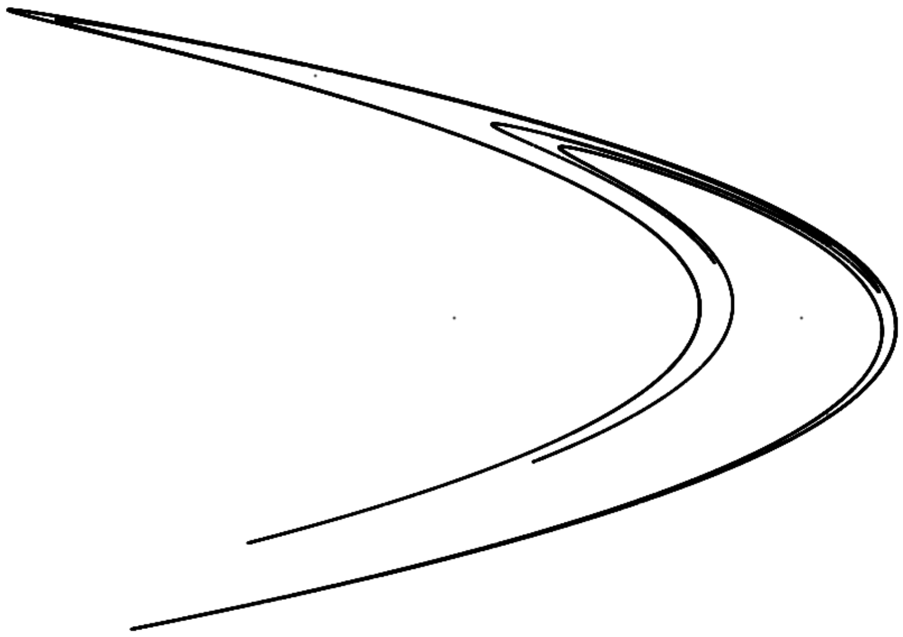


Henon Map

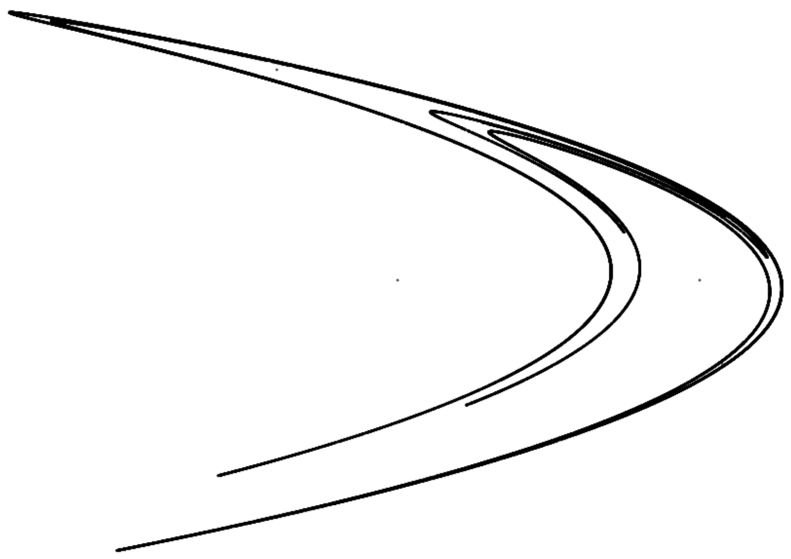
A THEORETICAL ANALYSIS

Samarth Sudarshan Inamdar (CED18I045) | High Performance Computing
August 2021

Department of Computer Science and Engineering,
Indian Institute of Information Technology Design and Manufacturing Kancheepuram, Chennai

Table of Contents

Introduction



The Henon map is an iterated discrete-time dynamical system that exhibits chaotic behavior in two-dimension. It is sometimes called Hénon-Pomeau attractor/map is a discrete-time dynamical system. It is one of the most studied examples of dynamical systems that exhibit chaotic behavior.

