

MPI - Forest-fire simulation

Il **forest-fire model** è stato creato utilizzando MPI durante il corso di *PROGRAMMAZIONE CONCORRENTE PARALLELA E SU CLOUD* all'università degli studi di Salerno

Indice

- [Introduzione](#)
- [Breve descrizione della soluzione](#)
- [Istruzioni per l'esecuzione](#)
- [Dettagli di implementazione](#)
- [Prova della correttezza](#)
- [Benchmarks](#)


Introduzione

Il modello è definito come un automa cellulare su una griglia NxM. Una cella può essere vuota, occupata da un albero, da un seme di albero, o in fiamme. Il modello è definito dalle seguenti sei regole che vengono eseguite contemporaneamente:

1. Una cella in fiamme si trasforma in una cella vuota;
2. Un albero brucerà se almeno un vicino sta bruciando;
3. Un seme di albero si trasforma in un albero;
4. Un albero si accende con probabilità $f=1\%$ anche se nessun vicino sta bruciando;
5. Uno spazio vuoto si accende con probabilità f ;
6. Uno spazio vuoto si riempie di un seme d'albero con probabilità $p=6\%$;

Lo stato di una cella può essere:

Emoji	ASCII	Stato
	T	Albero
	i	Seme albero

Emoji	ASCII	Stato
	-	Vuoto
	*	Fuoco

La simulazione è eseguita un numero fissato di volte. Ad ogni fase di simulazione, i processori eseguono le 5 regole precedentemente descritte, e comunicano con i processori vicini per completare il calcolo.

Breve descrizione della soluzione

Il codice prende in input dall'utente:

1. Se eseguire le stampe;
2. la grandezza della matrice;

Tutti i **processori** generano una porzione di matrice in base alla taglia che gli è stata assegnata, successivamente continuano la propria simulazione indipendentemente, fino a quando, se necessario completeranno il calcolo chiedendo ai worker vicini i dati necessari altrimenti no. Una volta terminata la simulazione sui singoli **workers**, se è stato chiesto di stampare l'andamento della simulazione, il processore **con rank 0** stampa a video lo stato di essa.

Istruzioni per l'esecuzione

Pre-requisiti

- Ubuntu 18.04 LTS (*min: 10 GB storage*);
- Set-up dell'ambiente seguendo <https://github.com/spagnuolocarmine/ubuntu-openmpi-openmp> ;

Esecuzione locale

- Avere nella stella directory i file: `mycollection.h` e `forest-fire.c` ;
- Eseguire il comando `mpicc forest-fire.c -o forest` per la compilazione;

- Eseguire `mpirun --allow-run-as-root -np 4 forest` (4 è un numero di processori indicativo, si può assegnare il valore più consono alla propria macchina).

Esecuzione distribuita

- Avere n macchine che condividono lo stesso [set-up](#) stabilito nei pre-requisiti;
- Ogni macchina deve avere nella stessa directory `hfile`, `mycollection.h` e `forest-fire.c`;
- Eseguire il comando `mpicc forest-fire.c -o forest` per la compilazione;
- Eseguire sulla macchina **master** `mpirun --allow-run-as-root -np 2 --hostfile hfile ./forest`.

Run-time

Allo start-up del programma viene chiesto all'utente la modalità di stampa (Eseguire la `<print-mode>` influisce molto negativamente sulle performance dovuto a 2 cause: `printf()` e `MPI_Gatherv()`)

- 0 la soluzione viene mostrata con i caratteri ASCII;
- 1 la soluzione viene mostrata con le emoji;
- 2 la soluzione viene mostrata in una modalità *“debug”* mostrando visivamente dove comunicano i processori tra loro.

```

FOREST-FIRE SIMULATION
BY ANTONIO ZIZZARI
Wanna run the <print-mode> (type 0 'NO', or 1 'YES')?:
1
Choose the program style (avialbe: 0, 1, 2):
2

```

Successivamente viene chiesta la grandezza della matrice sul quale si andrà a lavorare.

```

Give the matrix size (NxM), give N:
8
Give the matrix size (NxM), give M:
8

```

Infine si otterrà l'esecuzione desiderata con le relative stampe.

