**Relazione progetto Automa Cellulare**

*Cangeri Fabio 213542*

Il mio progetto consiste nell’implementazione in MPI di un automa cellulare che prevede l’evoluzione di un terreno, ispiratomi all’automa cellulare “Forest fire”.

Ho visto su internet l’evoluzione del modello Forest fire e mi è piaciuto tanto perché, a differenza di altri automi cellulari come il gioco della vita, le sue iterazioni non considerano soltanto le celle adiacenti, come generatrici di evoluzioni, ma anche dei parametri probabilistici, legati sia alla generazione di nuovi alberi che a nuovi incendi. Un aspetto che non ho apprezzato del modello “Forest fire” è la sua grafica, per questo ho deciso di caricare opportune immagini per rendere più comprensibile l’esperimento dell’automa cellulare.

Con l’automa Terreno, si vuole andare ad analizzare l’espansione di possibili infezioni che colpiscono le piante (ad esempio gli ulivi minacciati dal virus della “xylella”, anche se ho deciso di rimanere sul generale), similmente a quanto accade agli esseri umani oppure ad altri esseri viventi, tenendo come paramento fondamentale la distanza delle celle adiacenti alla posizione considerata nella matrice.

**Caratteristiche**

L’automa considera diversi parametri:

* Evoluzione statica del vegetale, da germoglio ad albero passando per pianta. Una volta, divenuto albero, esso permane in questa condizione;
* Probabilità che un vegetale secchi casualmente (percentuale crescente da germoglio ad albero, quindi un albero ha più probabilità di seccarsi rispetto a piante e germogli, così come le piante nei confronti dei germogli) oppure perché adiacente ad un certo numero di vegetali secchi;
* Probabilità che un vegetale si infetti casualmente (percentuale crescente da germoglio ad albero, quindi una pianta ha più probabilità di infettarsi rispetto ad un germoglio e l’albero ha più probabilità di infettarsi rispetto alla pianta ed al germoglio) oppure data dalla vicinanza di altre celle infette;
* Probabilità che dato una zolla di terreno vuota si generi un nuovo germoglio, perché ho dato la possibilità di piantare soltanto nuovi germogli, una volta che l’automa è stato lanciato;
* Probabilità di determinare la percentuale di piante presenti all’inizio;
* Possibilità di modificare tutte le percentuali citate sopra per osservare meglio e/o accentuare fenomeni;
* Possibilità di eliminare le infezioni dei vegetali, solo se è presente almeno un’infezione, rendendo il terreno non più biologico.

**Spiegazione**

Classi:

* Immagini: salvo le immagini che leggo da input, salvate nel resources in un vettore;
* Stato: creo delle variabili constanti che indicano lo stato in cui si trovano le piante;
* Pianta: indica lo stato della pianta, potevo anche fare una matrice di interi ma così è più leggibile;
* Percentuali: gestisce tutte le percentuali;
* Terreno: gestisce il core dell’automa.

Nella classe Terreno creo due matrici, in una salverò gli stati che si generano dalla prima matrice. Se una cella è vuota, si potrebbe generare un nuovo germoglio. Se un vegetale è infetto oppure secco diventerà vuoto. Se un vegetale è vivo potrebbe diventare infetto (questa probabilità ha la priorità rispetto alla condizione secco), secco oppure evolversi, mentre se è già un albero non potrà crescere di più.

**Soluzione MPI**

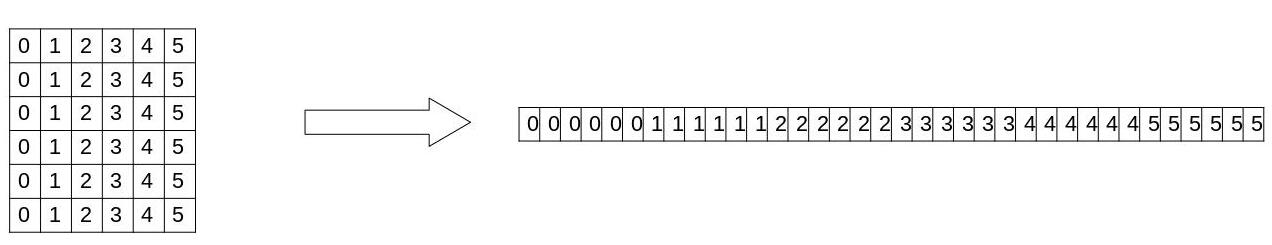
Dimensioni:

