Relazione Progetto CUDA

Silviu Robert Plesoiu
7051276
silviu.plesoiu@edu.unifi.it

Abstract

Il progetto qui descritto riguarda l'implementazione di una convoluzione 2D utilizzando CUDA per l'elaborazione di immagini. L'obiettivo principale è sfruttare la parallelizzazione offerta dalla GPU per migliorare le prestazioni rispetto a un'implementazione sequenziale su CPU.

1. Introduzione

Il progetto descritto riguarda l'implementazione di una convoluzione 2D utilizzando CUDA per l'elaborazione di immagini. L'obiettivo principale è sfruttare la parallelizzazione della GPU per migliorare le prestazioni rispetto a un'implementazione su CPU.

2. Algoritmo

L'algoritmo implementato esegue una convoluzione 2D su un'immagine utilizzando un kernel definito dall'utente. Il kernel viene applicato separatamente ai tre canali dell'immagine (rosso, verde e blu). La convoluzione viene eseguita su ciascun canale in parallelo sulla GPU.

2.1. Regole dell'algoritmo

- Per ogni pixel dell'immagine, il kernel viene applicato considerando un'area di dimensione 3×3 intorno al pixel.
- I bordi dell'immagine vengono gestiti tramite il clamping dei valori di indice.
- La memoria condivisa viene utilizzata per ridurre il numero di accessi alla memoria globale.

3. Implementazione Tecnica

3.1. Librerie Utilizzate

- **CUDA**: per il calcolo parallelo della convoluzione.
- **OpenCV**: per la gestione delle immagini, come caricamento, salvataggio e conversione.

3.2. Parallelizzazione con CUDA

Il codice CUDA utilizza una griglia di blocchi e thread per parallelizzare l'applicazione del kernel. Ogni thread gestisce un singolo pixel dell'immagine.

4. Benchmark e Risultati

Il test delle prestazioni è stato eseguito utilizzando un kernel fisso di dimensioni 3×3. Il focus è stato sul confronto tra una versione semplice e una versione ottimizzata con l'uso della memoria condivisa.

4.1. Risultati di Esecuzione

I tempi di esecuzione sono riportati nella seguente tabella:

Versione	Tempo di esecuzione (
Versione semplice	18.68 ms
Versione ottimizzata con memoria condivisa	0.098 ms

Table 1. Confronto tra versione semplice e ottimizzata.

4.2. Spiegazione delle prestazioni

Il miglioramento delle prestazioni è dovuto a due fattori principali:

- **Uso della memoria condivisa**: Riduzione degli accessi alla memoria globale.
- **Sincronizzazione dei thread**: Uso di `__syncthreads()` per garantire l'accesso corretto ai dati.