Parrallel Proccessing – HW 3

Parallel Computations with OpenMP
Goal: Creating a galaxy simulation

Dvir Zaguri 315602284
Yehonatan Arama 207938903

1. הקדמה

נדרשנו לכתוב תוכנית שמסמלצת התנהגות פיזיקלית של גלקסיה בעלת מספר כוכבים כאשר ביניהם אינטרקציה של משיכה מכוחות כבידה. התוכנית מריצה המון לולאות ופעולות איטרטיביות לצורך החישובים הכוחות המיקומים והתאוצות הפועלים על כל כוכב בכל איטרציה בתוכנית. את התוכנית מקבלנו באמצעות שימוש ב-Open_MP והקצאת פעולות שיכולות לעבוד באופן מקבילי ל-threads נפרדים.
התוכנית נכתבה בשפת C.

2. מהלך הניסוי

<u>השיטה</u>

התוכנית שלנו נכתבה בהתאם לדרישת הפרמטרים הפיזיקליים שהוגדרו במסמך המטלה. בנינו struct של כוכב לכת כך שנוכל לתת לכל כוכב מאפיינים. הגדרנו בשדות של ה-struct עבור כל כוכב מיקום ומהירות לפי קואורדינטה (x,y) כאשר נירמלנו את הערכים של המיקום בין 0 ל-100 ונירמלנו במשתנה בגודל המכפלה של שנת אור אחת. את המהירות חישבנו את תחום ההגרלה של מהירות עבור ציר x ולפיה נקבעה המהירות בציר y באופן חד ערכי. בהתאם למיקומים של כל הכוכבים והמהירויות שלהם חושב המיקום הבא שלהם לפי כל צעד (כמובן כתלות באורך הזמן שהוגדר עבור כל 'צעד' בתוכנית. לאחר מכן חושב הכוח הפועל על כל כוכב לפי המיקום של מרכז מסת הגלקסיה בפני כל כוכב ולפיו חישוב הכוח הפועל עליו – משם את התאוצה שלו ולפי גודל כל צעד בזמן חושבה המהירות החדשה של כל כוכב וכך פועלת התוכנית באופן איטרטיבי.

<u>פירוט הפתרון</u>

<u>סעיף 1</u>:

צירוף הקוד עבור התוכנית:

```
/*
Compilation command:
gcc HW3_1.c -o hw3 -fopenmp -lpng -lm
*/

#include "omp.h"
#include "stdio.h"
#include "stdlib.h"
#include <time.h>
#include <math.h>
#include <png.h>

#define N 1000
#define N_threads 8
#define steps 20000

#define WIDTH 400
#define HEIGHT 400

struct star{
    double x_pos;
    double y_pos;
    double x_vel;
    double y_vel;
};
```

```
double  G;
double  starMass;
double  ly_norm;
double  GMMN_1;
double  TIME = 3153600;
struct star galaxy[N];
png_byte image_data[WIDTH * HEIGHT * 4];

void gen_star(int index){
    double vel_abs = ((double)rand()/ RAND_MAX)*200 + 100;
    galaxy[index].x_pos =  ((double)rand() / RAND_MAX) * ly_norm;
    galaxy[index].y_pos =  ((double)rand() / RAND_MAX) * ly_norm;
    galaxy[index].x_vel =  ((double)rand() / RAND_MAX) * vel_abs;
    galaxy[index].y_vel = sqrt(vel_abs*vel_abs -
galaxy[index].x_vel*galaxy[index].x_vel );
}

void initialize_stars(){
    #pragma omp parallel for
    for(int i=0; i<N; i++){
        gen_star(i);
    }
}

double Calc_mass_centerX() {
    double totalXpos = 0;
    #pragma omp parallel for reduction (+:totalXpos)
    for (int i=0; i < N; i++)
    {
        totalXpos += galaxy[i].x_pos;
    }
    return  totalXpos;
}

double Calc_mass_centerY() {
    double totalYpos = 0;
    #pragma omp parallel for reduction (+:totalYpos)
    for (int i=0; i < N; i++)
    {
        totalYpos += galaxy[i].y_pos;
    }
    return  totalYpos;
}

double CalcNewPos (double oldPos , double vel)
{
    double newPos = oldPos + vel*TIME;
    return fmod(newPos,ly_norm);
}

double CalcF (double oldPos,double massCenter)
{
    double radius = pow(massCenter-oldPos,2);
```

```
    return GMMN_1/radius;
}

void simulate_galaxy(){
    double TOTAL_mass_center_x = Calc_mass_centerX();
    double TOTAL_mass_center_y = Calc_mass_centerY();
    double x_pos;
    double y_pos;
    double x_vel;
    double y_vel;
    double Fx;
    double Fy;
    #pragma omp parallel for private(x_pos,y_pos,Fx,Fy,x_vel,y_vel)
    for (int i= 0; i < N; i++)
    {
        x_pos = galaxy[i].x_pos;
        y_pos = galaxy[i].y_pos;
        x_vel = galaxy[i].x_vel;
        y_vel = galaxy[i].y_vel;
        Fx = CalcF(x_pos , TOTAL_mass_center_x);
        Fy = CalcF(y_pos , TOTAL_mass_center_y);
        x_pos = CalcNewPos(x_pos , x_vel);
        y_pos = CalcNewPos(y_pos , y_vel);
        galaxy[i].x_vel += (TIME*Fx)/starMass;
        galaxy[i].y_vel += (TIME*Fy)/starMass;
        galaxy[i].x_pos = x_pos;
        galaxy[i].y_pos = y_pos;
    }

}

void set_pixel(png_bytep pixel, int x, int y, int r, int g, int b, int a)
{
    pixel[(y * WIDTH + x) * 4] = r;
    pixel[(y * WIDTH + x) * 4 + 1] = g;
    pixel[(y * WIDTH + x) * 4 + 2] = b;
    pixel[(y * WIDTH + x) * 4 + 3] = a;
}

void save_png(const char *filename, int width, int height, png_bytep
image_data) {
    FILE *fp = fopen(filename, "wb");
    if (!fp) {
        fprintf(stderr, "Failed to open %s for writing\n", filename);
        return;
    }

    png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!png) {
        fclose(fp);
        fprintf(stderr, "Failed to create PNG write struct\n");
        return;
    }
```

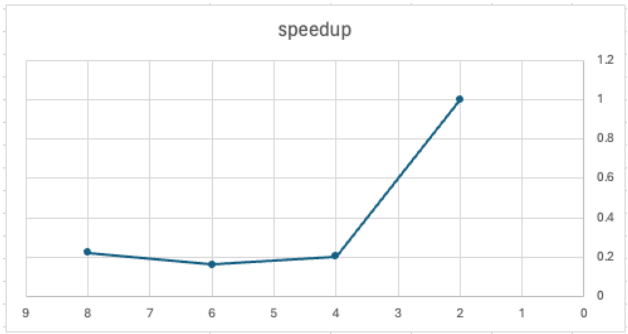
```
    png_infop info = png_create_info_struct(png);
    if (!info) {
        fclose(fp);
        png_destroy_write_struct(&png, NULL);
        fprintf(stderr, "Failed to create PNG info struct\n");
        return;
    }

    png_init_io(png, fp);

    png_set_IHDR(png, info, width, height, 8, PNG_COLOR_TYPE_RGBA,
PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_DEFAULT,
PNG_FILTER_TYPE_DEFAULT);

    png_bytepp rows = (png_bytepp)malloc(height * sizeof(png_bytep));
    for (int y = 0; y < height; ++y) {
        rows[y] = (png_bytep)(image_data + y * width * 4);
    }

    png_set_rows(png, info, rows);
    png_write_png(png, info, PNG_TRANSFORM_IDENTITY, NULL);

    fclose(fp);
    png_destroy_write_struct(&png, &info);
    free(rows);
}

void printGalaxy(char *filename){
    #pragma omp parallel for
    for (int y = 0; y < HEIGHT; ++y) {
        for (int x = 0; x < WIDTH; ++x) {
            set_pixel(image_data, x, y, 0, 0, 0, 255);
        }
    }


    #pragma omp parallel for
    for (int i=0;i<N;i++){
        int x = (galaxy[i].x_pos/ly_norm)*WIDTH;
        int y = (galaxy[i].y_pos/ly_norm)*HEIGHT;
        //printf("star:%d\ngalaxy[i].x_pos:%f \ngalaxy[i].y_pos:%f\nx:%d
y: %d\n",i,galaxy[i].x_pos,galaxy[i].y_pos,x,y);
        set_pixel(image_data, x,y , 255, 255, 100, 0);
    }


    // Save image as PNG
    save_png(filename, WIDTH, HEIGHT, image_data);


}

int main (int argc, char **argv){
```

```
    G = 6.647*pow(10,-17);
    starMass = 2*pow(10,30);
    ly_norm = 9*pow(10,14);
    GMMN_1 = starMass*starMass*(N-1)*G;
    double startTime;
    double endTime;
    omp_set_num_threads(N_threads);
    startTime = omp_get_wtime();
    initialize_stars();
    for (int i = 0; i < steps; i++)
    {
        if (i==0)
            printGalaxy("PictureTime0");
        if (i==(steps/2))
            printGalaxy("PictureTimeHalf");
        if (i==steps-1)
            printGalaxy("PictureTimeEnd");
        simulate_galaxy();
    }
    endTime = omp_get_wtime();
    printf("Run time of the program: %f seconds\n", endTime-startTime);
    return 0;
}
```