What is Henon Map?

The Henon map presents a simple two-dimensional invertible iterated map with quadratic nonlinearity and chaotic solutions called strange attractor. Strange attractors are a link between the chaos and the fractals.

The Henon map takes a point (x_n, y_n) in the plane and maps it to a new point. The below equation helps us to find the next point based on the previous point.

$$\begin{aligned} x_{n+1} &= 1 - a x_n^2 + y_n \\ y_{n+1} &= b x_n . \end{aligned}$$

Code (Henon Map)

SERIAL CODE

```
// Compilation
// g++ main.cpp -o main -lGL -lGLU -lglut -lm -lGLEW
#include <stdio.h>
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/gl.h>
#include <GL/freeglut.h>

typedef struct point
{
    double x;
    double y;
} Point;

int start = 0, end = 100000;
double a = 1.4;
double b = 0.3;
static GLfloat theta[] = {0.0, 0.0, 0.0};
GLint axis = 1;
Point initial = {1.0, 1.0};

Point eq(Point prev, Point pprev)
{
    Point p;
    p.y = b * prev.x;
    p.x = 1 - a * prev.x * prev.x + b * pprev.x;
    return p;
}

void henon_gen()
{
    int n = end - start;
    Point *coord = (Point *)malloc(n * sizeof(Point));
    coord[0] = initial;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    glPointSize(1.2);
    coord[1].x = 1 - a * coord[0].x * coord[0].x + coord[0].y;
    coord[1].y = b * coord[0].x;
    for (int i = 1; i < n; i++)
    {
        coord[i] = eq(coord[i - 1], coord[i - 2]);
        glBegin(GL_POINTS);
        glColor3f(1, 1, 1);
        glVertex2f(coord[i].x, coord[i].y);
        glEnd();
        glFlush();
        glutSwapBuffers();
    }
}

void spinCube()
{
    if (theta[axis] > 360.0)
        theta[axis] -= 360.0;
    else if (theta[axis] < 0)
        theta[axis] += 360.0;
    glutPostRedisplay();
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.5, 1.5, -1.5, 1.5, -1.5, 1.5);
    glMatrixMode(GL_MODELVIEW);
}

int main()
{
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Henon Map");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(henon_gen);
    glutIdleFunc(spinCube);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    glutLeaveMainLoop();
}
```

PARALLEL CODE

```
// Compilation
// g++ -O0 -fopenmp -o mainomp mainomp.cpp -lm -lGL -lGLU -lglut -
lGLEW
#include <stdio.h>
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/gl.h>
#include <GL/freeglut.h>
#include <omp.h>

typedef struct point
{
    double x;
    double y;
} Point;

int start = 0, end = 100000;

double a = 1.4;
double b = 0.3;

static GLfloat theta[] = {0.0, 0.0, 0.0};
GLint axis = 1;

Point initial = {1.0, 1.0};

Point eq(Point prev, Point pprev)
{
    Point p;
    double temp1, temp2, temp3;

#pragma omp parallel sections
    {
#pragma omp section
        temp1 = a * prev.x * prev.x;
#pragma omp section
        temp2 = b * pprev.x;
#pragma omp section
        p.y = b * prev.x;
    }
#pragma omp barrier
    p.x = 1 - temp1 + temp2;
    return p;
}
```

```
void henon_gen()
{
    int n = end - start;
    Point *coord = (Point *)malloc(n * sizeof(Point));
    coord[0] = initial;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    glPointSize(1.2);
    coord[1].x = 1 - a * coord[0].x * coord[0].x + coord[0].y;
    coord[1].y = b * coord[0].x;
    float startTime, endTime, execTime;
    startTime = omp_get_wtime();
    for (int i = 1; i < n; i++)
    {
        coord[i] = eq(coord[i - 1], coord[i - 2]);
        glBegin(GL_POINTS);
        glColor3f(1, 1, 1);
        glVertex2f(coord[i].x, coord[i].y);
        glEnd();
        glFlush();
        glutSwapBuffers();
    }
    endTime = omp_get_wtime();
    execTime = endTime - startTime;
    printf("%f \n", execTime);
    glutLeaveMainLoop();
}

void spinCube()
{
    if (theta[axis] > 360.0)
        theta[axis] -= 360.0;
    else if (theta[axis] < 0)
        theta[axis] += 360.0;
    glutPostRedisplay();
}
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.5, 1.5, -1.5, 1.5, -1.5, 1.5);
    glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char **argv)
{
    if (argc == 2)
        omp_set_num_threads(atoi(argv[1]));
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Henon Map");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(henon_gen);
    glutIdleFunc(spinCube);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

Critical Section