*matrice: SIZE \* SIZE*

*vettore: dimVettore*

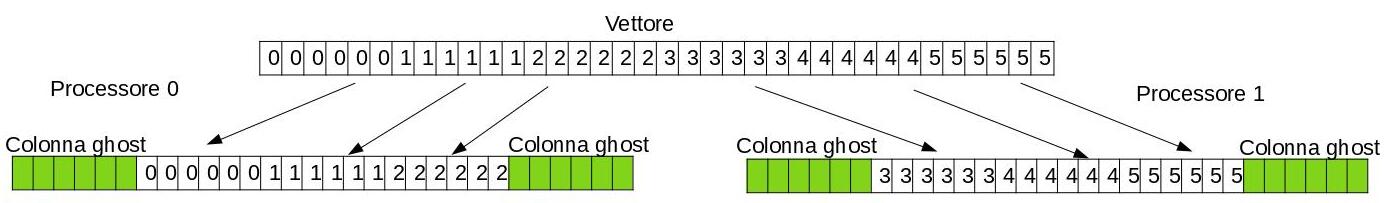
*vettoreLocale: dimVettoreLocale*

Ho inizializzato una matrice dinamica e l’ho linearizzata mettendola in un vettore, in modo da avere elementi di memoria continui. Così facendo le colonne sono scandite nel vettore una dopo l’altra, ovvero dalla posizione n alla posizione n + SIZE troviamo l’ennesima colonna della matrice.

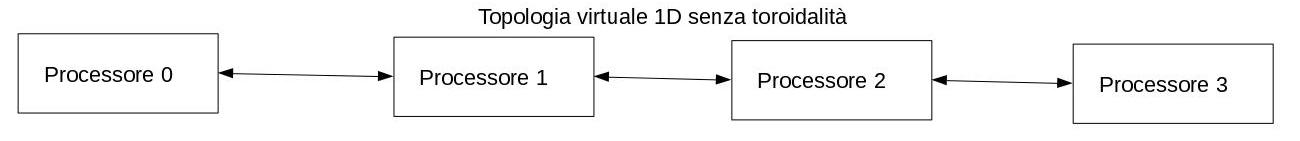


Questo passaggio, così come la generazione degli elementi nella matrice avviene soltanto nel processore 0.

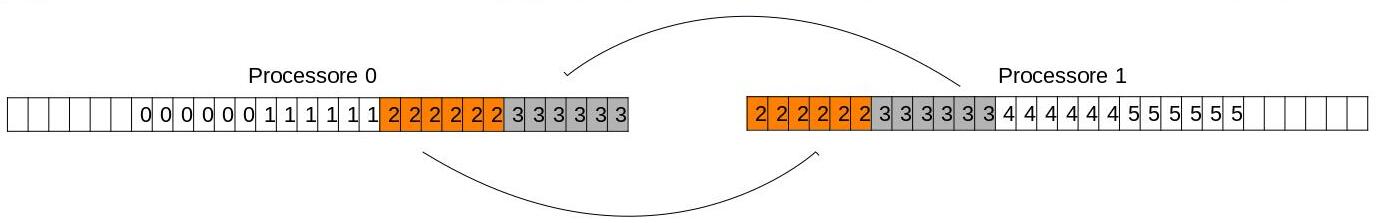
Dopo ho creato un datatype continuo, in quando del vettore, passo le colonne e quindi elementi continui. Il datatype creato lo utilizzo nello scatter, che distribuisce le colonne ai vari processori facendo attenzione che, definendo con dimVettoreLocale la dimensione del vettoreLocale, in ogni processore che le colonne ricevute, ovvero quelle su cui dovranno lavorare, vengano scritte dalla posizione SIZE in poi, in quanto la prima parte del vettoreLocale, ovvero quella che va dalla posizione 0 alla posizione SIZE, non riceva nulla in quanto rappresenta la colonna ghost, così come l’ultima parte del vettoreLocale che va dalla posizione dimVettoreLocale – SIZE a dimVettoreLocale, per lo stesso motivo.