Dettagli di implementazione

Il codice main `forest-fire.c` è il seguente:

[illegible]

```
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include "mpi.h"
#include "mycollective.h" //Libreria con tutte le funzioni essenziali
```

```

int main(int argc, char **argv) {
    /*Dichiarazione variabili iniziali*/
    int myrank,numtasks;
    int split,plus_i,N,M,style=0,print_mode;
    bool send=false;

    /*Probabilità della simulazione*/
    int prob_generate_tree=60;
    int prob_generate_fire=1;
    int prob_generate_seed=6;

    /*Variabili tempo per speed-up*/
    double start, end;

    /*Codice MPI essenziale*/
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

    //Nel caso ci interessasse la non esecuzione sequenziale
    /*
    if(numtasks==1){
        fprintf( stderr, "Numero processori non valido!!!\n");
        MPI_Abort(MPI_COMM_WORLD,1);
    }
    */

    //CODICE PER MOSTRARE CHE OLTRE I 16.000.000 ELEMENTI CRASHA IL CODICE
    //char test[16000001];
    //test[16000000]='1';

    /*Prendo in input le variabili essenziali*/
    if (myrank == 0)
    {
        print_menu();
        printf("Wanna run the <print-mode> (type 0 'NO', or 1 'YES')?:\n");
        scanf("%d",&print_mode);
        if(print_mode){
            printf("Choose the program style (available: 0, 1, 2):\n");
            scanf("%d",&style);
        }
        printf("Give the matrix size (NxM), give N:\n");
        scanf("%d",&N);
        printf("Give the matrix size (NxM), give M:\n");
        scanf("%d",&M);
    }
}

```

```

    printf("\n");
}

start = MPI_Wtime(); /*Inizio a segnare il tempo*/

/*Seed del numero random*/
srand((myrank+1)*10);

/*Broadcast delle variabili essenziali*/
MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&M, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&style, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&print_mode, 1, MPI_INT, 0, MPI_COMM_WORLD);

/*Calcolo come suddividere il carico tra i processori.*/
split_domain(numtasks, N, &split, &plus_i);
int rank_split=((myrank<plus_i) ? (split+1) : (split));

/*Generazione della foresta delegata al rank 0*/
char *splitted_forest=malloc((rank_split*M)*sizeof(char));
//char splitted_forest[rank_split*M];

/*Popolo il bosco*/
for(int i = 0; i < rank_split; i++)
{
    for(int j = 0; j < M; j++)
    {
        if(rand()%100<=probab_generate_tree)/* Probabilità che nasca un a
            splitted_forest[i * M + j]='T';
        else
            splitted_forest[i * M + j]='-';
    }
}

/****FORMATTO SCATTERV E GATHER****/
int send_counts[numtasks];
int send_displacements[numtasks];
if(print_mode){
    //char forest[N*M];
    char *forest=malloc((N*M)*sizeof(char));

    /*Porzione di array che riceverò dalla scatterv*/
    /*da quanti elementi è formato l'array che devo mandare ad ogni pro

```

```

for(int i=0;i<numtasks;i++) send_counts[i]=(i<plus_i) ? M*(split+1)

/*da quale index parto ad ogni processore*/
send_displacements[0]=0;
for(int i=1;i<numtasks;i++)
    send_displacements[i]=(i<=plus_i) ? send_displacements[i-1]+(M*

/*Ricevo i dati per stamparli*/
MPI_Gatherv(splitted_forest, (myrank<plus_i) ? M*(split+1) : M*(spl
/*Stampo la foresta al Day 0*/
if(myrank==0){
    printf("Day 0\n");print_forest(forest,N,M,style,split,plus_i);p
}

free(forest);/*Libero spazio*/
}
/*Variabili essenziali per il while*/
int i=0;
//char temp_forest[rank_split*M];/*array della tecnica double buffer ch
char *temp_forest=malloc((rank_split*M)*sizeof(char));/*array della te
while(1)
{
    //MPI_Scatterv(forest, send_counts, send_displacements, MPI_CHAR, s

//****CODICE SIMULAZIONE
//Variabili essenziali
MPI_Request request_send1=MPI_REQUEST_NULL,request_recv1=MPI_REQUES
MPI_Status status1,status2,status3,status4;
memset(temp_forest,'-',rank_split*M);/*Setto tutti i valori di temp
int count1,count2;

//int bot_send[M],bot_recv[M];
//int top_send[M],top_recv[M];

int *bot_send=malloc(M*sizeof(int));
int *bot_recv=malloc(M*sizeof(int));
int *top_send=malloc(M*sizeof(int));
int *top_recv=malloc(M*sizeof(int));

if(myrank==0)
{
    //int bot_send[M],bot_recv[M];
    //int top_send[M];/*non ci serve per il calcolo*/

    int top,bot;

