Relazione progetto APSD: “Forest and Rain”

Studenti: Davide Ragona 200864, Salvatore Gatto 200856.

Descrizione dell’automa cellulare.

L’automa cellulare da noi scelto è basato sul noto progetto Forest Fire, un automa che simula il comportamento di una foresta i cui alberi possono incendiarsi in seguito ad un fulmine a cui abbiamo aggiunto dei piccoli accorgimenti. La foresta è composta di default da celle di terreno nelle quali possono crescere degli alberi. Ogni albero è soggetto alla probabilità di incendiarsi in seguito allo scoccare di un fulmine.

Se un albero prende fuoco e ha come vicino un albero, anche quest’ultimo prenderà fuoco, e così l’incendio potrà propagarsi nella foresta. Il fuoco inoltre è caratterizzato da tre intensità distinte; arrivato all’ultima intensità la cella di fuoco si spegne trasformandosi in una cella di terreno.

Ogni cella della foresta è soggetta anche alla possibilità di avere al suo interno una nuvola che simuli la pioggia che può spegnere il fuoco di tale cella, a prescindere da quale sia la sua intensità. Ogni nuvola si muove costantemente da sinistra verso destra.

Descrizione del codice

Il linguaggio di programmazione da noi utilizzato per lo svolgimento di questo progetto è il C++, con l’aggiunta della libreria “mpi.h”, che ci ha permesso di utilizzare le direttive per parallelizzare il codice seguendo il paradigma del Message-Passing attraverso l’uso di MPI.

In aggiunta a questi strumenti, abbiamo utilizzato la libreria grafica “Allegro 5” per poter dare un’interfaccia grafica all’automa cellulare da noi realizzato.

Abbiamo scritto delle funzioni per poter migliorare la leggibilità del codice che ci hanno permesso di rendere più chiara la funzione di transizione dell’automa cellulare iscritta nel main.

Per poter gestire i vari stati della foresta abbiamo utilizzato un enumerativo chiamato “Stato” il cui contenuto rappresenta i vari stati della foresta. Gli stati in questione sono:

0. Albero
1. Terra
2. Brucia1
3. Brucia2
4. Brucia3
5. AlberoPioggia → cella di tipo pioggia che ricorda lo stato precedente (ALBERO).
6. TerraPioggia → cella di tipo pioggia che ricorda lo stato precedente (TERRA).
7. Bruciapioggia → cella di tipo pioggia che ricorda lo stato precedente (BRUCIA 1-2-3).

Il terreno della foresta è stato creato utilizzando una matrice di interi, associando ad ogni posizione della matrice un enumerativo che indica uno dei vari stati che l’automa cellulare possa assumere.

Per poter simulare il comportamento della nuvola che si muova “al di sopra” di una cella, abbiamo creato tre distinti stati che simulino il comportamento della cella in seguito ad una nuvola di pioggia.

Insieme alla dichiarazione dell’enumerativo “Stato”, abbiamo dichiaro delle costanti rappresenti la soglia minima entro la quale un numero randomico possa generarsi, da 0 a 1, per poter gestire la crescita casuale degli alberi, lo scoccare di un fulmine o lo spuntare di una nuvola di pioggia.

La matrice primaria sulla quale viene costruita la foresta è una matrice linearizzata di interi. Per creare questa matrice abbiamo utilizzato il comando malloc, settando tale struttura dati su un vettore unidimensionale. Attraverso questa tecnica, la matrice presenta una disposizione contigua degli indirizzi di memoria. Abbiamo utilizzato questa tecnica per creare anche le sottomatrici locali di ogni processo. Inoltre, poiché la matrice si trova su un vettore unidimensionale, per poter accedere alle relative celle di memoria abbiamo sfruttato la seguente formula: i*dimColonne+j, con i e j che sono le variabili di loop dei due for con i quali scandiamo la struttura dati.

