Clustering K-Means

Implementazione e risultati in modalità sequenziale e parallela con CUDA

Alessandro Lemmo

Firenze 2 luglio 2020

Introduzione

Funzionamento clustering k-means

- Il clustering k-means è un metodo che ha come obiettivo quello di partizionare n osservazioni in k clusters.
- Ogni punto viene assegnato ad un cluster sulla base del centroide ad esso più vicino.
- Il risultato è il partizionamento dello spazio dei dati in celle di Voronoi.

Diagramma di Voronoi

Introduzione

Aspetti teorici

- Lo scopo è quello di andare a minimizzare per ogni cluster, la distanza tra tutti i punti ad esso appartenenti ed il centroide.

$$S = \{S_1, S_2, ..., S_k\}$$

$$arg \min_S \sum_{i=1}^{k} \sum_{x \in S_i} d(x, u_i)$$

- Punti che costituiscono il cluster i-esimo all'iterazione t.

$$S_i^{(t)} = \left\{ x_p : d\left(x_p, m_i^{(t)}\right) \leq d\left(x_p, m_j^{(t)}\right) \ \forall j, \ 1 \leq j \leq k \right\}$$

Introduzione

Aspetti implementativi

- È stata realizzata l'implementazione del clustering k-means con punti nello spazio in tre dimensioni, con due diverse modalità
  - Sequenziale
  - Parallela con CUDA (Computed Unified Device Architecture) la quale è una piattaforma per la computazione parallela sviluppata da Nvidia.

- Modalità risolutiva iterativa per entrambe le versioni composta da tre fasi
  - Inizializzazione
  - Assegnazione
  - Aggiornamento

Versione sequenziale

Inizializzazione

- Definizione di tre vettori
  - Il primo contenente per ogni punto da clusterizzare l'indice del cluster di appartenenza. Inizialmente tutti posti a -1.
  - Il secondo contenente per ogni cluster il numero di punti ad esso assegnati.
  - Il terzo contenente i risultati, inizializzato con i k centroidi dei k clusters in coincidenza dei primi k punti.

```
01.  for i = 0 to total num of points
02.      clusters[i] = -1
03.  end
04.
05.  for i = 0 to total num of clusters
06.      clusterCounts [i] = 0
07.      result[i] = points [i]
08.      clusters[i] = i
09.  end
```

Versione sequenziale

Assegnazione

- Per ogni punto viene calcolata la distanza euclidea tra esso e tutti i centroidi.
  - Il centroide rispetto al quale risulta minore sarà quello a cui sarà associato il punto.

- Una volta effettuata l'operazione per tutti i punti si ha:
  - Il vettore contenente per ogni punto l'indice del cluster di appartenenza completo.
  - Il vettore contenente per ogni centroide il numero di punti assegnati completo.

```
01.  for i = 0 to total num of points
02.
03.      for j = 0 to total num of clusters
04.
05.          EuclideanDistance(punto i, centroide j)
06.
07.          if distance < min_distance
08.              best_cluster = j
09.      end
10.
11.      if best_cluster != actual_cluster[i]
12.          actual_cluster[i] = best_cluster
13.
14.      cluster_counts[best_cluster]++
15.  end
```

Versione sequenziale

Aggiornamento

- Per ogni cluster:
  - Vengono presi tutti i punti appartenenti ad uno stesso cluster e sommati componente per componente.
  - Le somme delle componenti x, y, e z vengono divise per il numero totale di punti appartenenti a tale cluster.
  - Si ottiene le nuove coordinate del centroide.

```
01. for i = 0 to total num of points
02.
03.     result[actual_cluster[i]] += points[i]
04.
05. for i = 0 to total num of clusters
06.
07.     result[i] /= cluster_counts[i]
08.
```

- Si itera ripetendo la fase di assegnazione.

- Si termina quando il numero di punti a cui viene cambiato il cluster di appartenenza è pari a zero
  - Ciò è interpretato come una convergenza dei centroidi ai posizionamenti ottimi.