```
Point eq(Point prev, Point pprev)
{
    Point p;
    double temp1, temp2, temp3;

#pragma omp parallel sections
    {
#pragma omp section
        temp1 = a * prev.x * prev.x;
#pragma omp section
        temp2 = b * pprev.x;
#pragma omp section
        p.y = b * prev.x;
    }
#pragma omp barrier
    p.x = 1 - temp1 + temp2;
    return p;
}
```

Above code snippet is the critical section of the code i.e., major computation is being done here as this part runs most frequently.

This part calculates the next point using the previous point using the henon equation stated above. Since there is dependency of the current point with the previous point, this gives us a very small room for parallelization and performance improvement.

Pragma sections are used to do the computations for each part that are independent of each other in parallel by multiple threads, which in turn increases the efficiency of the program.

The final calculation requires the values calculated in the sections. Hence, we add a barrier that doesn't run until all the threads complete the execution the assigned sections.

Here, number of sections used are 3. So, ideally the performance should remain the same for thread count of 3 or above. In contrast, the performance degrades due to the communication overhead and thread scheduling.

Observations

COMPILATION AND EXECUTION

Enabling OpenMP environment use -fopenmp flag while compiling using gcc.

```
g++ -O0 -fopenmp -o main_omp main_omp.cpp -lm -lGL -lGLU -lglut -lGLEW
```

For execution, use

