Mean Shift Clustering

Leonardo Del Bene
leonardo.delbene@edu.unifi.it

Abstract

In questo lavoro, è stato analizzato l'algoritmo di Mean Shift Clustering, implementandolo sia in versione sequenziale che parallela. La parallelizzazione è stata ottenuta utilizzando OpenMP per sfruttare l'hardware multi-core. I test sono stati eseguiti su immagini di dimensioni variabili per confrontare i tempi di esecuzione e valutare lo speedup e l'efficienza dell'algoritmo parallelo. I risultati hanno dimostrato che l'algoritmo beneficia significativamente della parallelizzazione, con un notevole miglioramento delle performance in termini di tempo di esecuzione e un andamento dello speedup che tende a essere superlineare per configurazioni con meno thread e immagini di dimensioni maggiori. Tuttavia, l'efficienza diminuisce con l'aumento dei thread, indicando un punto di saturazione nelle risorse computazionali disponibili. Questi risultati confermano l'efficacia della parallelizzazione per l'applicazione su dataset di dimensioni maggiori.

1. Introduzione

Il Mean Shift Clustering è un algoritmo non parametrico per l'analisi di dati, ampiamente utilizzato in diversi campi, come la computer vision, il riconoscimento di pattern e la segmentazione di immagini. A differenza di metodi come il k-means, il Mean Shift non richiede di specificare in anticipo il numero di cluster. Invece, si basa su una procedura iterativa che identifica le modalità della distribuzione di densità dei dati e assegna ogni punto al cluster più vicino.

L'algoritmo sfrutta un approccio basato su kernel, dove ciascun punto dati si sposta progressivamente verso la densità più alta calcolata localmente. Il risultato è una suddivisione dei dati in cluster che riflettono le modalità naturali della distribuzione. Questa caratteristica lo rende particolarmente efficace in contesti in cui la struttura dei dati è complessa o non lineare.

Tuttavia, l'algoritmo nella sua implementazione sequenziale soffre di alti costi computazionali, specialmente su dataset di grandi dimensioni. In questa relazione, affrontiamo questo problema sviluppando una versione parallela dell'algoritmo, progettata per sfruttare le moderne architetture hardware multi-core. Attraverso l'implementazione delle due versioni sequenziale e parallela analizziamo le performance in termini di tempo di esecuzione e calcoliamo lo *speedup*, definito come il rapporto tra il tempo di esecuzione della versione sequenziale e quello della versione parallela.

Questa analisi offre spunti sulla scalabilità e sull'efficacia della parallelizzazione dell'algoritmo, fornendo indicazioni utili per applicazioni future.

2. Algoritmo di Mean Shift Clustering

Il Mean Shift Clustering è un algoritmo iterativo non parametrico che identifica le modalità nella distribuzione di densità di un dataset. L'obiettivo è spostare iterativamente ciascun punto dati verso la densità locale più alta, utilizzando una finestra definita da un kernel.

2.1. Descrizione dell'algoritmo

L'algoritmo può essere suddiviso nei seguenti passi principali:

1. **Inizializzazione**: Ogni punto dati del dataset è considerato come un centro iniziale.

2. **Calcolo della densità locale**: Per ciascun punto x, si calcola la media pesata dei punti vicini utilizzando una finestra definita da un kernel K. La nuova posizione x' del punto è data da:

$$x' = \frac{\sum_{i=1}^{N} K(x - x_i) \cdot x_i}{\sum_{i=1}^{N} K(x - x_i)}$$

   dove x_i sono i punti vicini e K è un kernel, tipicamente gaussiano.

3. **Aggiornamento**: Ogni punto x viene spostato nella nuova posizione x' calcolata.

4. **Iterazione**: I passi precedenti vengono ripetuti fino alla convergenza, ossia quando lo spostamento $||x' - x||$ è inferiore a una soglia definita.