```

```

if(numtasks!=1){/*Esecuzione parallela*/
    /*Aspetto edge da rank 1 */
    MPI_Irecv(bot_recv, M, MPI_INT, 1, i+1, MPI_COMM_WORLD, &re

    /*Il fuoco si propaga e ritorno se ci sono ai bordi dei fuo
    fire_run_return_set(splitted_forest,temp_forest,rank_split,

    MPI_Isend(bot_send, bot, MPI_INT, 1, i, MPI_COMM_WORLD, &re

}

}else{/*Esecuzione sequenziale*/
    fire_run(splitted_forest,temp_forest,rank_split,M);/*Il fuo
}

if(numtasks!=1){/*Esecuzione parallela*/
    MPI_Wait(&request_send1, &status1);
    MPI_Wait(&request_recv1, &status2);
    MPI_Get_count(&status2, MPI_INT, &count1);
    if(count1>0)
        fire_run_bot_edge_pointer(bot_recv,count1,temp_forest,M

}else{/*Esecuzione sequenziale*/
    ;
}

forest_run(rank_split,M,temp_forest,prob_generate_fire,prob_gen

}
else if(myrank==numtasks-1)
{
    int top,bot;

    MPI_Irecv(top_recv, M, MPI_INT, myrank-1, i, MPI_COMM_WORLD, &r
    fire_run_return_set(splitted_forest,temp_forest,rank_split,M,to

    MPI_Isend(top_send, top, MPI_INT, myrank-1, i+1, MPI_COMM_WORLD

    MPI_Wait(&request_send1, &status1);

```



```

        MPI_Wait(&request_recv1, &status2);

        MPI_Get_count(&status2, MPI_INT, &count2);
        if(count2>0)
            fire_run_top_edge_pointer(top_recv, count2, temp_forest, M);

        forest_run(rank_split, M, temp_forest, prob_generate_fire, prob_gen
    }
else
{

    int top, bot;

    MPI_Irecv(bot_recv, M, MPI_INT, myrank+1, i+1, MPI_COMM_WORLD,
    MPI_Irecv(top_recv, M, MPI_INT, myrank-1, i, MPI_COMM_WORLD, &r

    fire_run_return_set(splitted_forest, temp_forest, rank_split, M, to
    /*send dell'array top & bot*/
    MPI_Isend(top_send, top, MPI_INT, myrank-1, i+1, MPI_COMM_WORLD
    MPI_Isend(bot_send, bot, MPI_INT, myrank+1, i, MPI_COMM_WORLD,

    MPI_Wait(&request_send1, &status1);
    MPI_Wait(&request_recv2, &status2);

    MPI_Get_count(&status2, MPI_INT, &count1);
    if(count1>0)
        fire_run_bot_edge_pointer(bot_recv, count1, temp_forest, M, ran

    MPI_Wait(&request_send2, &status3);
    MPI_Wait(&request_recv1, &status4);

    MPI_Get_count(&status4, MPI_INT, &count2);
    if(count2>0)
        fire_run_top_edge_pointer(top_recv, count2, temp_forest, M);

    forest_run(rank_split, M, temp_forest, prob_generate_fire, prob_gen
}

if(print_mode){
    //char forest[N*M];
    char *forest=malloc((N*M)*sizeof(char));
    /*Ricevo i dati per stamparli*/
    MPI_Gatherv(temp_forest, (myrank<plus_i) ? M*(split+1) : M*(spl
    if(myrank==0){
        printf("Day %d\n", i+1); print_forest(forest, N, M, style, split,
    }
}

```

```

        free(forest);
    }
    /*****/
    strcpy(splitted_forest,temp_forest);/*aggiorno i dati sull'array*/

    free(bot_send);
    free(bot_recv);
    free(top_send);
    free(top_recv);

    i++;
    if(i>5) break;

}

/*Fine codice*/
MPI_Barrier(MPI_COMM_WORLD);
end = MPI_Wtime();

free(splitted_forest);
free(temp_forest);

MPI_Finalize();
if (myrank == 0) { /* Master node scrive su stdout il tempo*/
printf("Time in ms = %f\n", end-start);
}

return 0;
}

```

esso fa affidamento ad una libreria mycollective.h scritta da me per l'esecuzione di funzioni che è la seguente :

```

/*****/
UTILS
*****/
//Suddivisione dominio per righe
void split_domain(int num_p,int domain, int *split,int *plus_i){
    if(num_p>domain){
        fprintf( stderr, "N matrice non compatibile con numero processori\n");
        MPI_Abort(MPI_COMM_WORLD,1);
    }
}