תיעוד ההרצות לסעיפים 2,3 :

<u>סעיף 2 :</u>

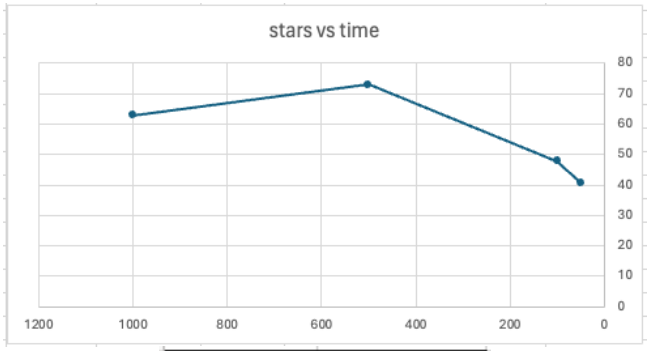
speed-up של התוכנית בחלוקה ל-2 threads לעומת 4,6,8 threads.



processors	speedup		tn
2	1		13.880415
4	0.203466684		68.219596
6	0.159669195		86.932329
8	0.220950147		62.821479

<u>סעיף 3 :</u>

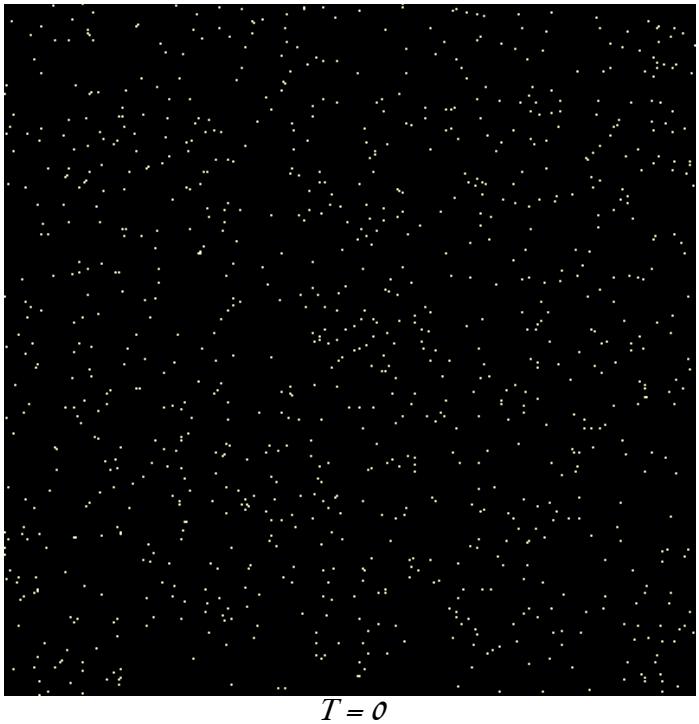
הדפסת יחס הזמנים בתוכנית בגרף הנדרש $(vertical = \frac{t_N}{t_{50}}, horizontal = N\ stars)$
$N=50,100,500,1000$