2.2. Vantaggi e limitazioni

L'algoritmo è in grado di individuare cluster di forma arbitraria senza richiedere il numero di cluster come parametro. Tuttavia, presenta alcune limitazioni:

- L'accuratezza e il numero di cluster dipendono dalla scelta del parametro di banda h (bandwidth) del kernel. Un valore maggiore di h tende a produrre un minor numero di cluster, fondendo i punti vicini in gruppi più grandi, mentre un valore minore di h aumenta il numero di cluster, distinguendo maggiormente i dettagli locali della distribuzione dei punti.

- L'implementazione sequenziale ha una complessità temporale elevata, pari a $O(N^2)$, dove N è il numero di punti dati.

2.3. Pseudocodice

Di seguito, viene riportato lo pseudocodice dell'algoritmo di Mean Shift:

Require: Dataset $\{x_1, x_2, \ldots, x_N\}$, kernel K, banda h
Ensure: Cluster dei dati
1: **for** ogni punto x nel dataset **do**
2: **repeat**
3: Calcola $x' = \frac{\sum_{i=1}^{N} K(x-x_i)\cdot x_i}{\sum_{i=1}^{N} K(x-x_i)}$
4: Aggiorna $x \leftarrow x'$
5: **until** Convergenza
6: **end for**
7: Assegna i punti al cluster più vicino

Questa descrizione fornisce una base per l'implementazione sia nella versione sequenziale che in quella parallela.

3. Implementazione

L'implementazione dell'algoritmo di Mean Shift Clustering è stata realizzata in linguaggio C++, utilizzando strutture dati e librerie per la gestione di immagini e operazioni matematiche. Il codice carica un'immagine, converte i pixel in punti con coordinate spaziali e colori (`x, y, r, g, b`), ed esegue il clustering basato sul kernel gaussiano.

Dettagli dell'implementazione:

- Ogni pixel è rappresentato da una struttura `Point`, che memorizza le coordinate spaziali e i valori di colore.

- Per ogni punto, viene iterativamente calcolato un nuovo punto medio utilizzando il kernel gaussiano:

$$x' = \frac{\sum_{i=1}^{N} K(\text{dist}(x, x_i)) \cdot x_i}{\sum_{i=1}^{N} K(\text{dist}(x, x_i))}$$

dove $K(\cdot)$ è il kernel gaussiano definito come:

$$K(d) = \exp\left(-\frac{d^2}{2h^2}\right)$$

e h è il parametro di banda (*bandwidth*).

- Il clustering prosegue fino alla convergenza, determinata da un valore di tolleranza (`tol`).

- L'immagine risultante viene ricostruita assegnando a ciascun pixel il colore del punto finale del suo cluster.

- Viene inoltre calcolato il numero totale dei cluster e il numero di punti che sono stati assegnati ad ogni cluster.

Esecuzione: Il programma esegue il clustering su un'immagine di input, salvandone una versione segmentata. Ad esempio:

```
Input:  img/input60.png
Output: img/output60.png
```

Il tempo di esecuzione totale viene misurato utilizzando il supporto delle librerie `std::chrono`.

Output L'output dell'algoritmo consiste in una versione segmentata dell'immagine di input e nel numero di cluster individuati nell'immagine. In Figura 3 viene mostrato un esempio di input e output dell'algoritmo applicato a un'immagine di dimensione 60×60. La versione segmentata evidenzia i gruppi di pixel con caratteristiche simili in termini di colore e posizione.

Di seguito è riportato il numero di cluster generati dall'algoritmo, insieme ai dettagli relativi al centroide di ciascun cluster e al numero di pixel appartenenti a ciascun gruppo:

- **Cluster 1**: Centroide (X: 15, Y: 42, R: 230, G: 215, B: 199) $->$ **Popolazione**: 1238 pixel

- **Cluster 2**: Centroide (X: 18, Y: 10, R: 120, G: 77, B: 58) $->$ **Popolazione**: 402 pixel

- **Cluster 3**: Centroide (X: 24, Y: 30, R: 222, G: 180, B: 149) $->$ **Popolazione**: 507 pixel

- **Cluster 4**: Centroide (X: 31, Y: 25, R: 61, G: 68, B: 57) $->$ **Popolazione**: 411 pixel

- **Cluster 5**: Centroide (X: 38, Y: 14, R: 143, G: 155, B: 126) $->$ **Popolazione**: 211 pixel