Versione Parallela

Aspetti implementativi

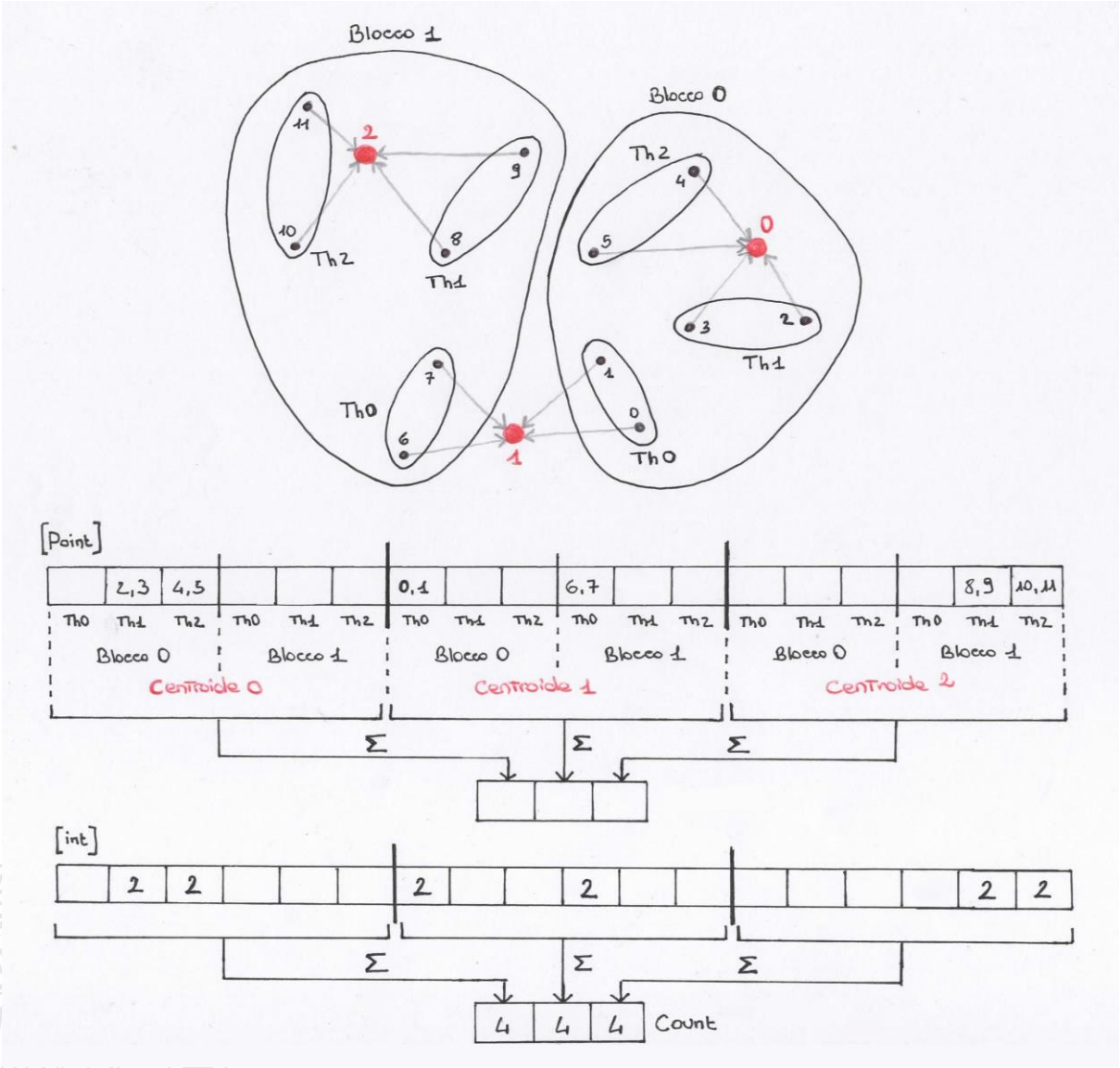
- Basa la logica di realizzazione del clustering k-means esattamente sugli stessi principi descritti nella vesione sequenziale.

- Usati due kernel per la parallelizzazione del codice
  - Il primo gestisce la fase di assegnazione dei punti ai clusters.
  - Il secondo gestisce la divisione per il numero totale di punti per stabilire il nuovo centroide.

- La somma delle coordinate componente per componente dei punti è gestita tramite la libreria Thrust
  - Basata sulla Standard Template Library permette di implementare operazioni in parallelo.

- Suddivisione dei compiti nei kernel sulla base del numero di blocchi (e quindi di thread) usati per gestirle.

