Università della Calabria

Dipartimento di Matematica ed Informatica

Progetto per il corso di
Algoritmi Paralleli e Sistemi Distribuiti

Modello di Reiter
Simulazione della crescita di una stella di neve

Professore
William Spataro

Studente
Cannella Eliana
187180

Anno Accademico 2017 / 2018

Introduzione

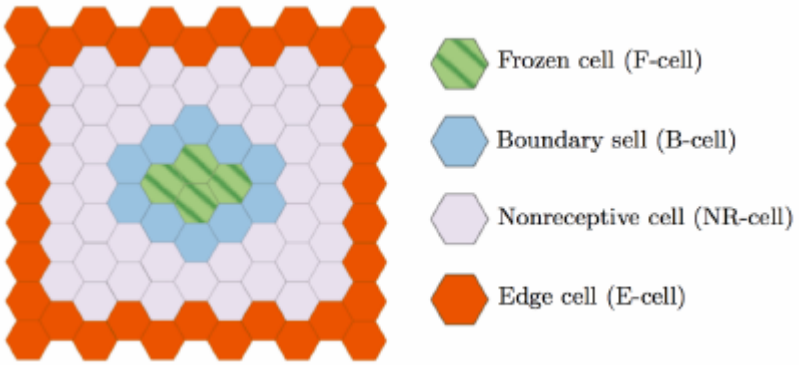
Il modello di Reiter è un automa che modella la crescita di una stella di neve. Utilizza alcune leggi fisiche per definire l'aggregazione dell'acqua e anche la sua distribuzione nel sistema.

Modello di Reiter

Il modello di Reiter è un automa esagonale. Data una tassellazione del piano in celle esagonali, ogni cella z ha sei vicini. Si denota s_t(z) lo stato variabile della cella z al tempo t che fornisce la quantità di acqua immagazzinata in z.

Le celle sono divise in 3 tipi:

- Una cella z è "frozen" se $s_t(z) \geq 1$.
- Se una cella non è "frozen" ma almeno un vicino è "frozen", la cella sarà una cella "boundary".
- Una cella che non è né "frozen" e nemmeno "boundary" è chiamata "nonreceptive".
- L'unione delle celle "frozen" e di quelle "boundary" sono chiamate celle "receptive"



La condizione iniziale nel modello di Reiter è:

$$s_0(z) = \begin{cases} 1 & \text{if } z = \mathcal{O} \\ \beta & \text{if } z \neq \mathcal{O} \end{cases}$$

In cui $\mathcal{O}$ è la cella di origine e β rappresenta una costante fissata del livello di vapore di background.

Inoltre si deifinisce

- $u_t(z)$, la quantità di acqua che partecipa nella diffusione
- $v_t(z)$, la quantità di acqua che non partecipa nella diffusione

Quindi:

$$s_t(z) = u_t(z) + v_t(z)$$

E si fissa $v_t(z) = s_t(z)$ se z è "receptive", e $u_t(z) = 0$ se z è "non-receptive".

Per γ, α due costanti fissate rappresentanti, rispettivamente, il vapore aggiunto e il coefficiente di diffusione, nel modello di Reiter lo stato della cella si evolve in funzione degli stati dei suoi vicini più prossimi secondo due regole di aggiornamento locale che riflettono i modelli matematici sottostanti:

- *Costante aggiuntiva*. Per ogni cella "receptive" z.

$$v_t^+(z) = v_t^-(z) + \gamma$$

- *Diffusione*. Per ogni cella z:

$$u_t^+(z) = u_t^-(z) + \frac{\alpha}{2}(\overline{u}_t^-(z) - u_t^-(z))$$

in cui si usa $\pm$ per denotare lo stato della cella prima e dopo che lo step sia completo, mentre $\overline{u}_t^-(z)$ definisce la media di u_t^- per i sei vicini più prossimi della cella z.

Le celle ai lati sono considerate come "edge cells", in cui il valore è $u_t^+(z) = \beta$. Così, l'acqua è aggiunta al sistema attraverso queste celle nel processo di diffusione.

