Diffusion Limited Aggregation Simulator

Gabriele Fabro
Francesco Fazzari

1 Introduzione

Per il nostro progetto di *"Programmazione di Sistemi Embedded e Multicore"* abbiamo deciso di sviluppare un software in linguaggio C che simula l'aggregazione di particelle ad un cristallo, questo fenomeno è comunemente chiamato **Diffusion Limited Aggregation**. In questo processo le particelle si muovono in una superficie 2D seguendo un moto Browniano, quando collidono con il seme o una sua diramazione si aggregano ad esso. Noi presentiamo più implementazioni, una single-thread e due multi-thread. Le due implementazioni multi-thread utilizzano le librerie **pthread** e **OpenMP**.

2 Come funziona

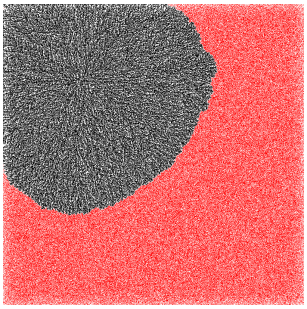
La nostra simulazione è scandita da istanti di tempo. In ogni istante di tempo vengono eseguiti in ordine i seguenti passi:

1. Viene piazzato in modo casuale sulla superficie il seme iniziale.
2. Viene generato un numero finito di particelle ognuna in posizione casuale. Le particelle possono sovrapporsi.
3. Nel primo istante se qualche particella è abbastanza vicina al seme si aggrega.
4. Nell' istante successivo tutte le particelle si muovono casualmente simulando un moto Browniano.
5. Se una particella dopo il movimento si trova abbastanza vicina a un seme da aggregarsi, dall'istante successivo inizierà a far parte del cristallo. Altrimenti continuerà il suo moto.

I punti 4 e 5 si ripeteranno finché tutte le particelle non saranno aggregate o fino al termine della simulazione. Il software svilupperà un'immagine finale dello stato del sistema al termine della simulazione ma è possibile avere anche un render video che mostra ogni cambiamento nel sistema ad ogni istante.

2.1 Più nel dettaglio

La nostra implementazione sfrutta una matrice per simulare la superficie di studio, dove ogni particella occupa una cella, e più particelle possono sovrapporsi. Le particelle sono rappresentate sulla matrice da un valore intero. Il valore 0 rappresenta una cella vuota, il valore 1 indica il seme oppure una sua diramazione, e qualsiasi valore maggiore o uguale a 2 e suo multiplo rappresenta il numero di particelle. Ad ogni istante di tempo per ogni particella vengono eseguite due funzioni che simulano i punti 4 e 5. Il punto 4 implementato in linguaggio C si presenta nel seguente modo.



```c
void move(particle *p, int n, int m)
{
    p->dire = rand() % 2 == 0 ? 1 : -1;
    p->current_position->x += rand() % 2 * p->dire;

    p->dire = rand() % 2 == 0 ? 1 : -1;
    p->current_position->y += rand() % 2 * p->dire;

    if (!(p->current_position->x >= 0 && p->current_position->x < m &&
    p->current_position->y >= 0 && p->current_position->y < n))
    {
        p->isOut = 1;
        return;
    }
    else
    {
        p->isOut = 0;
    }
}
```

Questo frammento di codice rappresenta il movimento
delle particelle, la nostra idea è stata quella di riprodurre il movimento casuale dandogli una direzione sia sull'asse delle X che sull'asse delle Y. I valori sono generai dalla funzione rand, della libreria random.

Nelle versioni multi-thread abbiamo adattato il codice utilizzando una reentrant random in quanto thread safe.
il punto 5 invece:

```c
int check_position(int n, int m, int **matrix,
                   particle *p, stuckedParticles *sp)
{
    if (p->isOut == 1)
    {
        return 0;
    }

    int directions[] = {0, 1, 0, -1, 1, 0, -1,
                        0, 1, 1, 1, -1, -1, 1, -1, -1};

    for (int i = 0; i < 8; i += 2)
    {
        int near_y = p->current_position->y + directions[i];
        int near_x = p->current_position->x + directions[i + 1];

        if (near_x >= 0 && near_x < n
            && near_y >= 0 && near_y < m)
        {
            if (matrix[near_y][near_x] == 1)
            {
                if(sp_append(sp, p) != 0){
                    perror("Error appending particle to \
                            stuckedParticles list. \n");
                }
                p->stuck = 1;
                return -1;
            }
        }
    }
    return 0;
}
```

Questo frammento di codice rappresenta il momento in cui la particella controlla se può unirsi al seme o ad una sua diramazione, oppure continuare a muoversi negli istanti successivi. Se la particella non è fuori dalla matrice, vengono controllate tutte le celle adiacenti. Se una cella contiene il seme o un cristallo, ed è ancora dentro la matrice, allora aggiorniamo la variabile stuck a '1' senza modificare la matrice. Al termine dell'istante di tempo, tutti i threads aggiornano la matrice con le nuove particelle aggregate. Per funzionare questa scelta progettuale necessita l'utilizzo di una barrier al termine di ogni tick prima di aggiornare la matrice, e un'altra barrrier prima del prossimo tick, per evitare che i threads invalidino la matrice durante l'esecuzione.

3 Efficienza

Dall' implementazione si deduce che il tempo di esecuzione è direttamente proporzionale al numero di particelle e all' orizzonte di simulazione. Durante il nostro studio abbiamo eseguito decine di tests su varie configurazioni e siamo arrivati alla conclusione che il tempo di esecuzione è in relazione al tempo di simulazione e alla saturazione della superficie di studio. Infatti all'aumentare della densità delle particelle alcuni test hanno avuto un' efficienza maggiore.

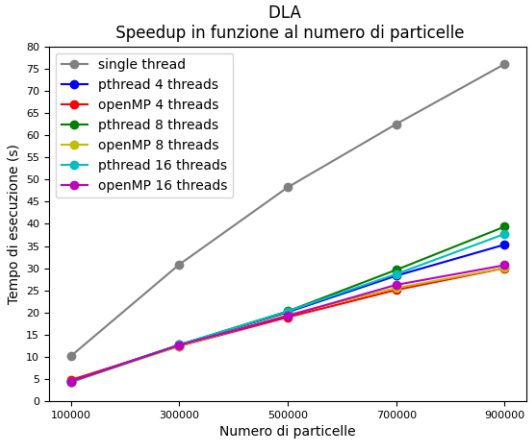


Figure 1: Media del tempo di esecuzione all'aumentare del numero di particelle in una matrice 1000x1000 con un tempo di simulazione pari a 1000 ticks

Dal grafico precedente è chiaro come le versioni multi-thread abbiano un notevole speedup. Di seguito vediamo in media un'aumento di prestazione del doppio rispetto al single-thread.

SPEEDUP		Numero di threads							
		pthread 2	openMP 2	pthread 4	openMP 4	pthread 8	openMP 8	pthread 16	openMP 16
Numero di Particelle	100.000	1,56	1,64	2,10	2,12	2,35	2,33	2,35	2,28
	300.000	1,65	1,69	2,42	2,48	2,43	2,48	2,42	2,44
	500.000	1,68	1,70	2,40	2,55	2,38	2,48	2,38	2,51
	700.000	1,92	1,95	2,21	2,49	2,11	2,45	2,18	2,38
	900.000	1,92	1,95	2,75	2,63	2,81	2,80	2,80	2,88

Figure 2: Tabella dello speedup in funzione al numero di particelle e al numero di threads.