Versione Parallela Primo kernel

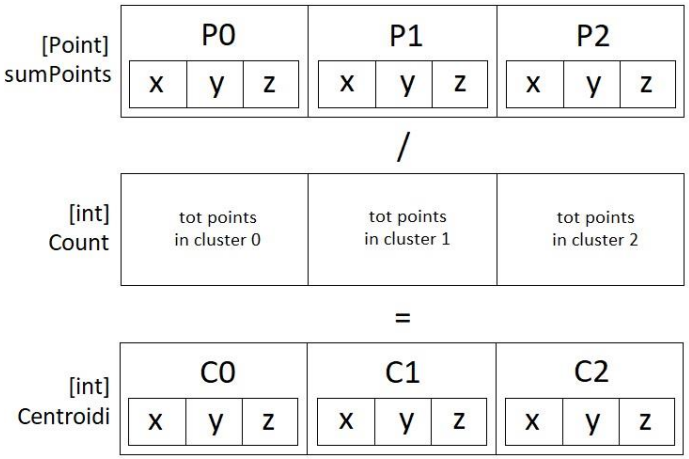


- Calcolo distanza euclidea
- Assegnazione punto al centroide più vicino
- Incremento contatore
- Riduzione vettori

Versione Parallela

Secondo kernel

- Si occupa di calcolare le coordinate dei nuovi centroidi.
- Divisione membro a membro e componente per componente dei due vettori calcolati nel primo kernel.

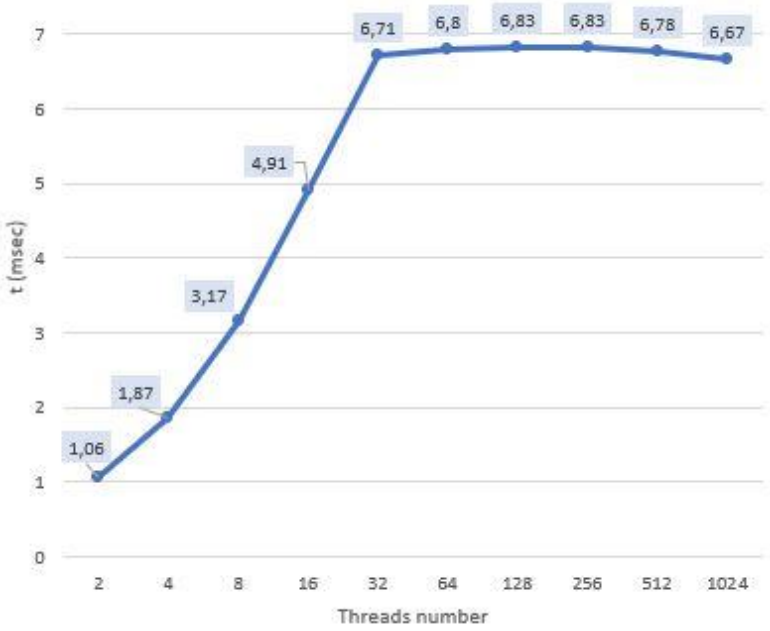


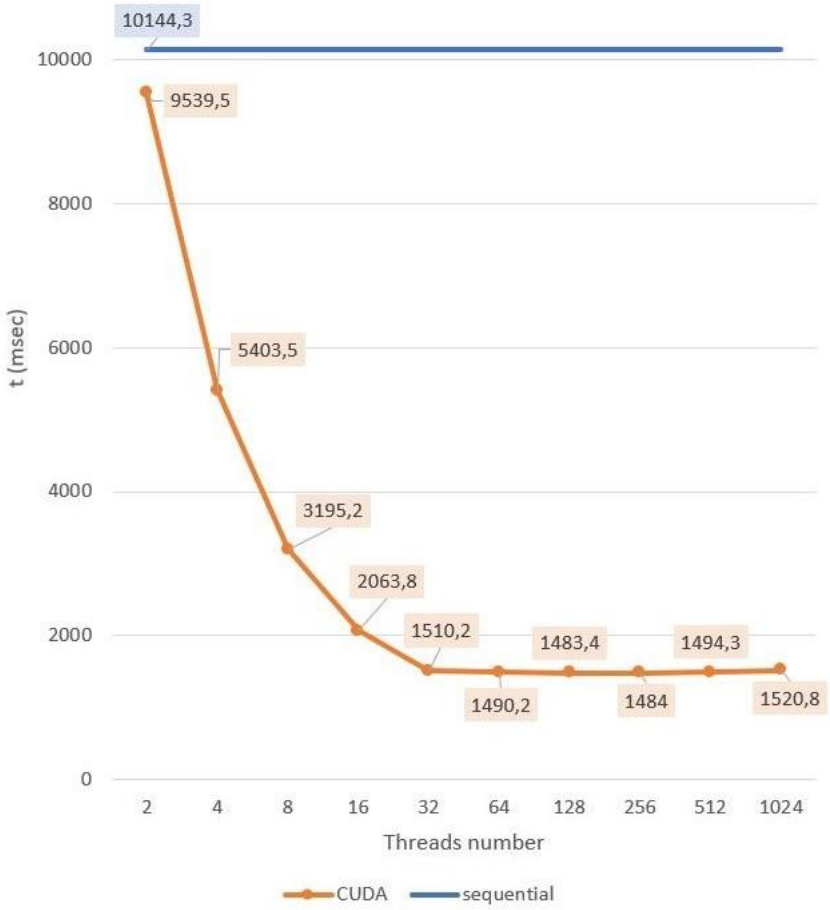
- Verifica criterio di arresto.