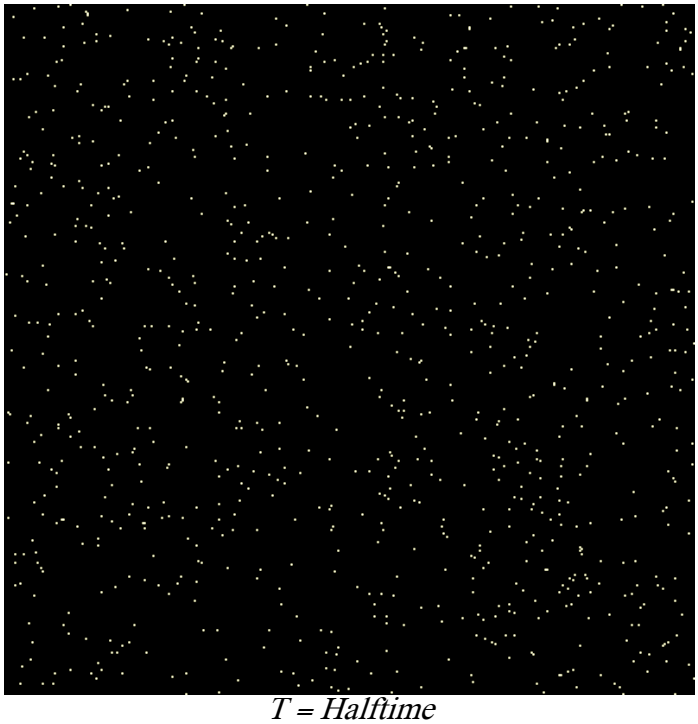
N (stars)	tN
50	40.834349
100	47.956928
500	72.971291
1000	62.821479

סעיף 4:

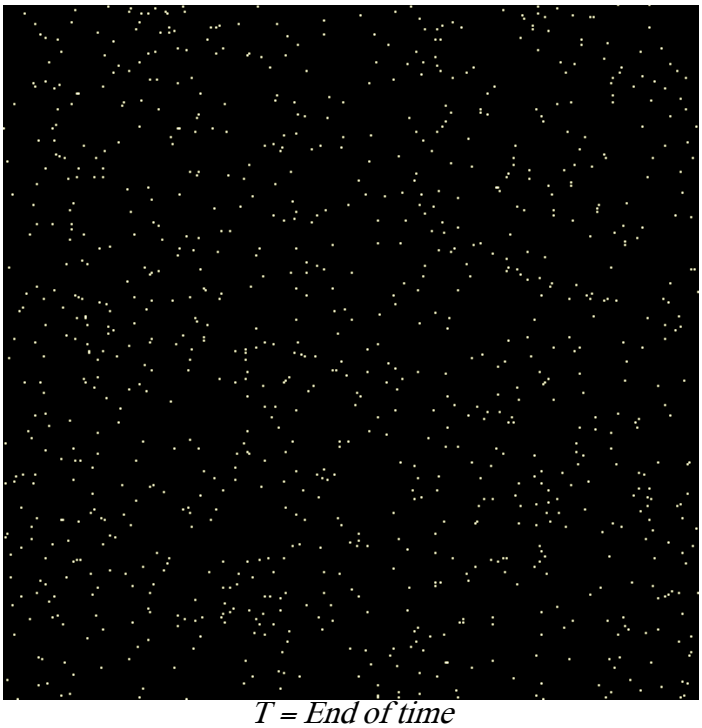
צילום הגלקסיה בשלושה מצבים. איתחול הגלקסיה, מחצית זמן הריצה וסוף הריצה עבור 1000 כוכבים וחלוקה ל-8 threads בפורמט png.



T = 0



T = Halftime



T = End of time

3. סיכום ומסקנות

ניתן לראות שבגלל שהרצנו את התוכנית על ה-vm שהוקצו לה שני processors אז קל לראות שהקוד רץ בצורה האופטימלית עבור פרמטרים זהים כאשר מספר הthreads של התוכנית הוא בדיוק 2. כאשר אנחנו מחלקים את התוכנית למספר גבוהה יותר של threads אז מערכת ההפעלה מנסה לדמות מקביליות למרות שאין לנו יותר משאבי חישוב בריצה של התוכנית ומחלקת את פעולות החישוב של כל ה-threads לשתי הליבות וזה עולה לנו בזמן ריצה.