Συστήματα Παράλληλης Επεξεργασίας
Πρώτη Εργαστηριακή Άσκηση

Αναστασία-Χριστίνα Λίβα 03119029
Γιώργος Μυστριώτης 03119065
Νικόλας Σταματόπουλος 03119020

Νοέμβριος 2023

Εξοικείωση με το περιβάλλον προγραμματισμού - Conway's Game of Life

Για την ανάπτυξη του παράλληλου προγράμματος στο μοντέλο κοινού χώρου διευθύνσεων με τη χρήση του OpenMP κάναμε χρήση του declaration #pragma omp parallel for. Το declaration αυτό επιτρέπει στο for loop να εκτελεστεί παράλληλα σε όσα threads γίνονται declared από το enviroment variable OMP_NUM_THREADS. Με τον τρόπο αυτό έχουμε γρηγορότερη εκτέλεση του προγράμματός μας και καλύτερη αξιοποίηση των πόρων του συστήματος.

Συνεπώς ο τελικός κώδικας του Game_of_Life.c είναι ο εξής:

```
/*****************************************************
 ************* 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>
```

```c
#include <stdlib.h>
#include <sys/time.h>

#define FINALIZE "convert -delay 20 `ls -1 out*.pgm | sort -V` output.gif\nrm *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 t, i, j, 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]);
  }

  /*Allocate and initialize matrices*/
  current = allocate_array(N);      //allocate array for current time step
  previous = allocate_array(N);       //allocate array for previous time step

  init_random(previous, current, N);  //initialize previous array with pattern

  #ifdef OUTPUT
  print_to_pgm(previous, N, 0);
  #endif

  /*Game of Life*/

  gettimeofday(&ts,NULL);

```

```
60   for ( t = 0 ; t < T ; t++ ) {
61     #pragma omp parallel for shared(N, previous, current) private(i, j,
       nbrs)
62     for ( i = 1 ; i < N-1 ; i++ )
63       for ( j = 1 ; j < N-1 ; j++ ) {
64         nbrs = previous[i+1][j+1] + previous[i+1][j] + previous[i+1][j
     -1] \
65                 + previous[i][j-1] + previous[i][j+1] \
66                 + previous[i-1][j-1] + previous[i-1][j] + previous[i-1][
     j+1];
67         if ( nbrs == 3 || ( previous[i][j]+nbrs ==3 ) )
68           current[i][j]=1;
69         else
70           current[i][j]=0;
71       }
72
73     #ifdef OUTPUT
74     print_to_pgm(current, N, t+1);
75     #endif
76     //Swap current array with previous array
77     swap=current;
78     current=previous;
79     previous=swap;
80
81   }
82   gettimeofday(&tf,NULL);
83   time=(tf.tv_sec-ts.tv_sec)+(tf.tv_usec-ts.tv_usec)*0.000001;
84
85   free_array(current, N);
86   free_array(previous, N);
87   printf("GameOfLife: Size %d Steps %d Time %lf\n", N, T, time);
88   #ifdef OUTPUT
89   system(FINALIZE);
90   #endif
91 }
92
93 int ** allocate_array(int N) {
94   int ** array;
95   int i,j;
96   array = malloc(N * sizeof(int*));
97   for ( i = 0; i < N ; i++ )
98     array[i] = malloc( N * sizeof(int));
99   for ( i = 0; i < N ; i++ )
100    for ( j = 0; j < N ; j++ )
101      array[i][j] = 0;
102  return array;
103 }
104
105 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);
  free(s);
}
```

Ο κώδικας γίνεται compiled με χρήση του παρακάτω Makefile:

```
CC = gcc
CFLAGS = -O3 -Wall -fopenmp -DOUTPUT

all: Game_Of_Life

Game_Of_Life: Game_Of_Life.c
  $(CC) $(CFLAGS) -o Game_Of_Life Game_Of_Life.c
```

Το οποίο εκτελείται μέσω του παρακάτω bash script στους κόμβους του εργαστηρίου:

make_on_queue.sh:

```
#!/bin/bash

```

```
## Give the Job a descriptive name
#PBS -N make_GOL

## Output and error files
#PBS -o make_GOL.out
#PBS -e make_GOL.err

## How many machines should we get?
#PBS -l nodes=1:ppn=2

#PBS -l walltime=00:10:00

## Start
## Run make in the src folder (modify properly)

cd /home/parallel/parlab13
make
```

Η εντολή που δώσαμε για να γίνει το make μέσω του παραπάνω script ήταν: qsub –q parlab make_on_queue.sh