Fatto ciò, inizializziamo il vettore unidimensionale in base ad un numero randomico che viene generato per ogni iterazione che viene effettuata sulla matrice. Se tale numero rispetti una di quelle probabilità, in quella determinata posizione verrà settato lo stato corrispondente a quella probabilità.

In seguito all’inizializzazione della matrice, inizia la parte del codice in MPI. Questa parte è caratterizzata dalla creazione di due sottomatrici locali, la prima rappresenta l’area corrente della foresta che un processo possiede, la seconda invece rappresenta l’area successiva nella quale salveremo le modifiche che decideremo di fare durante la funzione di transizione.

Per poter gestire il movimento delle nuvole fra i vari processi abbiamo utilizzato due vettori Ghost che funzionano sia in scrittura che in lettura, salvando in essi il successivo passo che la nuvola dovrà fare e che dovrà essere ripreso dal processo successivo.

Prima di poter iniziare a gestire i passi dell’automa cellulare, viene effettuato un MPI_Scatter della matrice trasferendo ciascuna porzione della foresta nelle varie sottomatrici di ogni processo presenti all’interno di MPI_COMM_WORLD. Inoltre, abbiamo creato due variabili cellaSu e cellaGiu che rappresentano il processo precedente e successivo al processo corrente.

Queste variabili vengono utilizzate affinché all’inizio di ogni passo la matrice corrente possa ricevere nella prima e nell’ultima riga i vettori ghost del precedente processo.

Dopo aver fatto ciò, prende luogo la funzione di transizione del progetto che verrà effettuata per n-passi da ogni processo. In questa parte del codice inizializziamo la sottomatrice di supporto, copiando in essa il contenuto della sottomatrice corrente in modo che le modifiche che dovremo apportare verranno effettuate in questa struttura dati di supporto.

I vettori ghost ricevuti dal Send and Receive iniziali vengono inoltre gestiti in questa parte del codice, controllando per entrambi lo stato attuale e aggiornandoli di conseguenza.

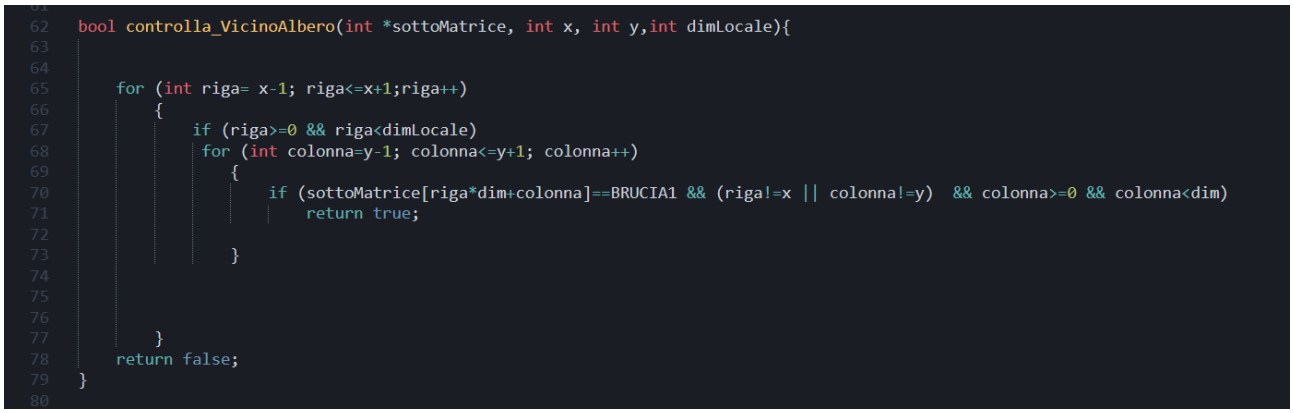
In seguito, il codice prevede la generazione di un numero casuale da 0 a 1, di tipo double, che se inferiore alle probabilità dichiarate all’inizio del programma, ci permette di generare casualmente una cella di pioggia, albero, o di fuoco. Le probabilità dichiarate sono tra di loro diverse in quanto la dinamicità dell’automa cellulare dipende anche da questa caratteristica. La possibilità che un albero possa crescere è maggiore rispetto alla probabilità che possa spuntare una nuvola di