```

```

*split = domain/num_p;
*plus_i = domain%num_p;//Numero di elementi che avranno un +1 sulla ri
}

//Funzione che applica il fuoco sulla foresta e lo fa runnare
void fire_run(char forest0[],char forest1[],int row,int col){

    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            switch (forest0[i * col + j])
            {
                case 'T':/*Se questo albero è stato bruciato da un vicini
                    if(forest1[i * col + j]=='*')
                        ;
                    else
                        forest1[i * col + j]='T';

                    break;

                case '*':
                    /* [x][y]*/
                    /*[-1][-1] [-1][0] [-1][+1] [0][-1] [0][0] [0][+1]
                    forest1[i * col + j]='-';/*Spendo il fuoco che ha b
                    for(int x = -1; x <=1; x++)/*Brucio nei dintorni gl
                    {
                        for(int y = -1; y <= 1; y++)
                        {
                            if( (((i+x)>=0) && ((i+x)<row)) && (((j+y)>
                                if(forest0[(i+x) * col + (j+y)]=='T')/*
                                    forest1[(i+x) * col + (j+y)]='*';
                                }
                            }
                        }
                    }
                    break;
                case 'i':
                    forest1[i * col + j]='i';/*La natura vince sugli um
                    break;*/
                case '-':
                    forest1[i * col + j]='-';/*Vuoto ero e vuoto rimang
                    break;

            }
        }
    }
}

```

```
}
```

```
//Accendo il fuoco sul bordo superiore
```

```
void fire_run_top_edge(char from[],char to[],int N){
    for(int i=0;i<N;i++)
    {
        if(from[i]=='*')
        {
            for(int x=-1;x<=1;x++)
            {
                if((i+x)>=0 && (i+x)<N)
                {
                    if(to[i+x]=='T')/*Gli alberi prendono fuoco*/
                        to[i+x]='*';
                }
            }
        }
    }
}
```

```
//Accendo il fuoco sul bordo inferiore
```

```
void fire_run_bot_edge(char from[],char to[],int N,int split){
    for(int i=0;i<N;i++)
    {
        if(from[i]=='*')
            for(int x=-1;x<=1;x++)
                if((N*(split-1)+i+x)>=(N*(split-1)) && (N*(split-1)+i+x)<N*
                    if(to[N*(split-1)+i+x]=='T')/*Gli alberi prendono fuoco
                        to[N*(split-1)+i+x]='*';
    }
}
```

```
//si accende il fuoco & nasce un seed & cresce un albero
```

```
void forest_run(int rank_split,int N,char forest[],int prob_generate_fire,i

    for(int i = 0; i < rank_split; i++)
    {
        for(int j = 0; j < N; j++)
        {
            switch (forest[i * N + j])
            {
                case 'T':
                    if(rand()%100<=prob_generate_fire)/* Probabilità che un
                        forest[i * N + j]='*';
                    break;
                case 'i':
                    forest[i * N + j]='T'; /* L'albero cresce*/
            }
        }
    }
}
```

```

        break;
    case '-':
        if(rand()%100<=prob_generate_fire)/* Probabilità prenda
            forest[i * N + j]='*';
        else if(rand()%100<=prob_generate_fire+prob_generate_se
            forest[i * N + j]='i';
        break;
    }
}
}

//Setto array top da inviare
void edge_top_send_array(char from[],char to[],int N){
    for(int i=0;i<N;i++)
    {
        to[i]=from[i];
    }
}

//Setto array bot da inviare
void edge_bot_send_array(char from[],char to[],int N,int split){
    for(int i=0;i<N;i++)
    {
        to[i]=from[(N*(split-1))+i];
    }
}

//Controlla se ci sono fuochi ai bordi
// 3 entrambi, 2 bot, 1 top, 0 nulla
int if_send_pack(char forest[],int row,int col){

    int top=0,bot=0;

    for(int i=0;i<row;i++)
    {
        if(forest[i]=='*') top++;
    }
    for(int i=0;i<row;i++)
    {
        if(forest[(row-1) * col + i]=='*') bot++;
    }

    if(top && bot){
        return 3;
    }
}

```

```

    }else if(bot && !(top)){
        return 2;
    }else if(top && !(bot)){
        return 1;
    }else{
        return 0;
    }
}