- **Cluster 6**: Centroide(X: 50 , Y: 28, R: 241, G: 237, B: 228) $->$ **Popolazione**: 831 pixel

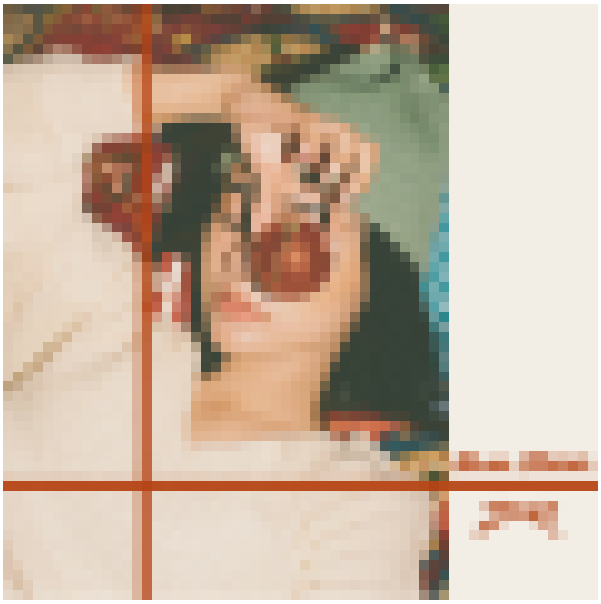

Figure 1. Immagine di input.

Figure 2. Immagine segmentata (output).

Figure 3. Confronto tra l'immagine originale e quella segmentata dall'algoritmo di Mean Shift Clustering.

3.1. Versione Sequenziale

Punti di forza:

- L'implementazione sequenziale è semplice da comprendere e debug.
- L'algoritmo calcola risultati accurati grazie alla somma ponderata basata sul kernel.

Debolezze:

- **Complessità temporale**: Con una complessità di $O(N^2)$, l'algoritmo è computazionalmente oneroso per dataset di grandi dimensioni.
- **Bassa scalabilità**: L'algoritmo non sfrutta l'hardware multi-core, rendendolo inefficiente per immagini di alta risoluzione.

3.2. Versione Parallela

Per migliorare le performance, l'algoritmo di Mean Shift Clustering è stato parallelizzato utilizzando OpenMP. La parallelizzazione si basa sul calcolo indipendente dello spostamento per ciascun punto del dataset, sfruttando il parallelismo intrinseco dell'algoritmo; inoltre, viene eseguita anche la parallelizzazione per il caricamento delle immagini e la loro ricostruzione, ottimizzando così le prestazioni complessive. Questo approccio consente di suddividere il lavoro tra i core disponibili del processore, riducendo i tempi di elaborazione e migliorando l'efficienza del sistema.

Dettagli dell'implementazione:

- **Parallelizzazione dei punti**: Ogni punto viene elaborato in modo indipendente all'interno di un ciclo `#pragma omp parallel for`, distribuendo il carico di lavoro tra i thread disponibili.
- **Riduzioni**: Per calcolare i contributi ponderati (*weighted sum*) dei punti vicini, sono state utilizzate le riduzioni parallele su variabili scalari tramite il costrutto `#pragma omp reduction`.
- **Controllo della convergenza**: Ogni thread verifica la convergenza del proprio punto utilizzando una soglia predefinita (`tol`), riducendo i cicli inutili.
- **Mappa dei cluster**: Per contare il numero dei punti nei cluster, è stato implementato un meccanismo di mappa locale per ciascun thread, combinata successivamente in una mappa globale utilizzando blocchi critici (`omp_lock`).

4. Setup dell'esperimento

Per valutare le performance delle implementazioni sequenziale e parallela dell'algoritmo di Mean Shift Clustering, sono stati utilizzati i seguenti strumenti hardware e software, insieme a un dataset specifico.

4.1. Hardware

I test sono stati eseguiti su un MacBook Pro con le seguenti specifiche tecniche:

- **Modello**: MacBook Pro 12,1.

- **Processore**: Intel Core i5 dual-core, 2,7 GHz.
- **Core e Hyper-Threading**: 2 core fisici, 4 thread grazie alla tecnologia Hyper-Threading.
- **Memoria**: 8 GB di RAM.