pioggia e quest'ultima è anche minore rispetto alla possibilità che un albero possa bruciarsi. Per tale motivo, l'automa tenderà a far crescere prima degli alberi e solo dopo ciò, un incendio potrà scoppiare nella foresta in quanto il fulmine colpisce solo le celle di tipo ALBERO. Le nuvole all'interno della foresta potranno spuntare dovunque in maniera randomica e spariranno solo se in tali celle casualmente cresca un albero.

In seguito, attraverso una funzione "controllaVicinoAlbero", viene effettuata la gestione dell'incendio, verificando se nella cella corrente ci sia un Albero e in una cella vicino ce ne sia una di tipo "Brucia1", equivalente alla fase iniziale di un incendio.

Se tale condizione viene soddisfatta, questa cella prenderà cosi effettivamente fuoco. Questo controllo viene effettuato per ogni cella.

La funzione "controllaVicinoAlbero" è di tipo booleano in quanto verifica solo la condizione, la modifica effettiva viene effettuata all'interno del main, nella matrice di supporto.



```
bool controlla_VicinoAlbero(int *sottoMatrice, int x, int y,int dimLocale){


    for (int riga= x-1; riga<=x+1;riga++)
        {
            if (riga>=0 && riga<dimLocale)
            for (int colonna=y-1; colonna<=y+1; colonna++)
               {
                   if (sottoMatrice[riga*dim+colonna]==BRUCIA1 && (riga!=x || colonna!=y)  && colonna>=0 && colonna<dim)
                       return true;

               }



        }
    return false;
}
```

Oltre a dover gestire l'incendio, la funzione di transizione verifica se nella cella corrente che stiamo analizzando sia presenta una nuvola di pioggia per poter gestire il movimento di quest'ultima. Tale movimento viene gestito verificando quale stato ci sia nel prossimo passo della nuvola e modificando la cella successiva in base al tipo di stato presente in esso, apportando tale modifica nella matrice di supporto. Fatto ciò, la cella corrente torna al suo stato originale deducibile dal tipo enumerativo che possiede. Se la nuvola inoltre si trova sopra una cella di tipo BRUCIA, spegnerà l'incendio trasformando la cella in TERRA.

Se la nuvola si trova in una posizione pari a dimLocale-1, questa non si sposterà più nella sottomatrice del processo corrente ma verrà salvata nel vettore ghost inferiore.

```
if (i*dim<(dim*(dimLocale-1))){
    if(sottoMatrice[i*dim+j]==BRUCIAPIOGGIA || sottoMatrice[i*dim+j]==TERRAPIOGGIA || sottoMatrice[i*dim+j]==ALBEROPIOGGIA){
        if (sottoMatrice[(i*dim)+dim+j]==TERRA)
            supportoSub[(i*dim)+dim+j]=TERRAPIOGGIA;
        else if (sottoMatrice[(i*dim)+dim+j]==ALBERO)
                supportoSub[(i*dim)+dim+j]=ALBEROPIOGGIA;
        else if (sottoMatrice[(i*dim)+dim+j]==BRUCIA1 || sottoMatrice[(i*dim)+dim+j]==BRUCIA2 || sottoMatrice[(i*dim)+dim+j]==BRUCIA3 )
                supportoSub[(i*dim)+dim+j]=BRUCIAPIOGGIA;


        if(sottoMatrice[i*dim+j]==BRUCIAPIOGGIA)
            supportoSub[i*dim+j]=TERRA;
        else if (sottoMatrice[i*dim+j]==ALBEROPIOGGIA)
            supportoSub[i*dim+j]=ALBERO;
        else if (sottoMatrice[i*dim+j]==TERRAPIOGGIA)
            supportoSub[i*dim+j]=TERRA;

    }

}
else if ((i*dim)>=(dim*(dimLocale-1))){

        if (supportoSub[i*dim+j]==BRUCIAPIOGGIA || supportoSub[i*dim+j]==TERRAPIOGGIA || supportoSub[i*dim+j]==ALBEROPIOGGIA){
            if (vettoriGiu[j]==TERRA)
                vettoriGiu[j]=TERRAPIOGGIA;
            else if (vettoriGiu[j]==ALBERO)
                vettoriGiu[j]=ALBEROPIOGGIA;
            else if (vettoriGiu[j]==BRUCIA1 || vettoriGiu[j]==BRUCIA2 ||vettoriGiu[j]==BRUCIA3 )
                vettoriGiu[j]=BRUCIAPIOGGIA;

            if(supportoSub[i*dim+j]==BRUCIAPIOGGIA)
                supportoSub[i*dim+j]=TERRA;
            else if (supportoSub[i*dim+j]==ALBEROPIOGGIA)
                supportoSub[i*dim+j]=ALBERO;
            else if (supportoSub[i*dim+j]==TERRAPIOGGIA)
                supportoSub[i*dim+j]=TERRA;
        }
}
```