```

//Funzione che applica il fuoco sulla foresta e ritorna se ci sono fuochi
// 3 entrambi, 2 bot, 1 top, 0 nulla

```

int fire_run_return(char forest0[],char forest1[],int row,int col){

    int top=0,bot=0;
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            switch (forest0[i * col + j])
            {
                case 'T':/*Se questo albero è stato bruciato da un vicini
                    if(forest1[i * col + j]=='*')
                        ;
                    else
                        forest1[i * col + j]='T';

                    break;

                case '*':
                    if(i==0) top++;
                    else if(i==row-1) bot++;
                    /* [x][y]*/
                    /*[-1][-1] [-1][0] [-1][+1] [0][-1] [0][0] [0][+1]
                    forest1[i * col + j]='-';/*Spendo il fuoco che ha bruciato
                    for(int x = -1; x <=1; x++)/*Brucio nei dintorni gli alberi
                    {
                        for(int y = -1; y <= 1; y++)
                        {
                            if( (((i+x)>=0) && ((i+x)<row)) && (((j+y)>=0) && ((j+y)<col))
                                if(forest0[(i+x) * col + (j+y)]=='T')/*
                                    forest1[(i+x) * col + (j+y)]='*';
                        }
                    }
                    break;

```

```

        case 'i':
            forest1[i * col + j]='i';/*La natura vince sugli um
            break;
        case '-':
            forest1[i * col + j]='-';/*Vuoto ero e vuoto rimang
            break;
    }
}
}
if(top && bot){
    return 3;
}else if(bot && !(top)){
    return 2;
}else if(top && !(bot)){
    return 1;
}else{
    return 0;
}
}

```

//Funzione che applica il fuoco sulla foresta e ritorna se ci sono fuochi

// 3 entrambi, 2 bot, 1 top, 0 nulla

```
void fire_run_return_set(char forest0[],char forest1[],int row,int col,int
```

```

    *top=0;
    *bot=0;
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            switch (forest0[i * col + j])
            {
                case 'T':/*Se questo albero è stato bruciato da un vicini
                    if(forest1[i * col + j]=='*')
                        ;
                    else
                        forest1[i * col + j]='T';

                    break;

                case '*':
                    /*Imposto l'array da ritornare*/
                    if(i==0){ top_edge[*top]=j;*top=*top+1;}
                    if(i==row-1){ bot_edge[*bot]=j;*bot=*bot+1;}

```



```
        if(to[N*(split-1)+from[i]+x]=='T')/*Gli alberi prendono fuoco*/
            to[N*(split-1)+from[i]+x]='*';
    }
}
}
}

/*****
STAMPA
*****/

void print_menu(){

    printf("\n\n");
    printf("\033[0;32m█░░██░███████▀███████▀███████▀███████▀███████\n");
    printf("█░███████▀███████▀███████▀███████▀███████▀███████\n");
    printf("█░███████▀███████▀███████▀███████▀███████▀███████\n");
    printf("\033[0;36m      _           ---              \n");
    printf("|_ \\ /   | _| _ _| _ _ . _     _/_ . _ _ _ _ .\n");
    printf("|_|_/   ||||| |-(-)| || (-)   /__| /_ /_-| | |\033[0m\n\n");
    fflush(stdout);
}

void help_print(int N){
    printf("+-----+");
    for(int i = 0; i < N-1; i++)
    {
        printf("------+");
    }
    printf("\n");
}

void help_print_split(int N){
    printf("\033[0;35m+-----+");
    for(int i = 0; i < N-1; i++)
    {
        printf("------+");
    }
    printf("\033[0m\n");
}

//Stampo la foresta
void print_forest(char forest[],int N,int M,int style,int split,int plus_i)
{
    help_print(M);
    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < M; j++)
```

```

        if(style==0){
            if(forest[i * M + j]=='T') printf("| \033[0;32m%c\033[0m
            else if(forest[i * M + j]=='*') printf("| \033[1;31m%c\033
            else if(forest[i * M + j]=='i') printf("| \033[0;33m%c\033
            else printf("|  %c  ", forest[i * M + j]);
        }
        else{
            if(forest[i * M + j]=='T') printf("| 🌳 ");
            else if(forest[i * M + j]=='*') printf("| 🔥 ");
            else if(forest[i * M + j]=='i') printf("| 🌱 ");
            else printf("| - ");
        }
    }
    printf("|\\n");
    if(style==2)
    {
        if(plus_i>0)
        {
            if((i+1)%(split+1)==0)
            {
                plus_i--;
                help_print_split(M);
            }else help_print(M);
        }
        else if((i+1)%split==0 && i<N-1) help_print_split(M);
        else help_print(M);
    }
    else
        help_print(M);
}
}

```

Prova della correttezza

Ad inizio programma, dopo che son state passate le variabili principali via linea di comando, il programma andrà a calcolare quanta porzione di array dovrà avere ogni processore **worker**, esso è definito da questa funzione:

```

//Suddivisione dominio per righe
void split_domain(int num_p,int domain, int *split,int *plus_i){
    if(num_p>domain){