4.2. Librerie e Software

L'implementazione e i test sono stati realizzati utilizzando i seguenti strumenti:

- **Linguaggio di programmazione**: C++.
- **Librerie per la gestione delle immagini**: `stb_image.h` e `stb_image_write.h`.
- **Libreria di parallelizzazione**: OpenMP, utilizzata per sfruttare i thread disponibili.
- **Misurazione delle performance**: Utilizzo di `std::chrono` per misurare i tempi di esecuzione.

4.3. Parametri del Dataset

Il dataset utilizzato per i test è stato generato a partire da immagini RGB, trattando ogni pixel come un punto dati con caratteristiche spaziali e di colore. I principali parametri del dataset sono:

- **Numero di punti**: N = Larghezza $\times$ Altezza dell'immagine.
- **Dimensionalità dei punti**: Ogni punto è rappresentato da un vettore a 5 dimensioni (x, y, r, g, b).
- **Dimensioni delle immagini di input**: I test sono stati effettuati su cinque immagini di dimensioni crescenti: 40×40, 50×50, 60×60, 70×70 e 80×80, per un totale rispettivamente di 1600, 2500, 3600, 4900 e 6400 punti.
- **Parametro di banda (*bandwidth*)**: 20, utilizzato per il kernel gaussiano.
- **Tolleranza per la convergenza**: tol $= 10^{-3}$.
- **Numero massimo di iterazioni**: 300.

4.4. Obiettivo del Setup

L'obiettivo dell'esperimento è misurare il tempo di esecuzione delle versioni sequenziale e parallela, valutando lo *speedup* ottenuto dalla parallelizzazione. Per garantire accuratezza nei risultati, i test sono stati ripetuti più volte e i tempi medi sono stati utilizzati per il confronto.

5. Risultati ottenuti

In questa sezione vengono presentati i risultati ottenuti dall'esecuzione dell'algoritmo di Mean Shift Clustering, sia nella versione sequenziale che in quella parallela. I risultati sono analizzati per valutarne le performance in termini di tempo di esecuzione e speedup, con particolare attenzione all'efficacia della parallelizzazione.

I dati raccolti sono rappresentati sia in forma tabellare che attraverso grafici esplicativi, al fine di fornire una visione chiara e immediata delle differenze tra le due implementazioni. La tabella riporta i tempi di esecuzione per diverse configurazioni del dataset e il corrispondente speedup, mentre i grafici visualizzano il comportamento dello speedup in funzione del numero di punti e del numero di thread utilizzati.

L'obiettivo principale dell'analisi è evidenziare i vantaggi della parallelizzazione, mostrando come l'algoritmo parallelo riduca significativamente i tempi di esecuzione rispetto alla versione sequenziale, specialmente su dataset di grandi dimensioni.

5.1. Versione Sequenziale

Nella tabella 5.2 è possibile osservare i tempi di esecuzione (in ms) dell'algoritmo in modalità sequenziale, è riporatato anche il tempio medio di esecuzione.

5.2. versione parallela

La tabella 5.2 riporta i tempi di esecuzione della versione parallela dell'algoritmo di Mean Shift Clustering. I test sono stati eseguiti utilizzando configurazioni con un minimo di 2 thread per un massimo di 8, per analizzare l'impatto del numero di thread sulle performance. I dati raccolti mostrano i tempi medi di esecuzione per ogni configurazione. La media è stata effettuata su **10 esecuzioni**

La tabella 5.2 e la figura 4 mostrano come il numero di thread influisca sui tempi medi di esecuzione dell'algoritmo per diverse dimensioni di immagine. I risultati evidenziano che l'incremento dei thread riduce significativamente i tempi di esecuzione fino a 4 thread, con un beneficio maggiore per immagini più grandi (60×60 e superiori).