Αφού δημιουργήθεί το executable μέσω της παραπάνω διαδικασίας, μπορούμε να το δρομολογήσουμε για όλα μας τα ζητούμενα με ένα νέο bash script, το οποίο θα εκτελεί όλα τα versions του executable που χρειαζόμαστε, στους κόμβους του εργαστηρίου. Η δρομολόγηση του script γίνεται με την εντολή: qsub –q parlab run_on_queue.sh.

run_on_queue.sh:

```
#!/bin/bash

## Give the Job a descriptive name
#PBS -N run_GOL

## Output and error files
#PBS -o run_GOL.out
#PBS -e run_GOL.err

## How many machines should we get?
#PBS -l nodes=1:ppn=8

#PBS -l walltime=00:10:00

## Start
```

```
## Run make in the src folder (modify properly)

cd /home/parallel/parlab13

for num_threads in 1 2 4 6 8
do
    printf "\n"
    printf "Number of Threads used = $num_threads\n"
    export OMP_NUM_THREADS=$num_threads
    ./Game_Of_Life 64 1000
    ./Game_Of_Life 1024 1000
    ./Game_Of_Life 4096 1000
    printf "\n"
done
```

Όπως βλέπουμε στο παραπάνω script εκτελούμε για 1, 2, 4, 6 και 8 threads το Game_Of_Life με παραμέτρους: μεγέθη 64, 1024 και 4096 και γενιές 1000. Για κάθε αριθμό threads βλέπουμε πως πριν εκτελέσουμε το executable κάνουμε export το environment variable OMP_NUM_THREADS ίσο με τον αριθμό των threads κάθε φορά.

Αντίστοιχα, σε περίπτωση που χρειαζόμασταν μονάχα ένα executable (για να μην αλλάζουμε το βασικό μας run_on_queue.sh script) είχαμε το run_on_queue_one.sh, το οποίο δρομολογούσαμε με την qsub –q parlab make_on_queue_one.sh.

run_on_queue_one.sh:

```
#!/bin/bash

## Give the Job a descriptive name
#PBS -N run_GOL_one

## Output and error files
#PBS -o run_GOL_one.out
#PBS -e run_GOL_one.err

## How many machines should we get?
#PBS -l nodes=1:ppn=8

#PBS -l walltime=00:10:00

## Start
## Run make in the src folder (modify properly)

cd /home/parallel/parlab13
```

```
19
20 num_threads=8
21
22 printf "\n"
23 printf "Number of Threads used = $num_threads\n"
24 export OMP_NUM_THREADS=$num_threads
25 ./Game_Of_Life 1024 1000
26 printf "\n"
```

Τρέχοντας, λοιπόν, το run_on_queue.sh παίρνουμε:

```
parlab13@scirouter:~$ cat run_GOL.out

Number of Threads used = 1
GameOfLife: Size 64 Steps 1000 Time 0.023094
GameOfLife: Size 1024 Steps 1000 Time 10.967422
GameOfLife: Size 4096 Steps 1000 Time 175.799121


Number of Threads used = 2
GameOfLife: Size 64 Steps 1000 Time 0.013497
GameOfLife: Size 1024 Steps 1000 Time 5.456104
GameOfLife: Size 4096 Steps 1000 Time 88.182485


Number of Threads used = 4
GameOfLife: Size 64 Steps 1000 Time 0.010173
GameOfLife: Size 1024 Steps 1000 Time 2.724551
GameOfLife: Size 4096 Steps 1000 Time 44.442680


Number of Threads used = 6
GameOfLife: Size 64 Steps 1000 Time 0.009549
GameOfLife: Size 1024 Steps 1000 Time 1.833963
GameOfLife: Size 4096 Steps 1000 Time 31.259575


Number of Threads used = 8
GameOfLife: Size 64 Steps 1000 Time 0.009916
GameOfLife: Size 1024 Steps 1000 Time 1.375487
GameOfLife: Size 4096 Steps 1000 Time 28.215872
```

Figure 1: Complete runs from run_on_queue.sh

Όπως μπορούμε να δούμε με χρήση περισσότερων threads και την αξιοποίηση του parallel for παίρνουμε πολύ καλύτερα αποτελέσματα.

Για τις γραφικές αναπαραστάσεις (gifs) χρησιμοποιήθηκε η παράμετρος -DOUTPUT όπως φαίνεται στο Makefile. Μέσω αυτής γινόταν χρήση του imagemagick και παράγονταν κάποια .pgm files τα οποία τελικά δημιουργούσαν το gif. Για διάφορα sizes τα gifs υπάρχουν στον φάκελο που παραδόθηκε.

Σημείωση: Τα μεγέθη 1024 και 4096 ήταν πολύ μεγάλα για να παραχθούν τα gifs τους, όμως κατά ένα run διαγράφθηκαν στο ενδιάμεσο κάποια pgms του 1024 και έτσι το gif μπόρεσε και παράχθηκε (προφανώς με κάποια λιγότερα frames).