Risultati

Variazione threads

- 10 clusters
- 1000000 di punti
- Variazione threads da 2 a 1024
- Valore ottimale a 32 threads

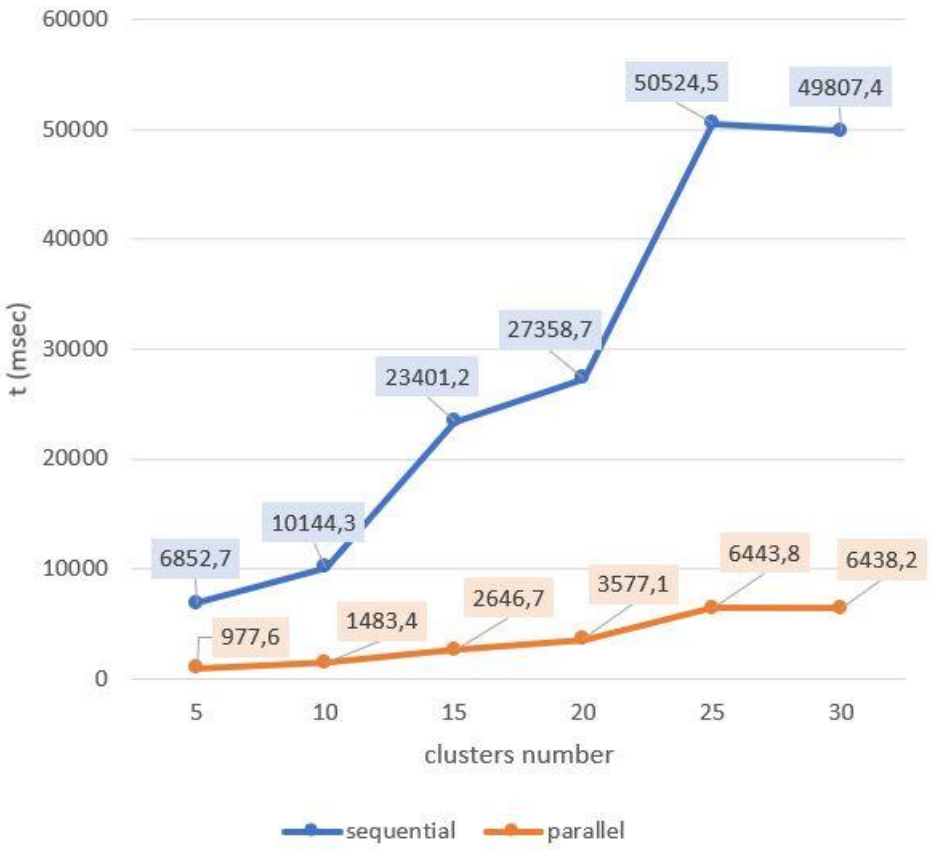




Risultati

Variazione clusters

- 128 threads
- 1000000 di punti
- Variazione clusters da 5 a 30
- Andamento altalenante dovuto al fatto che un numero maggiore di clusters non porta necessariamente ad un numero maggiore di iterazioni e quindi ad un tempo più alto.



Risultati

Variazione punti

- 128 threads
- 10 clusters
- Variazione punti da 100 a 1000000
- Versione parallela migliore da 100000 punti