Conbinando le due variabili, si ottiene:

$$s_{t+1}(z) = u_t^+(z) + v_t^+(z)$$

Descrizione Automa Cellulare

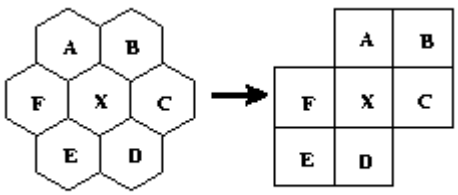
$$ReiterModel =< R, X, S, \sigma >$$

in cui:

- $R = \{(x, y) \mid x, y \in \mathbb{N}, 0 \leq x \leq l_x, 0 \leq y \leq l_y\}$ è una regione quadrata di celle quadrate, individuate da coordinate intere. Individua lo spazio in cui si diffonde la stella.
- $X =< (0,0), (-1,0), (-1,1), (0,1), (0,-1), (1,-1), (1,0) >$ ed è la vicinanza esagonale.
- $S = S_u \times S_v$ è l'insieme finito degli stati.
- $\sigma : S^7 \to S$ è la funzione di transizione.

Quindi in dettaglio, l'automa in questione ha una matrice esagonale rappresentata come matrice quadrata. Inoltre si hanno 3 matrici, una per considerare l'acqua rimasta nel sistema, una per definire l'acqua che partecipa alla diffusione e una per definire l'acqua che non partecipa alla diffusione.

Si utilizza la vicinanza esagonale, definita come segue:



La funzione di transizione applica le formule definite sopra ad ogni cella, le salva in alcune matrici i cui valori saranno poi assegnati alle matrici originali.

Inizialmente la matrice del sistema viene inizializzato a β tranne la cella centrale che avrà come valore 1. Da questo punto avrà inizio la diffusione e la crescita della stella.

Implementazione

Versione 1

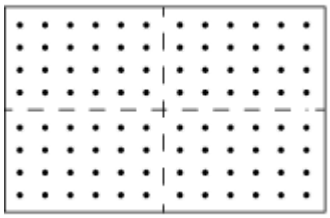
Topologia

Per rendere più strutturato il problema, si utilizza una topologia virtuale cartesiana in cui si mappano *4 processi* che saranno utilizzati per la computazione dell'automa cellulare. Infatti, grazie alla topologia, si creano le vicinanze.

Oltre alla topologia, ci sarà il processo *master*.

Decomposizione del problema

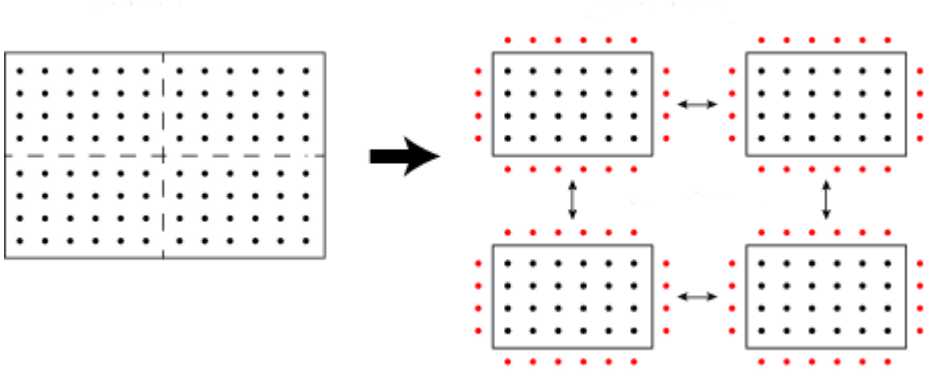
Per ottenere una buona parallelizzazione, è stato adottata una scomposizione del problema. Infatti, avendo una matrice, si divide in sottoblocchi ed ogni sottoblocco viene assegnato ad un processo.



Il processo master si occupa della divisione della matrice e dell'assegnazione agli altri processi, e della rappresentazione grafica. Questo processo manderà, a tutti gli altri processi, inizialmente le informazioni relative al blocco utilizzando *MPI_Struct* e poi il sottoblocco utilizzando *MPI_Send* e costruendolo con *MPI_Type_Vector*.

Ogni processo si occupa quindi di una sotto-matrice di dimensione $N_{row}/p \times N_{col}/p$.

Inoltre, ogni processo avrà bisogno di conoscere i suoi vicini in quanto l'update dell'automa, ovvero il send della sottomatrice al master, avverrà ogni tot di step, perciò nelle iterazioni intermedie ogni processo avrà bisogno di avere i bordi aggiornati per calcolare il valore esatto dell'ultima riga e l'ultima colonna del proprio sottoblocco.



Ogni processo comunicherà con i propri vicini attraverso la topologia per scambiarsi i bordi. Lo scambio dei vicini è stato effettuato con comunicazioni **non bloccanti**, per aumentare l'efficienza e diminuire la possibilità di deadlock.