```
./main_omp NUMBER_OF_THREADS
```

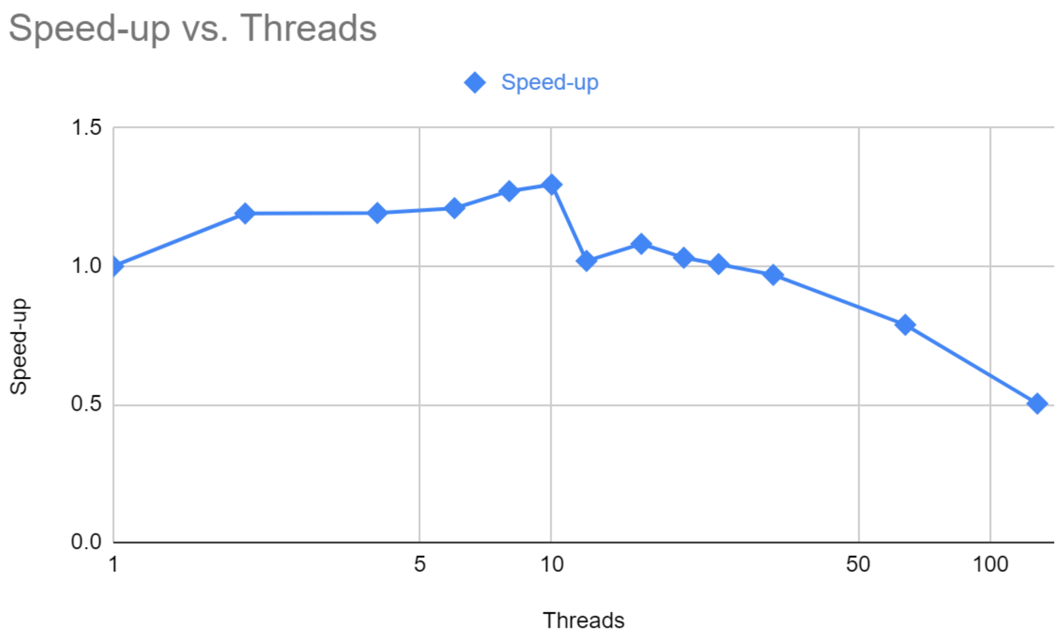
Or to change the number of threads in the environment variables,

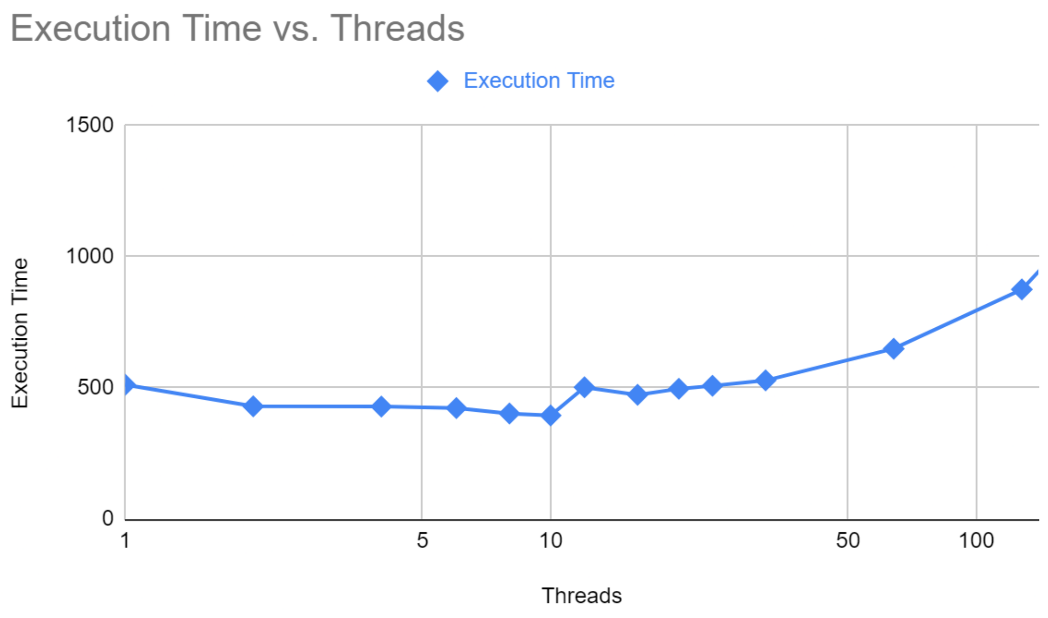
```
export OMP_NUM_THREADS = 1
```

OBSERVATION TABLE

Thread Count (P)	Execution Time (T(P))	Speedup (S(P))	Parallelization Fraction (f(P))
1	512.28418	1	N/A
2	430.14502	1.190956901	32.06781049
4	429.464844	1.19284311	21.55557123
6	423.345215	1.210086147	20.83350651
8	402.692383	1.272147678	24.4488846
10	395.386719	1.295653484	25.35429999
12	502.616211	1.019235291	2.058793866
16	473.777344	1.081276229	8.017807304
20	496.820312	1.031125676	3.177485549
24	508.308594	1.007821206	0.8097922456
32	528.750977	0.9688571791	-3.318077088
64	649.761719	0.788418531	-27.26215792
128	875.375	0.5033077344	-99.46265358
256	1398.125977	0.3151239697	-218.1877244

GRAPH





Conclusion

We see the highest speedup of **1.295** at thread count of **10**, where the execution time is reduced by **116.89** seconds to generate and visualize the Henon Map in comparison to the serial code. Parallelization fraction of **25.35%** is observed at thread count of 10**.** Also, the performance reduces as we go from thread count of 32 and above.

References

Wikipedia | Henon Map

Wikipedia |Dynamical System

Wikipedia | Attractor

Dynamical Properties of the Hénon Mapping