```

```

        fprintf( stderr, "N matrice non compatibile con numero processori\n
MPI_Abort(MPI_COMM_WORLD,1);
    }

    *split = domain/num_p;
    *plus_i = domain%num_p;//Numero di elementi che avranno un +1 sulla ri
}

```

Dopo aver ottenuto il numero di righe da assegnare ad ogni processore, in modo indipendente il worker genera una porzione della foresta nel seguente modo:

```

/*Popolo il bosco*/
for(int i = 0; i < rank_split; i++)
{
    for(int j = 0; j < M; j++)
    {
        if(rand()%100<=prob_generate_tree)/* Probabilità che nasca un a
            splitted_forest[i * M + j]='T';
        else
            splitted_forest[i * M + j]='-';
    }
}

```

Quel che abbiamo ottenuto è un array di alberi di questo tipo:

Day 0

```

+-----+-----+-----+-----+-----+-----+-----+-----+
| - | T | - | T | T | - | - | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | T | T | - | T | - | T | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | T | - | - | - | T | T | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
| - | T | T | T | - | - | T | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | - | T | T | - | T | - | T |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | - | - | T | T | - | T | T |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | T | T | - | T | - | T | T |
+-----+-----+-----+-----+-----+-----+-----+-----+
| - | T | T | - | T | - | T | T |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Successivamente, viene lanciata la funzione `fire_run_return_set()` che ha il compito di propagare il fuoco alle celle vicine, ma dato che questa foresta è ancora “intatta” cioè non ci sono fuochi, viene lanciata la funzione `forest_run()` :

```
//si accende il fuoco & nasce un seed & cresce un albero
void forest_run(int rank_split,int N,char forest[],int prob_generate_fire,i

for(int i = 0; i < rank_split; i++)
{
    for(int j = 0; j < N; j++)
    {
        switch (forest[i * N + j])
        {
            case 'T':
                if(rand()%100<=prob_generate_fire)/* Probabilità che un
                    forest[i * N + j]='*';
                break;
            case 'i':
                forest[i * N + j]='T'; /* L'albero cresce*/
                break;
            case '-':
                if(rand()%100<=prob_generate_fire)/* Probabilità prenda
                    forest[i * N + j]='*';
                else if(rand()%100<=prob_generate_fire+prob_generate_se
                    forest[i * N + j]='i';
                break;
        }
    }
}
}
```

Essa farà in modo che: in accordo con le probabilità [assegnate](#) nascerà un seme d'albero, un seme diventa un albero, e uno spazio vuoto o un albero può prender fuoco. L'output ci genererà la seguente matrice:

Day 1

```
+-----+-----+-----+-----+-----+-----+-----+
| - | T | - | T | T | - | - | - |
+-----+-----+-----+-----+-----+-----+-----+
| T | T | T | i | T | - | T | - |
+-----+-----+-----+-----+-----+-----+-----+
| T | T | - | - | * | T | T | - |
+-----+-----+-----+-----+-----+-----+-----+
```

```

| - | T | T | T | - | - | T | - |
+---+---+---+---+---+---+---+---+
| T | - | T | T | - | T | - | T |
+---+---+---+---+---+---+---+---+
| T | - | - | T | T | * | T | T |
+---+---+---+---+---+---+---+---+
| T | T | T | - | T | - | T | T |
+---+---+---+---+---+---+---+---+
| - | T | T | - | T | - | T | T |
+---+---+---+---+---+---+---+---+

```

Come è possibile notare sono nati dei fuochi, e un seme di albero.

Al prossimo ciclo di esecuzione il fuoco verrà propagato attraverso la funzione precedentemente citata `fire_run_return_set()`, inserendo il risultato in una nuova matrice:

```

//Funzione che applica il fuoco sulla foresta e ritorna se ci sono fuochi
// 3 entrambi, 2 bot, 1 top, 0 nulla
void fire_run_return_set(char forest0[],char forest1[],int row,int col,int

    *top=0;
    *bot=0;
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            switch (forest0[i * col + j])
            {
                case 'T':/*Se questo albero è stato bruciato da un vicini
                    if(forest1[i * col + j]=='*')
                        ;
                    else
                        forest1[i * col + j]='T';

                    break;

                case '*':
                    /*Imposto l'array da ritornare*/
                    if(i==0){ top_edge[*top]=j;*top=*top+1;}
                    if(i==row-1){ bot_edge[*bot]=j;*bot=*bot+1;}