Avendo usato comunicazioni non bloccanti, il processo eseguirà i seguenti step:

1. Esegue la funzione di transizione per tutto il suo sottoblocco escludendo l'ultima riga e l'ultima colonna
2. Riceve ed invia i bordi
3. Esegue la funzione di transizione sull'ultima colonna e sull'ultima riga

I bordi sono stati definiti secondo l'orientamento, *orizzontale* e *verticale*. I primi sono costruiti con un *MPI_Type_contiguos*, invece i secondi con un *MPI_Type_Vector*, per facilitarne l'invio.

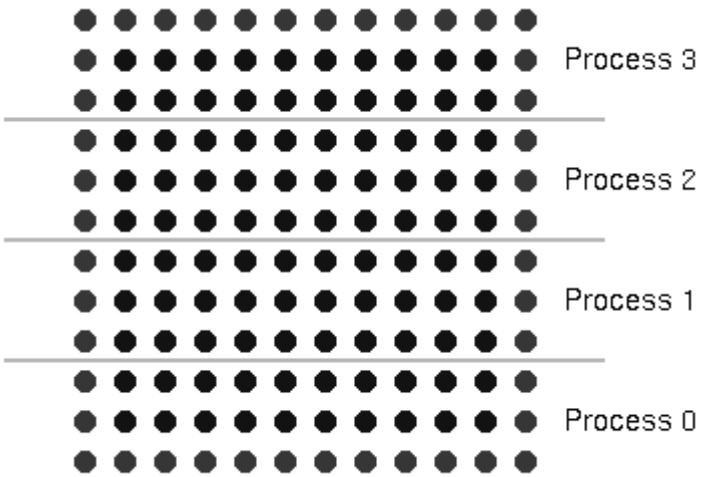
Ogni processo utilizza quindi due matrici linearizzate (per ogni matrice, stato del sistema, e due sottostati) di dimensione uguali: la prima rappresenta la matrice con i dati correnti e la seconda rappresenta, invece, i valori dello stato successivo calcolati con la funzione di transizone. Alla fine della funzione, la nuova matrice sarà assegnata alla matrice dello stato corrente.

Versione 2

Si è implementata anche una versione linearizzata senza utilizzare topologie e costrutti. Quindi, non avendo una topologia, in questo caso anche il master collabora nella computazione.

L'invio dei bordi è uguale a prima, solo che i vicini vengono mandati in maniera **bloccante** dal master a tutti gli altri processi, in modo che loro sapranno con chi comunicare.

Quello che cambia è la divisione della matrice, perchè i sottoblocchi vengono dati a righe, quindi ogni processo avrà *N_{col}/p* righe.



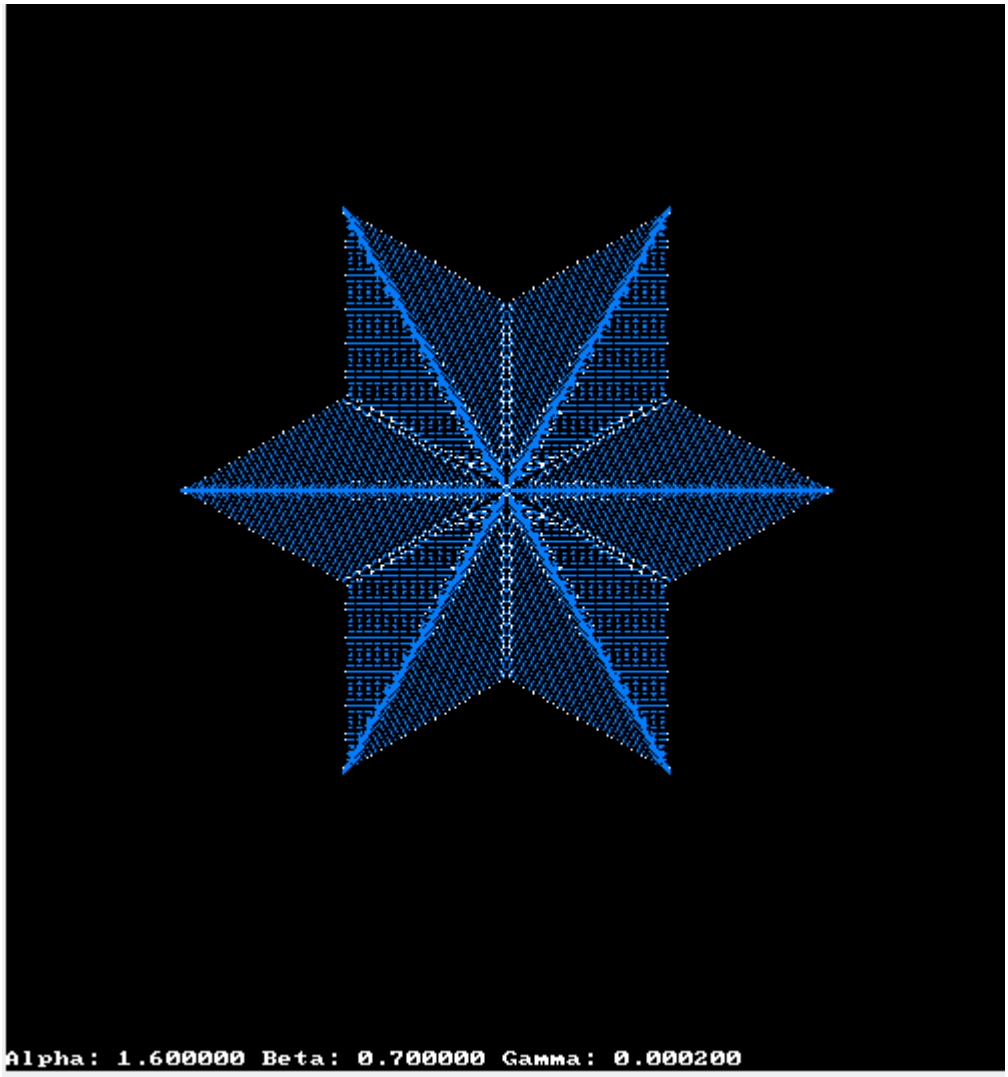
Inoltre non vengono utilizzati Datatype di MPI, perciò i send saranno con lunghezza pari all'intero blocco.

