Diffusion-limited aggregation

Programmazione di SIstemi Embedded e Multicore

Simone Di Cesare

Febbraio 2024

1 Introduzione

Il Diffusion-limited aggregation (DLA) è un processo nel quale, partendo da uno spazio N-dimensionale popolato da K particelle, si formano cristalli. Il mio progetto si concentra sull'ottimizzazione di questo processo, utilizzando tecnologie come CUDA e OpenMP per ridurre i tempi di generazione dei cristalli.

2 Il Progetto

Il progetto è stato sviluppato sul sistema operativo Arch Linux, installato su un HP Omen del 2018, facendo uso del compilatore GCC per le versioni seriale e OpenMP e NVCC per la versione CUDA.

2.1 La Simulazione

La simulazione è stata implementata considerando le seguenti caratteristiche:

- Spazio bidimensionale.
- Assenza di collisioni con i bordi dello spazio.
- Generazione di un singolo cristallo iniziale.
- Uso di un seed statico per le particelle.

2.2 Funzionamento della Simulazione

Il funzionamento della simulazione è suddiviso in una serie di passaggi chiave che vengono eseguiti per ogni particella per un numero di iterazioni specificato.

1. **Lettura dei dati in ingresso**:
    - Vengono letti i seguenti parametri in ingresso:
        - Dimensioni della griglia (larghezza X altezza).

- Numero di particelle.
- Numero di iterazioni da svolgere.
- Posizione iniziale del primo cristallo (X, Y).
- (Facoltativo) File di output per la mappa (default: crystal.txt).

2. **Inizializzazione della griglia**:
   - La griglia viene inizializzata in un vettore ad una dimensione.
3. **Inizializzazione delle particelle**:
   - Viene assegnato un seed univoco a ciascuna particella (seed = index * 4).
   - Viene assegnata una posizione iniziale randomica a ciascuna particella, basata sul suo seed.
4. **Esecuzione della Simulazione**:
   - Per ogni particella e per ogni iterazione:
     (a) **Controllo dei vicini**:
       - Se la particella ha un vicino già cristallizzato, la particella stessa cristallizza e si ferma.
     (b) **Movimento della Particella**:
       - La particella si muove in una direzione casuale, determinata dal suo seed.
     (c) **Normalizzazione della Posizione**:
       - Se la particella esce dalla griglia, viene riportata all'interno seguendo delle regole di normalizzazione.

3 Performance

Come facilmente deducibile dal funzionamento della simulazione, una versione seriale dell'algoritmo sarà inefficente tanto quanto sono grandi il numero di particelle e/o di iterazioni della simulazione. Ad esempio, se eseguiamo la simulazione con questi parametri di ingresso:

1. Larghezza Griglia: 10000
2. Altezza Griglia: 10000
3. Numero di Particelle: 32768
4. Numero Iterazioni: 1000000
5. Posizione del primo Cristallo: (99, 99)

Osserviamo come il tempo di esecuzione totale è pari a 2566.17 secondi (quasi 43 minuti) suddivisi in:

- 40.10 millisecondi in inizializzazzione.
- 2557.53 secondi in simulazione.
- Restante in stampa e pulizia della memoria.

Per ovviare a queste alte tempistiche di simulazione, si propongono due strategie utilizzate per abbassare i tempi di simulazione.

3.1 OpenMP

OpenMP è un'interfaccia di programmazione che permette, tramite delle direttive pragma, di scrivere codice che può essere eseguito simultaneamente su più core della CPU.
Tramite questo meccanismo, ci è possibile suddividere le particelle tra più thread che verranno poi eseguiti simultaneamente, mantenendo invariato il funzionamento della simulazione ma aumentando l'efficenza. Nello specifico, la direttiva che permette una migliore performance è:
 # pragma omp parallel for schedule(static, 4)
Applicata sia per l'inizializzazione delle particelle, sia per il movimento di esse ad ogni iterazione, questa direttiva va ad eseguire in parallelo il for che va a muovere le particelle, assegnando staticamente le iterazioni del ciclo in chunck di 4. Quest'utlimo parametro **schedule(static, 4)** serve ad evitare il false sharing; Il numero 4 deriva dalla comune lunghezza delle linee di cache ad oggi, ossia 64 byte, diviso le dimensioni della struttura dati contenente le particelle, ossia 16 byte. Perciò, possono entrare al più 4 particelle (64/8) per linea di cache, e questo comporta ad assegnare chunk da 4 iterazioni per thread.
Eseguendo la simulazione con i parametri precedenti otteniamo un tempo totale di 919.59 secodni (15 minuti), suddiviso in:

- 57.71 millisecondi in inizializzazzione.
- 910.37 secondi in simulazione.
- Il restante in stampa e pulizia della memoria.

Notiamo come, a patto di un piccolo aumento nel tempo di inizializzazzione, andiamo drasticamente a ridurre i tempi di simulazione, ottenendo uno Speedup di 2.79, con un'efficenza di 0.23 (considerando i 12 core del computer).

3.2 CUDA

CUDA è una piattaforma di calcolo parallelo sviluppato da NVIDIA, con lo scopo di sfruttare la GPU per elaborare dati in parallelo.
Per ottimizzare la simulazione vengono lanciati tanti thread quante particelle, suddivisi in blocchi contenenti 256 thread, i quali si occupano di inizializzare e muovere le singole particelle.
Notiamo come, utilizzando gli stessi parametri della simulazione seriale, otteniamo un tempo totale d'esecuzione di 175.79 secondi (2.93 minuti), suddiviso in:

- 180.57 millisecondi in inizializzazzione.
- 163.45 secondi in simulazione.
- Il restante in stampa e pulizia della memoria.

Come per OpenMP abbiamo un aumento della durata di inizializzazzione, ma andiamo a ridurre di circa 15 volte il tempo d'esecuzione totale della simulazione.

4 Correttezza

Per mantenere invariato il risultato della simulazione per le varie versioni, viene utilizzato un seed statico per ogni particella. Il seed viene calcolato a partire dall'indice di quest'ultima, seguendo la formula **seed = index * 4**. Ogni qualvolta che bisogna estrarre un numero randomico, viene utilizzato il seed della particella e aggiornato con il nuovo valore generato. In questo modo, per ogni simulazione non ci sarà variazione alcuna nel movimento o nel posizionamento delle particelle a prescindere dalla versione del programma lanciato.