Oltre i 4 thread (limite massimo di thread che l'hardware su cui sono stati eseguiti i test può supportare contemporaneamente attivi), l'efficacia della parallelizzazione diminuisce, mostrando una saturazione delle risorse. Per immagini più piccole (40×40 e 50×50), il guadagno è meno evidente, poiché il dataset non sfrutta appieno il parallelismo. Il miglior compromesso tra riduzione dei tempi e utilizzo delle risorse si osserva con 4 thread, che offre un'elevata scalabilità senza introdurre un sovraccarico eccessivo.

Dimensione Immagine	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Media (ms)
40×40	6081	5481	5398	5454	5411	5343	5354	5366	5343	5343	5457
50×50	11200	11150	11080	11075	11084	11064	11087	11084	11103	11142	11107
60×60	31759	32292	31733	31692	32127	31799	31789	31627	31806	31485	31821
70×70	59277	59181	59170	59024	59030	60279	59837	58959	59045	59031	59283
80×80	88039	87986	88310	88124	90642	90470	90007	89838	88960	90422	89280

Table 1. Tempi di esecuzione (ms) della versione sequenziale su 5 immagini eseguite 10 volte ciascuna.

Dimensione Immagine	2 Thread	3 Thread	4 Thread	5 Thread	6 Thread	7 Thread	8 Thread
40×40	1926	1600	1428	1411	1520	1448	1450
50×50	3793	3192	2791	2802	2797	2804	2782
60×60	10611	8864	7909	7548	7641	7760	7704
70×70	19924	16532	14418	14430	14391	14470	14430
80×80	29647	24755	21475	21464	21638	21602	21563

Table 2. Tempi medi di esecuzione (ms) per ciascuna dimensione dell'immagine e configurazione di thread.

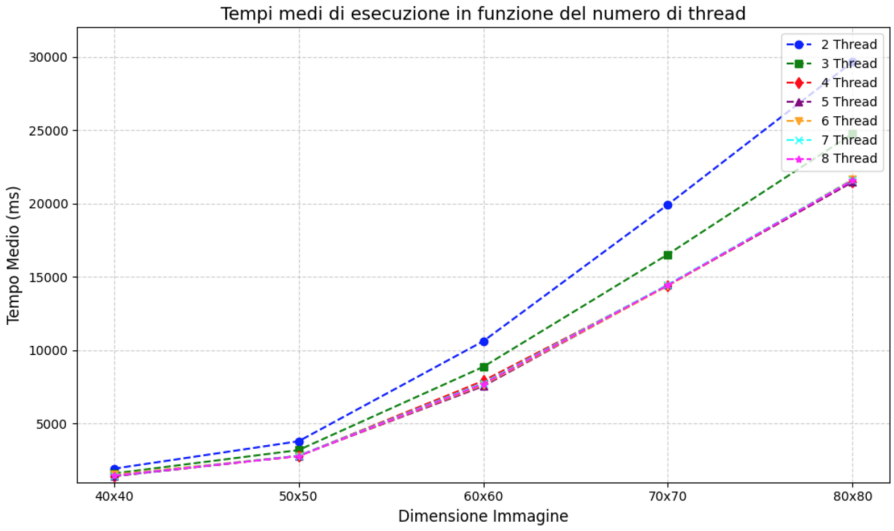

Figure 4. Grafico dei tempi medi, in funzione della dimensione dell'immagine, dell'esecuzione dell'algoritmo di Mean Shift Clustering

5.3. Speedup ed Efficienza

Lo *speedup* è una misura del miglioramento delle performance ottenuto dalla parallelizzazione dell'algoritmo rispetto alla sua versione sequenziale. Viene calcolato come il rapporto tra il tempo di esecuzione della versione sequenziale (T_{seq}) e il tempo della versione parallela (T_{par}):

$$\text{Speedup} = \frac{T_{\text{seq}}}{T_{\text{par}}}$$

Dai dati raccolti nelle tabelle 5.2 e 5.2 è possibile calcolare lo speedup nelle varie configurazioni. Per il calcolo dello speedup, è stato utilizzato il valore medio del tempo di esecuzione per ciascuna configurazione, ottenuto dalla media dei tempi di esecuzione delle 10 esecuzioni riportate nelle tabelle. In tabella 5.3 è stato mostrato lo speedup solo per le configurazioni con massimo 4 thread, per osservare lo speedup delle configurazioni con più di 4 thread si rimanda alle figure 5 e 6