Rappresentazione Grafica

La rappresentazione grafica è stata implementata usando *Allegro 4.0*.

Il processo master si occuperà della rappresentazione grafica. Infatti ad ogni tot di step gli slave manderanno in maniera **non bloccante** al master le loro sottomatrici, ed il master si occuperà di rimettere insieme i pezzi e disegnare.

È da notare che al master verrà mandata soltanto la griglia del sistema e non anche le altre due matrici rappresentanti l'acqua che partecipa alla diffusione e quella che non partecipa, perchè il master dovrà soltanto disegnare il sistema.



Nella rappresentazione si è usato il blu per definire il solidificamento dell'acqua ed il bianco per lo stato in cui l'acqua nella cella è compresa tra 0.9 e 1.0 e che quindi si andrà a solidificare.

Risultati

I test sono stati effettuati su una macchina dotata di un processo *Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz* e *8 GB* di RAM. Le esecuzioni sono state lanciate in relazione alla versione dell'implementazione.

Le dimensioni per la matrice sono state:

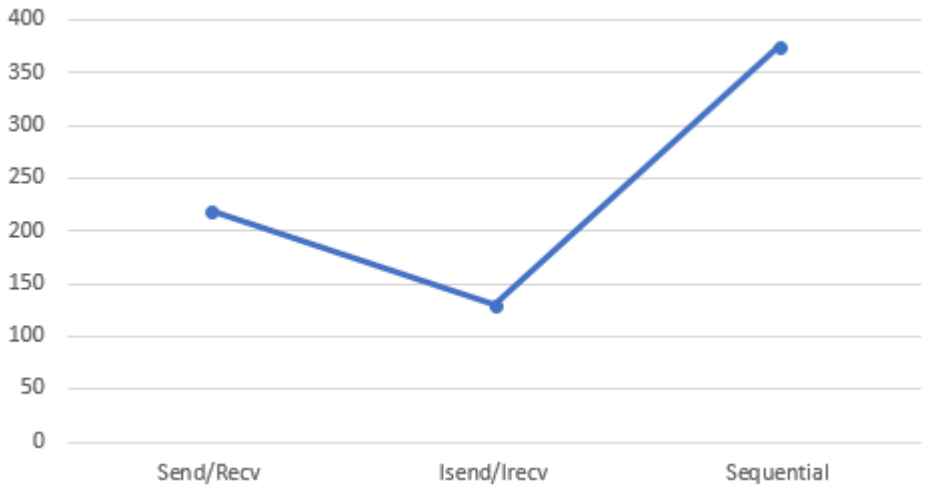
- **500x500**
- **1000x1000**
- **2000x2000**

E sono stati eseguiti su un totale di **1000** passi.

Versione 1

Le esecuzioni sono state lanciate con 4 threads. Inoltre si sono presi i tempi sia con l'implementazione in cui i bordi venivano mandati in modo *bloccante* e sia in modo *non bloccante*.