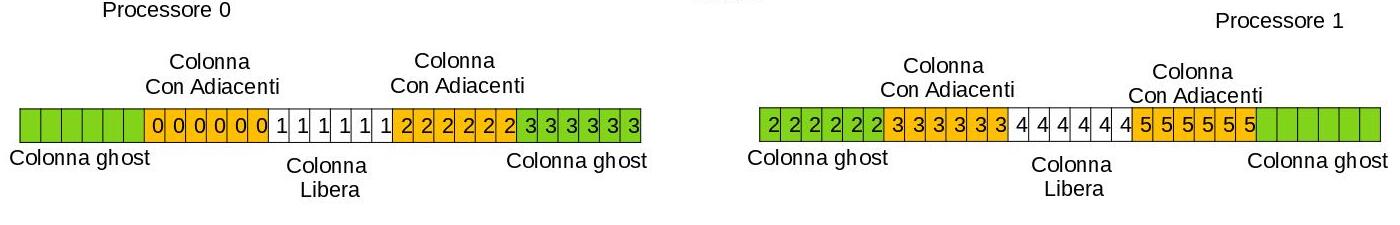
Successivamente definisco una topologia virtuale ad una dimensione. Mi serve per definire che i vicini di ogni processore i-esimo siano soltanto i+1 ed i-1, qualora essi esistano (ovvero i + 1 < processi e i - 1 >= 0).



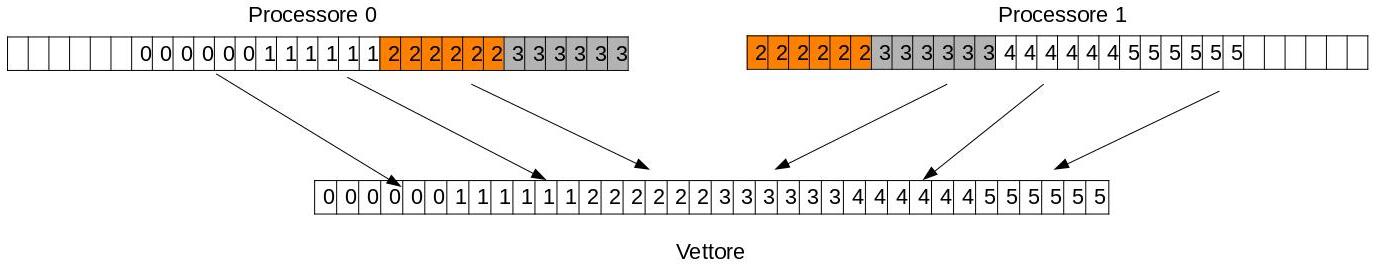
A questo punto, se il vicino del processore i-esimo esiste (ovvero il valore intero del suo vicino è un numero positivo), mi dispongo a ricevere ed inviare la colonna, che sarà colonna ghost nel processore medesimo. Più nel dettaglio, se il processore i-esimo ha un vicino sinistro, devo ricevere gli elementi che posizionerò dall’indice 0 all’indice SIZE, ed invierò al vicino sinistro gli elementi che lui riceverà come vicino destro; e/o se ha un vicino destro deve ricevere gli elementi che posizionerà dall’indice dimVettoreLocale – SIZE fino a dimVettoreLocale ed invierà gli elementi precedenti a tale posizione allo stesso vicino destro.



Successivamente, evolvo gli elementi, utilizzando una copia di vettoreLocale dove salverò gli elementi, e siccome sto effettuando una messaggistica non bloccante, quella ricevente, dovrei effettuare una barrier per essere certo di aver ricevuto quegli elementi. Prima però posso evolvere quelle colonne che non sono adiacenti alle celle ghost. Così facendo ottimizzo i tempi e dopo averle evolute, applico una barrier per poter infine aggiornare le celle che sono adiacenti a quelle ghost.