Dim Img	4 thread	3 thread	2 thread
40×40	3.82	3.41	2.83
50×50	3.98	3.48	2.93
60×60	4.02	3.59	2.99
70×70	4.11	3.58	2.97
80×80	4.16	3.60	3.01

Table 3. Valori di Speedup per le diverse configurazioni di thread

Come possiamo vedere dalla tabella 5.3 e dalle figure 5 e 6 lo **speedup** della configurazione con **4 thread** migliora progressivamente con l'aumento della dimensione dell'immagine di input, mantenendo, per tutte le altre configurazioni, un andamento pressoché lineare. Con immagini di input ad alta dimensionalità, ci si aspetterebbe un comportamento dello **speedup super-lineare**, in quanto lo speedup tende ad aumentare con l'incremento del numero di pixel. Già con immagini di dimensione 70×70, lo speedup ha superato leggermente il valore di 4, che corrisponde al numero di thread, indicando che l'algoritmo sta beneficiando di una parallelizzazione efficace per immagini più grandi. Questo suggerisce che, con un ulteriore aumento della dimensione dell'immagine, lo speedup potrebbe continuare ad aumentare, a conferma di una maggiore scalabilità del sistema parallelo.
Con le configurazioni a **3 thread** e **2 thread**, notiamo che lo speedup ottiene fin da subito un **andamento super-lineare**. Anche in questo caso, lo speedup aumenta progressivamente con l'aumentare della dimensionalità dell'immagine di input. Questo comportamento super-lineare indica che l'efficienza della parallelizzazione dell'algoritmo migliora ulteriormente man mano che aumenta il numero dei pixel nell'immagine, permettendo così di sfruttare al meglio le risorse computazionali disponibili. L'andamento super-lineare si mantiene per tutte le configurazioni testate, evidenziando una scalabilità continua anche per immagini di dimensioni più grandi.
Per quanto riguarda le configurazioni con più di 4 thread, lo speedup rimane costante, come si può osservare nella figura 5. Questo comportamento è dovuto al fatto che, nonostante l'incremento del numero di thread, il tempo di esecuzione parallela rimane pressoché invariato, arrivando a una saturazione. La mancata riduzione del tempo di esecuzione all'aumentare del numero di thread può essere attribuita all'eccessivo costo nella gestione dei thread stessi, portando a un overhead elevato che annulla i benefici derivanti dall'aumento del numero di thread.

L'**efficienza** la possiamo calcolare come il rapporto tra lo speedup e il numero di thread. Nelle nostre configurazioni abbiamo:

Dim Immagine	Eff (4 thread)	Eff (3 thread)	Eff (2 thread)
40×40	0.96	1.14	1.42
50×50	0.99	1.16	1.46
60×60	1.01	1.20	1.50
70×70	1.03	1.19	1.48
80×80	1.04	1.20	1.50

Table 4. Efficienza dell'algoritmo calcolata come rapporto tra lo speedup e il numero di thread.

I risultati sull'efficienza sono mostrati nella tabella 5.3 ed in figura 7 è possibile osservare anche l'efficienza per le configurazioni a più di 4 thread. Possiamo affermare che essi sono in linea con le aspettative, mostrando valori generalmente superiori a 1. L'efficienza è massima con 2 thread, grazie a uno sfruttamento ottimale delle risorse, ma diminuisce all'aumentare dei thread a causa della suddivisione del carico su un maggior numero di unità di esecuzione. Questo risultato è frutto di aver usato immagini ad bassa dimensionalità, quindi con un numero totale di pixels relativamente basso. Nonostante questa riduzione dell'efficienza, i tempi di esecuzione dell'algoritmo si abbassano in modo significativo all'aumentare del numero di thread, dimostrando che la parallelizzazione è efficace, anche su immagini di dimensioni limitate (1600-6400 pixel). L'efficienza per configurazioni a più di 4 thread cala drasticamente come era ovvio aspettarsi.