	500x500	1000x1000	2000x2000
Sequential	26.007761	117.310752	374.942568
Send/Recv	16.770004	55.620216	220.641125
Isend/Irecv	6.665900	43.605325	130.573338



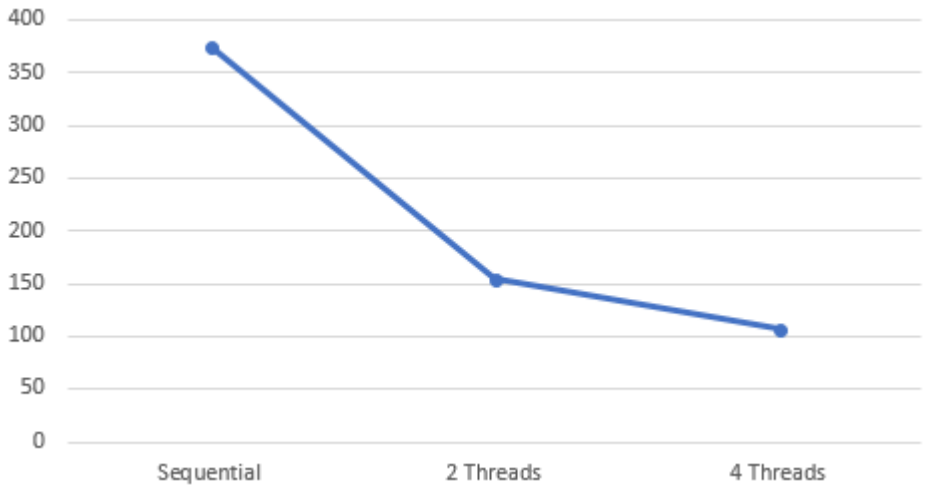
Per quanto riguarda lo speed-up:

	500x500	1000x1000	2000x2000
Sequential	1	1	1
Send/Recv	1.55085	2.109139	1.699332
Isend/Irecv	3.901613	2.690285	2.871509

Versione 2

Le esecuzioni sono state lanciate con 2 e 4 thread. Si sono presi i tempi solo con l'implementazione in cui i bordi venivano mandati in modo *bloccante*.

	500x500	1000x1000	2000x2000
Sequential	26.007761	117.310752	374.942568
2	10.507298	44.689545	154.830099
4	6.069994	40.693913	106.848232



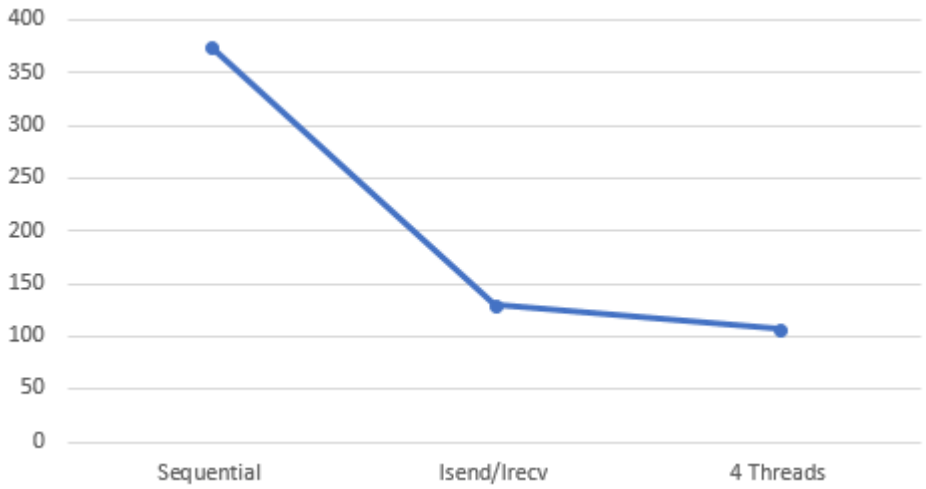
Per quanto riguarda lo speed-up:

	500x500	**1000x1000**	**2000x2000**
Sequential	1	1	374.942568
2	2.475209	2.625016	2.421639
4	4.284644	2.882759	3.509113

Sia nella prima versione che nella seconda versione, si nota un netto miglioramento delle versioni parallele rispetto a quella sequenziale.

Confronto

Mettendo in relazione i tempi della versione 1 e della versione 2:



Quello che è più notevole, è che confrontando le due versioni, il miglioramento maggiore è nella versione senza utilizzo di topologia e strutture derivate. Questo è dovuto al fatto che la topologia ha lo scopo di strutturare il codice ma si ha un peggioramento anche minimo.