                    /* [x][y]*/
                    /*[-1][-1] [-1][0] [-1][+1] [0][-1] [0][0] [0][+1]
                    forest1[i * col + j]='-';/*Spendo il fuoco che ha b
                    for(int x = -1; x <=1; x++)/*Brucio nei dintorni gl

```

```

        {
            for(int y = -1; y <= 1; y++)
            {
                if( (((i+x)>=0) && ((i+x)<row)) && (((j+y)>
                    if(forest0[(i+x) * col + (j+y)]=='T')/*
                        forest1[(i+x) * col + (j+y)]='*';
                }
            }
        }
        break;
    case 'i':
        forest1[i * col + j]='i';/*La natura vince sugli um
        break;
    case '-':
        forest1[i * col + j]='-';/*Vuoto ero e vuoto rimang
        break;
    }
}
}
}

```

a questo punto viene lanciata `forest_run()` , e ciò che otterremo sarà la seguente matrice:

Day 2

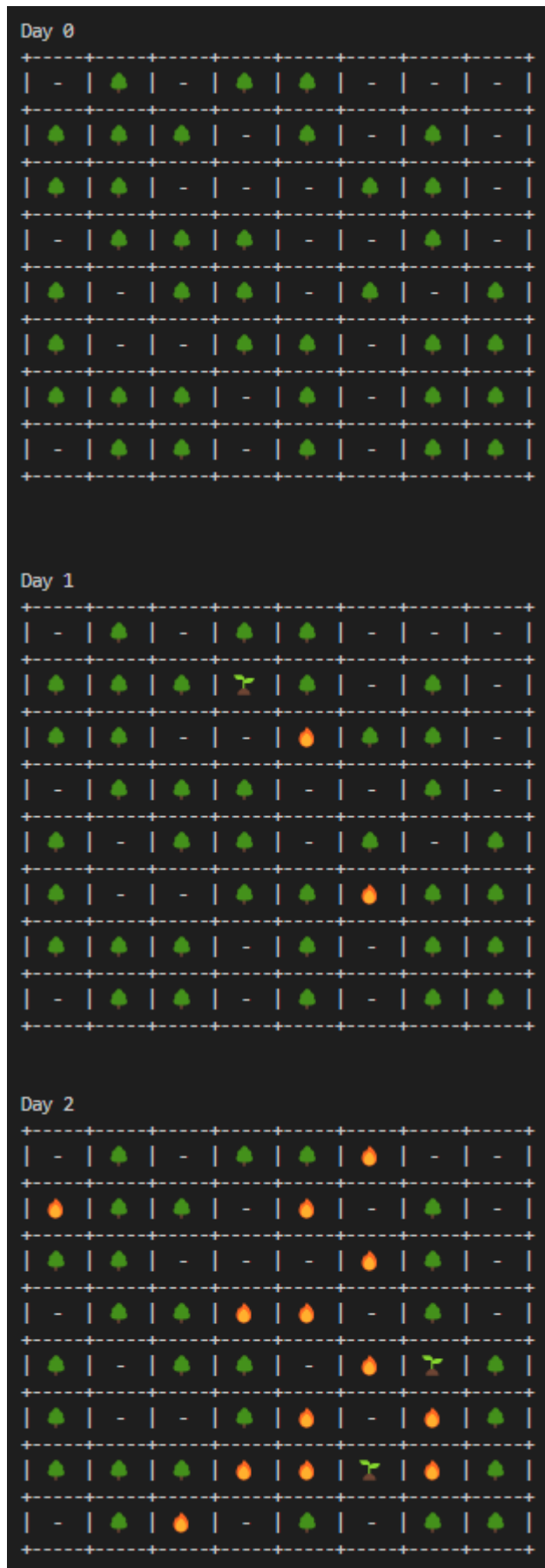
```

+-----+-----+-----+-----+-----+-----+-----+-----+
| - | T | - | T | T | * | - | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
| * | T | T | - | * | - | T | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | T | - | - | - | * | T | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
| - | T | T | * | * | - | T | - |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | - | T | T | - | * | i | T |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | - | - | T | * | - | * | T |
+-----+-----+-----+-----+-----+-----+-----+-----+
| T | T | T | * | * | i | * | T |
+-----+-----+-----+-----+-----+-----+-----+-----+
| - | T | * | - | T | - | T | T |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Essa contiene sia il fuoco propagato, che la generazione dei nuovi elementi.

Ricapitolando in successione l'output ottenuto sarà il seguente(Questa volta con una grafica migliore):



Dovendo favorire il calcolo su più processori la soluzione proposta è eseguita su più processori che comunicano in modo intelligente.

