Progetto FINALTERM

Implementazione Parallela di K-means tramite CUDA

Introduzione

Il progetto si propone di implementare una versione parallela dell'algoritmo K-means utilizzando la tecnologia CUDA. L'obiettivo è confrontare le prestazioni dell'algoritmo rispetto alla sua controparte sequenziale, sfruttando la potenza di calcolo parallelo delle GPU. L'ambiente di sviluppo principale è stato "Google Colab" che permette di sfruttare le risorse di accelerazione grafica messe a disposizione.

Descrizione dell'Algoritmo K-means

L'algoritmo K-means è un metodo di clustering che mira a suddividere un insieme di dati in gruppi omogenei, noti come cluster.

Implementazione Sequenziale

La funzione sequenziale dell'algoritmo K-means è stata implementata utilizzando la libreria scikit-learn in Python.

Implementazione Parallela con CUDA

L'implementazione parallela è stata realizzata utilizzando la libreria CUDA per sfruttare la potenza di calcolo delle GPU. Il codice CUDA è composto da due kernel principali: 'kmeans_kernel' e 'update_centers_kernel'.

Kernel di Assegnazione dei Cluster

(kmeans_kernel): calcola la distanza tra ciascun punto e tutti i centroidi. Assegna il punto al cluster più vicino.

Kernel di Aggiornamento dei Centroidi

(update_centers_kernel): calcola la media delle caratteristiche per i punti assegnati a ciascun cluster. Aggiorna il centroide di ciascun cluster con la media calcolata.

In entrambi i kernel, sono state adottate pratiche per evitare race conditions durante l'aggiornamento dei risultati.

Analisi del codice CUDA

Il codice CUDA di questo progetto sfrutta la potenza di calcolo parallelo delle GPU: viene scritto all'interno di una stringa denominata cuda_kernel e successivamente compilato e eseguito utilizzando la libreria PyCUDA. Nel codice CUDA, le funzioni kernel sono definite utilizzando il qualificatore __global__. Questo qualificatore indica che la funzione può essere chiamata da un host (la CPU) e può essere eseguita su un dispositivo CUDA (la GPU). Una funzione globale in CUDA è chiamata "kernel" ed è eseguita da ciascun thread parallelo. All'interno dei kernel, vengono utilizzati gli indici dei thread e dei blocchi CUDA per suddividere il calcolo tra i thread. Gli indici 'blockIdx.x' e 'threadIdx.x' vengono combinati per calcolare l'ID univoco di ciascun thread. Inoltre, vengono utilizzate istruzioni di controllo condizionale per garantire che solo i thread validi eseguano il calcolo. In seguito, per eseguire i kernel, il codice utilizza la libreria PyCUDA per compilare il codice CUDA e ottenere le funzioni kernel dal modulo CUDA compilato. I dati vengono trasferiti dalla CPU alla GPU utilizzando l'oggetto gpuarray.to_gpu di PyCUDA, questa funzione offre un'interfaccia più alta e semplificata per l'utilizzo della GPU rispetto a gestire manualmente le allocazioni di memoria e i trasferimenti dei dati tramite le funzioni CUDA di basso livello. Dopo l'esecuzione dei kernel, i risultati vengono trasferiti dalla GPU alla CPU utilizzando il metodo get() dell'oggetto gpuarray. Viene quindi verificata la convergenza confrontando i centroidi vecchi e nuovi. Se i centroidi convergono, l'algoritmo termina; altrimenti i nuovi centroidi vengono utilizzati per continuare l'iterazione (fino ad un massimo di iterazioni specificato dalla variabile 'max_iter'). Infine, è importante notare che il codice esegue la sincronizzazione CUDA utilizzando cuda.Context.synchronize() per garantire che tutte le operazioni CUDA siano state completate prima di procedere con le operazioni sulla CPU.

Risultati e Confronto

Questo progetto è stato eseguito su Google Colab, un ambiente di sviluppo basato su cloud, installando i pacchetti necessari mediante i comandi "!pip install chainer" e "!pip install pycuda". L'implementazione parallela di K-means mediante CUDA ha dimostrato un

notevole miglioramento delle prestazioni rispetto alla controparte sequenziale. L'analisi dei risultati evidenzia uno speedup considerevole, con il tempo di esecuzione del codice parallelo costantemente prossimo a 0.08 secondi (per 100.000 punti). Tuttavia è importante notare che il tempo di esecuzione della funzione sequenziale ottenuto con lo stesso numero di punti è un po' variabile, oscillando tra il secondo e i decimi di secondo.