SAPIENZA
UNIVERSITÀ DI ROMA

DIPARTIMENTO DI INFORMATICA

Implementazione in Parallelo dell'Analisi dei Componenti Principali

PROGETTO DI PROGRAMMAZIONE DI SISTEMI EMBEDDED E MULTICORE

Professore:
Salvatore Pontarelli

Studenti:
Ruben Ciranni
Gabriele Matiddi
Luigi Pizza

Anno Accademico 2023/2024

Indice

1 Introduzione

Lo scopo del progetto è l'implementazione di un algoritmo parallelo per l'Analisi delle Componenti Principali (PCA) [Bishop, 2006]. Questo algoritmo rientra nell'ambito della statistica multivariata ed è utilizzato per la riduzione della dimensionalità di dati, per la loro visualizzazione e per l'individuazione di correlazioni tra le variabili latenti dei dati stessi.
Nella nostra analisi, ci soffermeremo sull'utilizzo dell'algoritmo di PCA per la compressione di immagini, così da poter visualizzare chiaramente il risultato finale e confrontarlo con i dati immessi.
Per semplicità, lavoreremo su immagini in bianco e nero, ma il procedimento è estendibile, con poche modifiche, ad immagini a colori e ad altri tipi di dati provenienti da spazi ad alta dimensionalità.

2 Fondamenti di PCA

Tratteremo i dati di input come una matrice $M \in \mathbb{R}^{s\times d}$. Sia la matrice $P \in \mathbb{R}^{s\times d}$ ottenuta da M sottraendo la media delle sue righe da ciascuna riga. L'obiettivo della PCA, dato un intero t, è calcolare il sottospazio $\mathcal{U}$ generato dalle colonne di una matrice ortonormale $E^t \in \mathbb{R}^{d\times t}$ tale che la proiezione delle righe di P su $\mathcal{U}$ conservi il massimo della varianza, o equivalentemente, che minimizzi il costo della proiezione. Le colonne di E^t, che formano una base ortonormale di $\mathcal{U}$, sono dette componenti principali. In questo modo, è possibile ridurre il problema dell'avere la perdita minima di informazioni (al seguito della compressione di un immagine) al trovare una corretta base E^t.
Sia $S = \frac{1}{s}P^TP \in \mathbb{R}^{d\times d}$ la matrice delle covarianze delle righe di P. È possibile dimostrare che la suddetta base E^t è data data dai t autovettori di S corrispondenti ai primi t autovalori più grandi. Essendo S una matrice simmetrica, per il teorema spettrale, gli autovettori sono ortonormali.
L'algoritmo sequenziale di PCA è il seguente:

1. Si centrino le righe di M calcolando la media delle sue righe e sottraendola a ciascuna riga, ottenendo P.
2. Si calcoli la matrice delle covarianze $S = \frac{1}{s}P^TP$
3. Si calcoli la decomposizione spettrale di S. In particolare, si calcoli la matrice $L \in \mathbb{R}^{d\times d}$ degli autovalori e la matrice $E \in \mathbb{R}^{d\times d}$ degli autovettori di S tale che gli elementi sulla diagonale di L siano in ordine decrescente e $S = ELE^T$.
4. Sia $E^t \in \mathbb{R}^{d\times t}$ dato dalle prime t colonne di E.
5. Si proietti le righe di P sul sottospazio dato dalle colonne di E^t. In particolare, si calcoli la proiezione $Z \in \mathbb{R}^{d\times t}$ delle righe di P sul sottospazio $\mathcal{U}$, $Z = PE^t$.
6. Si riproietti le righe di Z sullo spazio originario di P, ottenendo la versione $\hat{P} \in \mathbb{R}^{d\times d}$ compressa di P, $\hat{P} = Z(E^t)^T$.

7. Riaggiungere la media delle righe di M calcolata all'inizio a ciascuna riga di $\hat{P}$, ottenendo $\hat{M}$, l'immagine compressa.

In alternativa, è possibile eseguire la PCA come segue (si può dimostrare l'equivalenza dei due metodi):

1. Si centrino le righe di M calcolando la media delle sue righe e sottraendola a ciascuna riga, ottenendo P.

2. Si esegua la Decomposizione ai Valori Singolari (SVD) di P: $P = UDE^T$ tale che i valori singolari sulla diagonale di D siano in ordine decrescente.

