a1/life_par.c

```c
/*****************************************************
 ************* Conway's game of life ******************
 *****************************************************

    Usage: ./exec ArraySize TimeSteps

    Compile with -DOUTPUT to print output in output.gif
    (You will need ImageMagick for that - Install with
    sudo apt-get install imagemagick)
    WARNING: Do not print output for large array sizes!
    or multiple time steps!
 *****************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <omp.h>

#define FINALIZE "\
convert -delay 20 `ls -1 out*.pgm | sort -V` output.gif\n\
rm *pgm\n\
"

int **allocate_array(int N);
void free_array(int **array, int N);
void init_random(int **array1, int **array2, int N);
void print_to_pgm(int **array, int N, int t);

int main(int argc, char *argv[])
{
    int N;                    // array dimensions
    int T;                    // time steps
    int **current, **previous; // arrays - one for current timestep, one for previous timestep
    int **swap;               // array pointer
    int i, j, t, nbrs;        // helper variables

    double time; // variables for timing
    struct timeval ts, tf;

    /*Read input arguments*/
    if (argc != 3)
    {
        fprintf(stderr, "Usage: ./exec ArraySize TimeSteps\n");
        exit(-1);
    }
    else
    {
        N = atoi(argv[1]);
        T = atoi(argv[2]);
    }

```

```
52      /*Allocate and initialize matrices*/
53      current = allocate_array(N);  // allocate array for current time step
54      previous = allocate_array(N); // allocate array for previous time step
55
56      init_random(previous, current, N); // initialize previous array with pattern
57
58  #ifdef OUTPUT
59      print_to_pgm(previous, N, 0);
60  #endif
61
62      /*Game of Life*/
63
64      gettimeofday(&ts, NULL);
65
66      gettimeofday(&ts, NULL);
67
68      for (t = 0; t < T; ++t)
69      {
70
71  /* Parallelize rows; implicit barrier at loop end */
72  /*  scedule is static
73      (i, j) as loop indices are private
74      (N, previous, current) are shared, as they are enclosed by the loop
75      nbrs must be private
76      by default */
77  /* Kept here for clarity */
78  #pragma omp parallel for schedule(static) private(i, j, nbrs) shared(N, previous, current)
79          for (i = 1; i < N - 1; ++i)
80          {
81              for (j = 1; j < N - 1; ++j)
82              {
83                  nbrs =
84                      previous[i + 1][j + 1] + previous[i + 1][j] + previous[i + 1][j - 1] +
85                      previous[i][j - 1] + previous[i][j + 1] +
86                      previous[i - 1][j - 1] + previous[i - 1][j] + previous[i - 1][j + 1];
87
88                  current[i][j] = (nbrs == 3 || (previous[i][j] + nbrs == 3)) ? 1 : 0;
89              }
90          } /* implicit barrier here: all threads finished step t */
91
92  #ifdef OUTPUT
93          print_to_pgm(current, N, t + 1); /* single thread here: we're back in serial */
94  #endif
95
96          /* Safe to swap: we're outside the parallel region created by 'parallel for' */
97          swap = current;
98          current = previous;
99          previous = swap;
100     }
101
102     gettimeofday(&tf, NULL);
103     time = (tf.tv_sec - ts.tv_sec) + (tf.tv_usec - ts.tv_usec) * 0.000001;
104
105     free_array(current, N);
```

```c
    free_array(previous, N);
    printf("GameOfLife: Size %d Steps %d Time %lf\n", N, T, time);
#ifdef OUTPUT
    system(FINALIZE);
#endif
}

int **allocate_array(int N)
{
    int **array;
    int i, j;
    array = malloc(N * sizeof(int *));
    for (i = 0; i < N; i++)
        array[i] = malloc(N * sizeof(int));
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            array[i][j] = 0;
    return array;
}

void free_array(int **array, int N)
{
    int i;
    for (i = 0; i < N; i++)
        free(array[i]);
    free(array);
}

void init_random(int **array1, int **array2, int N)
{
    int i, pos, x, y;

    for (i = 0; i < (N * N) / 10; i++)
    {
        pos = rand() % ((N - 2) * (N - 2));
        array1[pos % (N - 2) + 1][pos / (N - 2) + 1] = 1;
        array2[pos % (N - 2) + 1][pos / (N - 2) + 1] = 1;
    }
}

void print_to_pgm(int **array, int N, int t)
{
    int i, j;
    char *s = malloc(30 * sizeof(char));
    sprintf(s, "out%d.pgm", t);
    FILE *f = fopen(s, "wb");
    fprintf(f, "P5\n%d %d 1\n", N, N);
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            if (array[i][j] == 1)
                fputc(1, f);
            else
                fputc(0, f);
    fclose(f);
```

```
160     free(s);
161 }
162
163
```