2 o più processori durante la loro esecuzione condividono delle “criticità” dovute al fatto che per completare la loro simulazione hanno bisogno di comunicare con i bordi superiori ed inferiori.

A tal proposito durante l'esecuzione della funzione `fire_run_return_set()` vengono segnati quanti elementi sono presenti al bordo superiore e al bordo inferiore, definiti con il nome di `top` e `bot`, ed in contemporanea vengono salvati in un array l'indice di ogni cella che contiene fuoco.

Sfruttando i dati precedentemente ottenuti, quel che si conosce adesso è se sul bordo ci sono fuochi o meno, e se ci sono a quale indice alloggiano, e si comunica tra processori in modo intelligente lanciando *send dinamiche*

```
MPI_Isend(top_send, top, MPI_INT, myrank-1, i+1, MPI_COMM_WORLD, &request_s
MPI_Isend(bot_send, bot, MPI_INT, myrank+1, i, MPI_COMM_WORLD, &request_sen
```

Il processore che riceverà i dati andrà a controllare se ha ricevuto elementi nel seguente modo:

```
MPI_Get_count(&status2, MPI_INT, &count1);
if(count1>0)
    fire_run_bot_edge_pointer(bot_recv, count1, temp_forest, M, rank_split);

MPI_Get_count(&status4, MPI_INT, &count2);
if(count2>0)
    fire_run_top_edge_pointer(top_recv, count2, temp_forest, M);
```

Se ha ricevuto qualcosa allora semplicemente farà la propagazione del fuoco ai bordi nel seguente modo:

```
//Accendo il fuoco sul bordo superiore
void fire_run_top_edge_pointer(int from[], int N_top, char to[], int N){
    for(int i=0; i<N_top; i++)
    {
        for(int x=-1; x<=1; x++)
        {
            if((from[i]+x)>=0 && (from[i]+x)<N)
            {
                if(to[from[i]+x]=='T')/*Gli alberi prendono fuoco*/
                    to[from[i]+x]='*';
            }
        }
    }
}
```



```

//Accendo il fuoco sul bordo inferiore
void fire_run_bot_edge_pointer(int from[],int N_bot,char to[],int N,int spl
    for(int i=0;i<N_bot;i++)
    {
        for(int x=-1;x<=1;x++){
            if((N*(split-1)+from[i]+x)>=(N*(split-1)) && (N*(split-1)+from[
                if(to[N*(split-1)+from[i]+x]=='T')/*Gli alberi prendono fuo
                    to[N*(split-1)+from[i]+x]='*';
            }
        }
    }
}

```

Il tutto è iterato un numero di volte finito. Al termine di esso viene stampato il tempo impiegato dall'algoritmo.

Fine.

Benchmarks

I benchmarks sono stati lanciati su **Google Cloud Platform**, utilizzando 6 macchine **e2-standard-4** aventi ognuno 4 vCPUs.

Con un totale di 24 vCPUs i risultati ottenuti sono:

Strong scalability

L'algoritmo è stato eseguito su una matrice 10.000x10.000

vCPUs	Tempo	Speed-up
1	32.928257	-
2	16.311782	2,019
3	15.983904	2,06
4	12.809180	2,571
5	9.636696	3,417
6	8.069048	4,081

vCPUs	Tempo	Speed-up
7	6.912726	4,763
8	6.072358	5,423
9	5.734646	5,742
10	5.303405	6,209
11	4.695636	7,013
12	4.288977	7,677
13	3.737966	8,809
14	3.496114	9,419
15	3.240169	10,16
16	3.182754	10,35
17	2.875140	11,45
18	2.734069	12,04
19	2.623592	12,55
20	2.527962	13,03
21	2.372710	13,88
22	2.235581	14,73
23	2.173712	15,15
24	2.057124	16,00694

Come è possibile notare a 24 vCPU si ottiene uno Speed-up di ben 16, sebbene il risultato non è super notevole rimane un ottimo risultato.

Weak scalability

L'algoritmo è stato eseguito su una matrice **np**x10.000.000

vCPUs	Tempo
1	2.868079
2	3.003850
3	4.720806
4	4.882781
5	4.995334
6	4.997530
7	5.064155
8	5.067452
9	5.083282
10	5.059520
11	5.104069
12	5.449976
13	5.156495
14	5.126553
15	5.201446
16	5.699584
17	5.142877
18	5.490366
19	5.171884
20	5.138652
21	5.379677
22	5.127093

vCPUs	Tempo
23	5.197929
24	5.142240

E' stato possibile dimostrare che dando lo stesso numero di elementi a n processori il tempo di esecuzione è pressoché identico.