3. Sia $E^t \in \mathbb{R}^{d \times t}$ dato dalle prime t colonne di E.

4. Si proietti le righe di P sul sottospazio dato dalle colonne di E^t. In particolare, si calcoli la proiezione $Z \in \mathbb{R}^{d \times t}$ delle righe di P sul sottospazio $\mathcal{U}$, $Z = PE^t$.

5. Si riproietti le righe di Z sullo spazio originario di P, ottenendo la versione $\hat{P} \in \mathbb{R}^{d \times d}$ compressa di P, $\hat{P} = Z(E^t)^T$.

6. Riaggiungere la media delle righe di M calcolata all'inizio a ciascuna riga di $\hat{P}$, ottenendo $\hat{M}$, l'immagine compressa.

3 Un Algoritmo Distribuito per PCA

L'algoritmo 1 (basato su [Liang et al., 2013]) è una versione in parallelo dell'algoritmo della PCA, il quale può esser visto come una combinazione dei due metodi descritti nella sezione precedente.
Nell'algoritmo 1 ogni nodo centra la sua porzione dell'immagine (M_{local}) ed esegue due proiezioni: una PCA locale che proietta P_{local} in P^t_{local}, e una PCA globale che proietta P^t_{local} in $\hat{P}_{local}$. $\hat{P}_{local}$ viene poi decentrato con la media calcolata all'inizio ottenendo $\hat{M}_{local}$. Ciascuna $\hat{M}_{local}$ forma una porzione dell'immagine compressa.

Algorithm 1 Parallel PCA

$M \in \mathbb{R}^{s \times d} \leftarrow$ immagine globale
$t \leftarrow$ numero di componenti principali
$n \leftarrow$ numero di nodi
$s_{local} \leftarrow s/n$
Su ciascun nodo $v_{rank}, 1 \leq i \leq n$
  $M_{local} \in \mathbb{R}^{s_{local} \times d} \leftarrow$ frammento locale dell'immagine
  $m_{local} \in \mathbb{R}^d \leftarrow \vec{0}$
  for $r_j \in M_{local}, 1 \leq j \leq s_{local}$ **do**
    $m_{local} \leftarrow m_{local} + r_j/s$
  end for
  $m \leftarrow$ CollectiveSum(m_{local})
  $P_{local} \leftarrow M_{local} - m$
  $U_{local}, D_{local}, E^T_{local} \leftarrow$ SVD(P_{local})
  $D^t_{local} \leftarrow$ Primi t elementi di D sulla diagonale, altrimenti 0
  $P^t_{local} \leftarrow U_{local} \times D^t_{local} \times E^T_{local}$
  $S^t_{local} \leftarrow P^t_{local} \times (P^t{}_{local})^T$
  $S^t \leftarrow$ CollectiveSum(S^t_{local})
  if rank $= 0$ **then**
    $E, L \leftarrow$ EigenDecomposition (S^t)
    $E^t \leftarrow$ Prime t colonne di E
    Condividere E^t con gli altri nodi
  else:
    Aspettare E^t
  end if
  $\hat{P}_{local} \leftarrow P^t_{local} E^t \times (E^t)^T$
  $\hat{M}_{local} \leftarrow \hat{P}_{local} + m$
$\hat{M} \leftarrow$ Concat$(\hat{M}_{local})$
Output $\hat{M}$