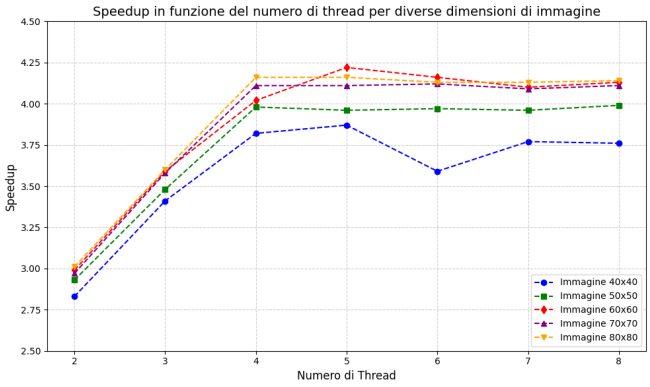

Figure 5. Grafico che mette in relazione lo speedup con il numero di thread per ogni dimensione dell'immagine

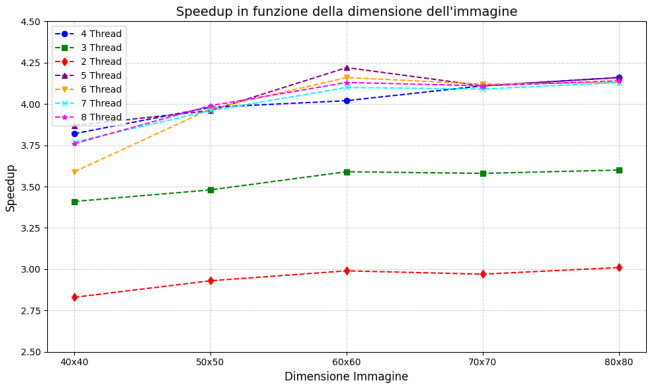

Figure 6. Grafico dello speedup in funzione della dimensione dell'immagine per ogni configurazione di thread usati

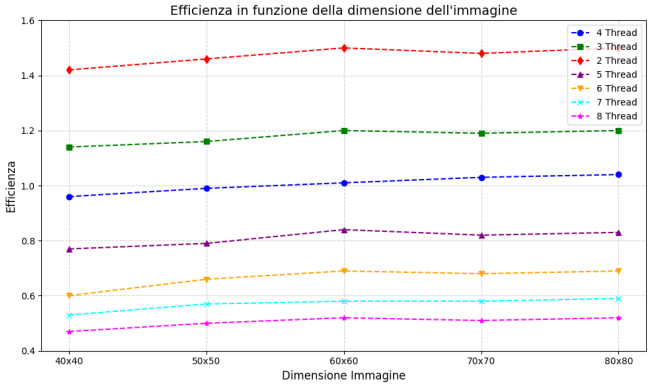

Figure 7. Grafico sull'efficienza in funzione della dimensione dell'immagine

6. Conclusioni

L'analisi condotta sull'algoritmo di Mean Shift Clustering, implementato sia in versione sequenziale che parallela, ha evidenziato i vantaggi derivanti dalla parallelizzazione. I risultati sperimentali hanno mostrato che, sfruttando OpenMP, l'algoritmo parallelo riduce significativamente i tempi di esecuzione rispetto alla versione sequenziale, con uno speedup che tende a crescere con l'aumentare della dimensionalità del dataset.

In particolare, l'algoritmo ha dimostrato un comportamento super-lineare per configurazioni con un minor numero di thread, beneficiando di un'ottimale distribuzione del carico di lavoro su immagini di piccole e medie dimensioni. Tuttavia, l'efficienza tende a diminuire con l'aumentare del numero di thread, a causa del sovraccarico introdotto dalla gestione parallela e del ridotto carico per thread nei dataset di dimensioni limitate. Questo comportamento suggerisce che l'algoritmo possa ottenere ulteriori miglioramenti di performance su dataset di grandi dimensioni, dove il parallelismo può essere sfruttato in modo più efficiente.

In conclusione, l'implementazione parallela dell'algoritmo di Mean Shift Clustering rappresenta una soluzione efficace per ridurre i tempi di esecuzione su immagini e dataset complessi, considerando anche $O(N^2)$ passi necessari per completare l'algoritmo.