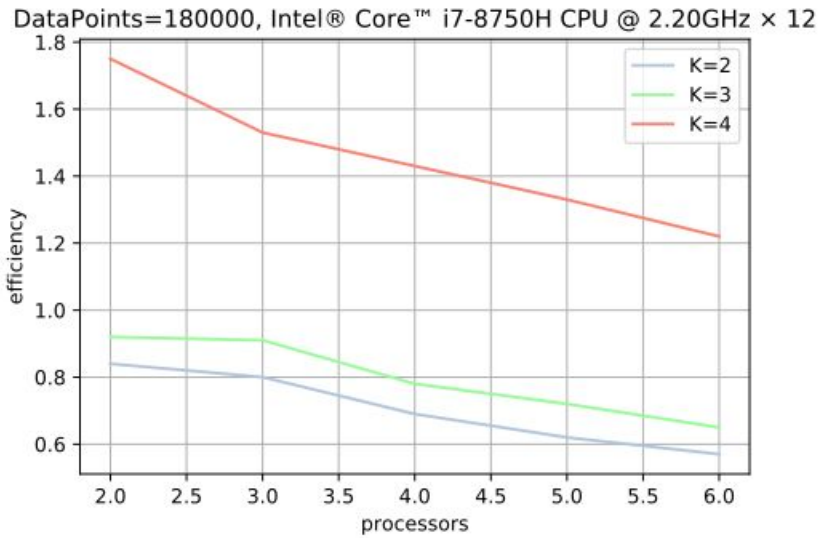


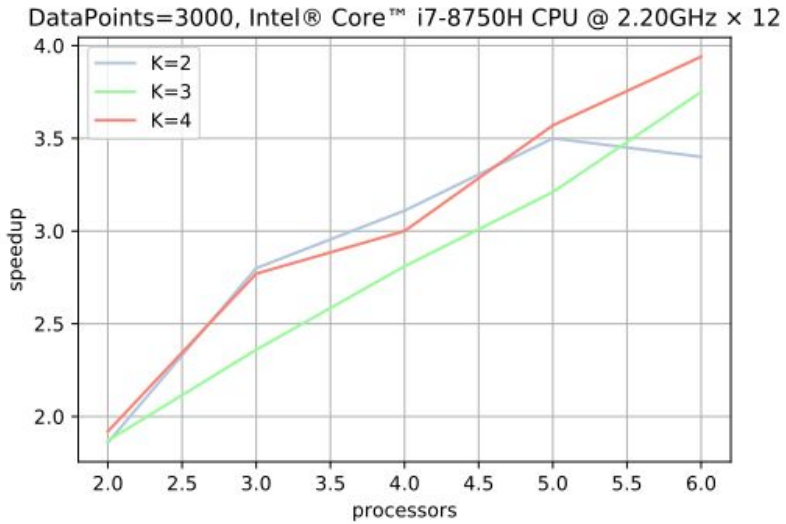

K=2, Intel® Core™ i7-8750H CPU @ 2.20GHz × 12
Seriale
MPI 2 Processors
MPI 3 Processors
MPI 4 Processors
MPI 5 Processors
MPI 6 Processors
time
size
17.5
15.0
12.5
10.0
7.5
5.0
2.5
40000
60000
80000
100000
120000
140000
160000
180000

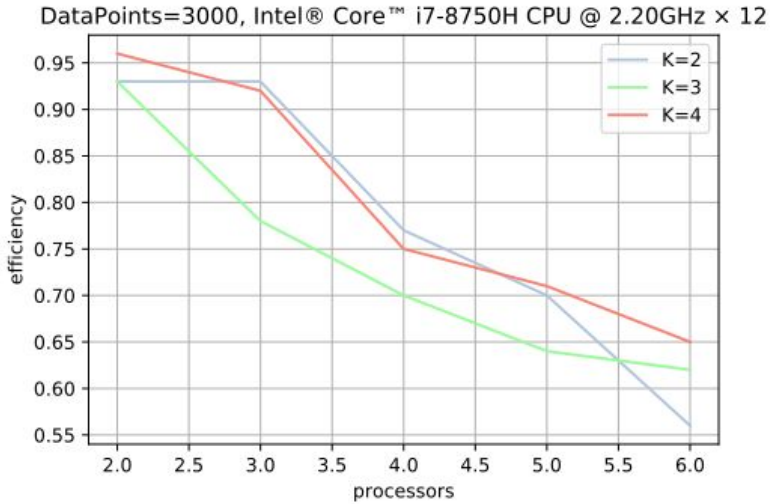

K=3, Intel® Core™ i7-8750H CPU @ 2.20GHz × 12
Seriale
MPI 2 Processors
MPI 3 Processors
MPI 4 Processors
MPI 5 Processors
MPI 6 Processors
time
size
0
5
10
15
20
25
30
35
40
40000
60000
80000
100000
120000
140000
160000
180000


K=4, Intel® Core™ i7-8750H CPU @ 2.20GHz × 12
Seriale
MPI 2 Processors
MPI 3 Processors
MPI 4 Processors
MPI 5 Processors
MPI 6 Processors
time
0
20
40
60
80
100
40000
60000
80000
100000
120000
140000
160000
180000
size


DataPoints=180000, Intel® Core™ i7-8750H CPU @ 2.20GHz × 12
K=2
K=3
K=4
speedup
processors
2
3
4
5
6
7
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0


DataPoints=180000, Intel® Core™ i7-8750H CPU @ 2.20GHz × 12
K=2
K=3
K=4
efficiency
processors
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0


DataPoints=3000, Intel® Core™ i7-8750H CPU @ 2.20GHz × 12
K=2
K=3
K=4
speedup
processors
4.0
3.5
3.0
2.5
2.0
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0


DataPoints=3000, Intel® Core™ i7-8750H CPU @ 2.20GHz × 12
K=2
K=3
K=4
efficiency
processors
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0