4 Implementazione MPI

La prima implementazione è stata realizzata utilizzando la libreria `MPI` (*Message Passing Interface*).
Il programma riceve in input da linea di comando il numero di processi da avviare, il nome del file dell'immagine da comprimere, il numero di componenti principali t, e lo stile di normalizzazione dell'immagine.
Inizialmente, l'immagine viene caricata in memoria dal processo 0, e viene distribuita fra i processi tramite un `MPI_Scatter`. Vengono inoltre condivise le dimensioni delle matrici M_{local} (i.e. s_{local}, d) e il numero di componenti principali t tramite un `MPI_Bcast`. Ottenuti questi dati, ciascun processo calcola la media parziale delle righe della matrice fornitagli. Le medie parziali vengono sommate con un `MPI_Allreduce`. La media appena trovata viene sottratta ad ogni riga delle M_{local}, affinché ciascun processo ottenga il proprio frammento di immagine centrato. A questo punto, viene eseguita la SVD, usata per ottenere la matrice delle covarianze locale. Viene poi calcolata la matrice delle covarianze globale sommando tutte le matrici delle covarianze locali nel processo 0 con un `MPI_Reduce`.
Il processo 0 calcola i t autovettori corrispondenti ai primi t autovalori più grandi della matrice delle covarianze globale, attraverso la decomposizione spettrale della matrice delle covarianze. A questo punto, il processo 0 esegue un `MPI_Bcast` per diffondere tali autovettori agli altri processi, che utilizzeranno gli autovettori trovati per proiettare il proprio frammento di immagine sul sottospazio e riproiettare il risultato nello spazio originario, ottenendo il frammento locale dell'immagine compressa.
I frammenti vengono poi decentrati (utilizzando la media precedentemente calcolata) e normalizzati in base all'input, in modo che ciascun valore sia compreso fra 0 e 255 (questo step e necessario perché nella riproiezione alcuni valori potrebbero uscire dall'intervallo [0, 255]). L'immagine verrà poi ricostruita nel processo 0 tramite un `MPI_Gather` e mandata in output.

5 Implementazione PThreads

La seconda implementazione è realizzata utilizzando la libreria `PThreads` (*POSIX threads*).
Il programma riceve in input da linea di comando il numero di pthreads `thread_count` da avviare, il nome del file dell'immagine da comprimere, il numero di componenti principali t, e lo stile di normalizzazione dell'immagine.
Inizialmente, viene processato l'input, inizializzati i *mutex* ed allocata la memoria necessaria per i futuri calcoli. In base al `thread_count` ottenuto in input, l'immagine viene frammentata e vengono avviate le rispettive subroutines, passando come parametro un puntatore alla struttura `ThreadData` contenente i dati necessari ad eseguire la PCA locale per ogni thread.
Nelle subroutines ogni thread calcola la propria media parziale e la somma su uno spazio di memoria condivisa (utilizzando un *mutex*). Dopo aver superato una *barrier* (implementata con un *mutex*, una *condition variable* e un *counter*), ciascun thread

sottrae la media da ciascuna riga, centrando il proprio frammento dell'immagine.
A questo punto, ogni thread esegue la SVD sul proprio frammento dell'immagine e calcola la matrice delle covarianze locale.
Viene poi calcolata la matrice delle covarianze globale sommando su uno spazio di memoria condivisa le matrici delle covarianze locali (anche qui utilizzando *mutex* e *barrier*). Una volta che tutti i thread hanno completato la somma, il thread con rank 0 estrae i primi t autovettori dalla matrice della covarianze globale e li scrive ordinati su un ulteriore spazio di memoria condivisa. Ogni thread successivamente esegue la proiezione di ciascun frammento di immagine nel sottospazio $\mathcal{U}$, e i frammenti vengono riproiettati nello spazio originario. Il risultato viene poi normalizzato nell intervallo $[0, 256)$, ottenendo l'immagine compressa.
Nel thread principale viene utilizzato `pthread_join` per concludere l'esecuzione di ciascun thread. Infine viene mandata in output l'immagine compressa, la quale si trova nello stesso spazio di memoria condivisa dove era stata inizialmente caricata l'immagine originale.