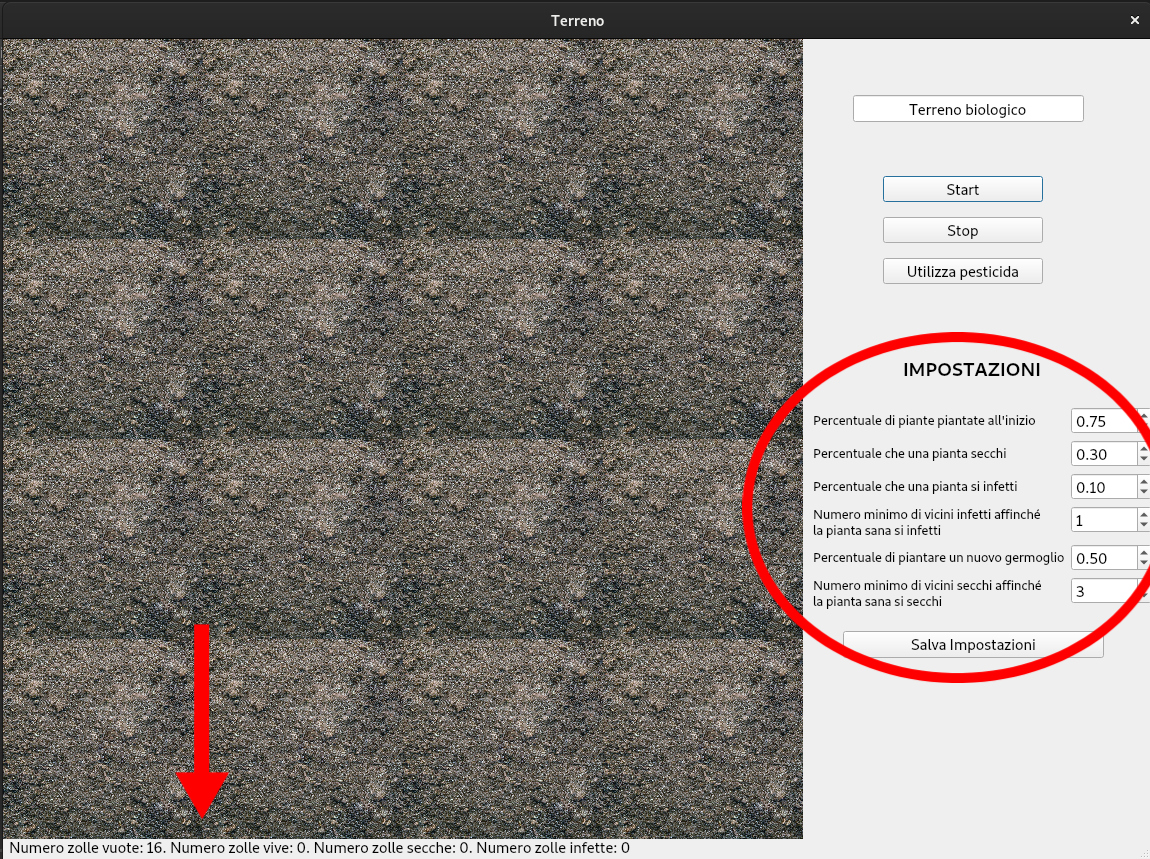
Infine, tramite una gather invio il vettoreLocale di ogni processore al processore 0, facendo attenzione a non inviare le celle ghost.



Infine stampo il vettore in modo tale che rappresenta la matrice di origine.

**Interfaccia**

La parte sequenziale si avvia con la grafica e tramite essa si può interagire arrestando ed avviando l’esecuzione del programma. Si possono anche modificare le impostazioni in esecuzione tramite gli elementi grafici (cerchiati in rosso). In alto a destra è indicato lo stato del terreno che può essere biologico oppure no, qualora si utilizzino pesticidi per guarire le piante infette ed in basso è indicato un conteggio delle zolle di terra vuote, vive, secche o infette (freccia rossa).

****

L’interfaccia grafica si avvia solo per la versione seriale, la versione parallelizzata stampa la matrice su file mostrando i tempi di esecuzione.

**Compilazione**

Per la versione sequenziale basta avviare qt ed aprire un progetto esistente caricando il file.pro. Si esegue direttamente poi all’interno di qt.

Per la versione parallela basta compilare da terminale, con un sottomultiplo di SIZE processori:

- mpicc terreno.c

- mpirun -np 2 ./a.out