Finita la funzione di transizione vengono effettuati due send e receive finali per trasferire i vettori ghost nelle sottomatrici degli altri processi. In seguito, aggiorniamo la sottomatrice corrente eguagliandola alla sottomatrice di supporto. Fatto ciò, effettuiamo il gather per ricombinare assieme tutte le sottomatrici di tutti i processi in un'unica matrice che possiamo visualizzare con la funzione "disegnaConAllegro".

Tale funzione utilizza la libreria Allegro per poter disegnare per ogni cella della matrice un quadrato il cui colore varia in base al tipo di enumerativo corrispondente.

I colori associati ad ogni stato sono i seguenti:

- Terra → marrone;
- Albero → verde;
- Brucia1 → Rosso;
- Brucia2 → Rosso tenue;
- Brucia3 → Arancione
- Bruciapioggia, Terrapioggia e Alberopioggia → Celeste chiaro.

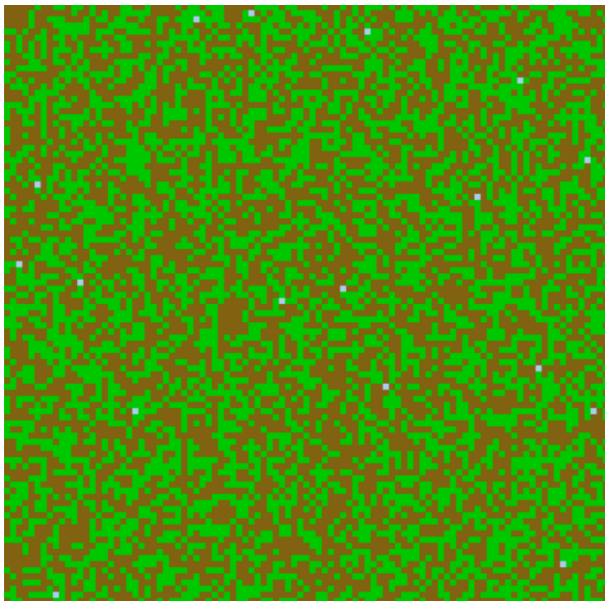
```
void disegnaConAllegro(int* world) {
    al_clear_to_color(al_map_rgb(0, 0, 0));
    for (unsigned i = 0; i < dim; i++)
    {
        for (unsigned j = 0; j < dim; j++)
        {
            if( world[i*dim+j]==TERRAPIOGGIA || world[i*dim+j]==ALBEROPIOGGIA ||world[i*dim+j]==BRUCIAPIOGGIA)
            {
                al_draw_filled_rectangle(i * latoQuadrato, j * latoQuadrato, i * latoQuadrato + latoQuadrato,
                    j * latoQuadrato + latoQuadrato, al_map_rgb(171, 205, 239)); // celeste chiaro
            }
            else{
                switch (world[i*dim+j]) {
                case 0: // albero
                    al_draw_filled_rectangle(i * latoQuadrato, j * latoQuadrato,
                            i * latoQuadrato + latoQuadrato,
                            j * latoQuadrato + latoQuadrato, al_map_rgb(0,200,0)); // verde
                    break;
                case 1:  // terra
                    al_draw_filled_rectangle(i * latoQuadrato, j * latoQuadrato,
                            i * latoQuadrato + latoQuadrato,
                            j * latoQuadrato + latoQuadrato, al_map_rgb(128, 98, 16)); // marroncino
                    break;

                case 2: // brucia1
                    al_draw_filled_rectangle(i * latoQuadrato, j * latoQuadrato,
                            i * latoQuadrato + latoQuadrato,
                            j * latoQuadrato + latoQuadrato, al_map_rgb(225, 0, 0)); // rosso
                    break;
                case 3: // brucia2
                    al_draw_filled_rectangle(i * latoQuadrato, j * latoQuadrato,
                            i * latoQuadrato + latoQuadrato,
                            j * latoQuadrato + latoQuadrato, al_map_rgb(255, 50,30)); // rosso chiaro
                    break;
                case 4: // brucia3
                    al_draw_filled_rectangle(i * latoQuadrato, j * latoQuadrato,
                            i * latoQuadrato + latoQuadrato,
                            j * latoQuadrato + latoQuadrato, al_map_rgb(255, 130, 100)); // arancione chiaro
                  break;

                default:
                    break;
                }
            }
        }
```