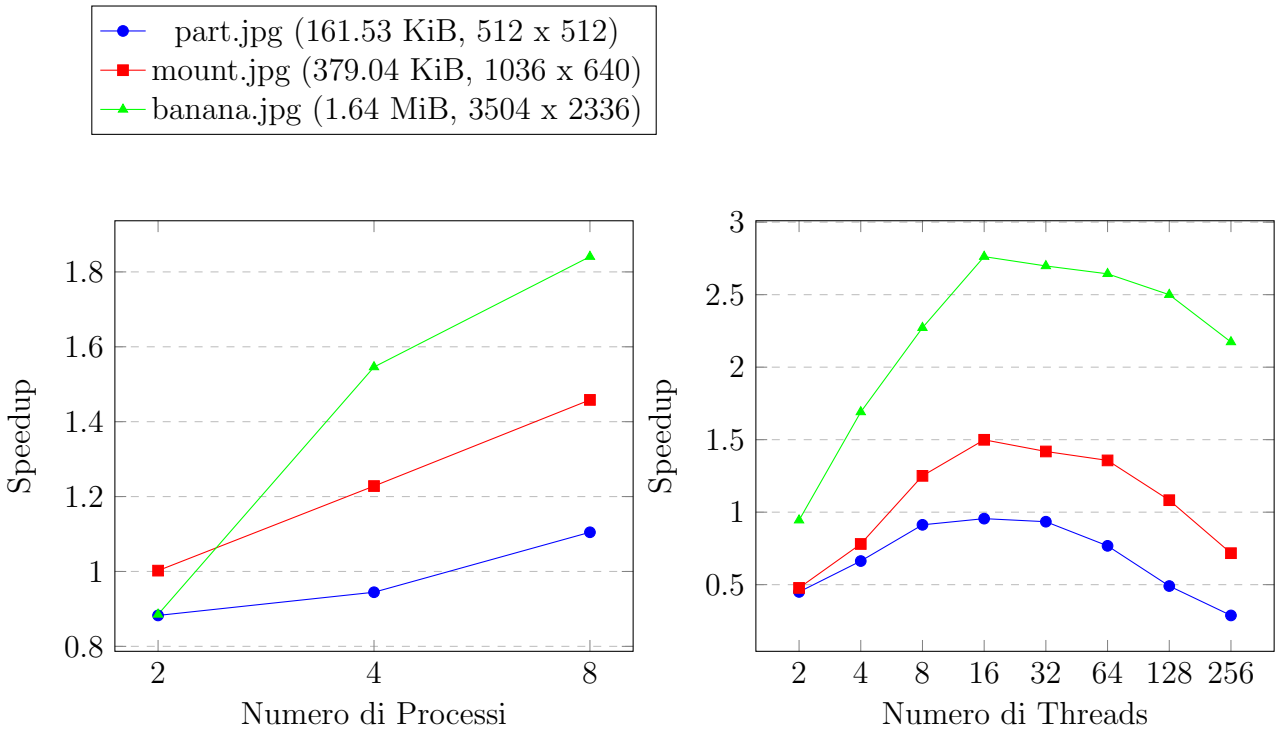
6 Risultati

Il programma ottiene l'effetto desiderato: i risultati ottenuti sono stati confrontati con implementazioni seriali già esistenti. Nella Figura 1 vi è un esempio di esecuzione, eseguendo la compressione dell'immagine part.jpg di 161.3 KiB a 26.0 KiB, riducendola al 23% della dimensione originale.
La Figura 2 illustra lo speedup dell'implementazione parallela in `MPI` e `PThreads` al variare dell'immagine, e del numero di processi/thread. I risultati sono stati ottenuti su un Lenovo ThinkBook 16p Gen 4 con un processore Intel Core i9-13900H di tredicesima generazione, eseguendo il programma su tre foto rispettivamente di dimensioni piccola, media e grande. Il numero di componenti principali t è stato impostato a 150.
Lo speedup maggiore si ottiene sull' immagine di grande dimensione, sulla quale in `MPI` con 8 processi lo speedup è stato di 1.84 e in `PThreads` con 16 thread di 2.6. Per l'immagine di medie dimensioni lo speedup è sempre superiore a 1 su `MPI` fino a un massimo di 1.46 con 8 processi, mentre su `PThreads` lo speedup è superiore a 1 solamente fra gli 8 e i 16 thread, con un picco di 1.5. Per quanto riguarda l'immagine di piccole dimensioni invece la versione seriale è generalmente migliore della versione parallela. Infine, ovviamente, in `PThreads` sopra i 16 thread lo speedup peggiora progressivamente fino a scendere sotto l'1.

(a) L'immagine prima dell'applicazione della PCA (161.3 KiB).

(b) L'immagine dopo l'applicazione della PCA (37.1 KiB).

Figura 1: Esempio di applicazione della PCA per la compressione di un immagine.



(a) MPI speedup

(b) Pthreads speedup

Figura 2: Speedup delle implementazioni MPI e PThreads

Riferimenti bibliografici

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Liang et al., 2013] Liang, Y., Balcan, M.-F., and Kanchanapally, V. (2013). Distributed pca and k-means clustering. In *The Big Learning Workshop at NIPS*, volume 2013.