Infine, il programma termina con il comando MPI_Finalize().

Snapshot del progetto



Screenshot dell’automa cellulare senza incendio

Screenshot dell’automa cellulare eseguito con 2 processi incendio e nuvole che passano tra questi

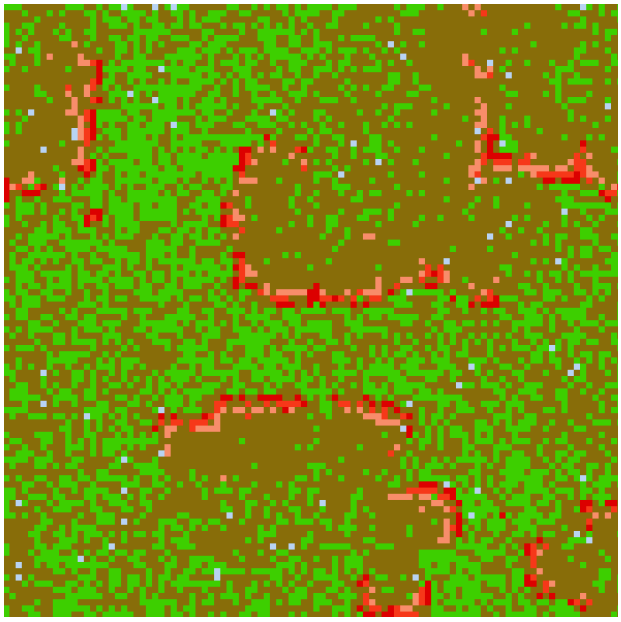


Tabella dello Speed-Up

Timing con 100 passi

Threads -Dim	100	400	800	1000
1	0.13	1.32	5.8	8.12
2	0.09	0.95	3.54	5.38
4	0.06	0.73	2.68	3.72

Timing con 1000 passi

Threads -Dim	100	400	800	1000
1	1.29	15.02	63.01	95.2601s
2	0.77	10.27	37.41	39.0212s
4	0.64	7.25	30.52	34.2842 s

Speed Up (con 100 Passi)

Threads -Dim	100	400	800	1000
2	1.44	1.38	1.63	1.50
4	2.16	1.80	2.16	2.18

Speed Up (con 1000 Passi)

Threads -Dim	100	400	800	1000
2	1.67	1.26	1.39	1.67
4	2.01	1.79	1.70	1.90