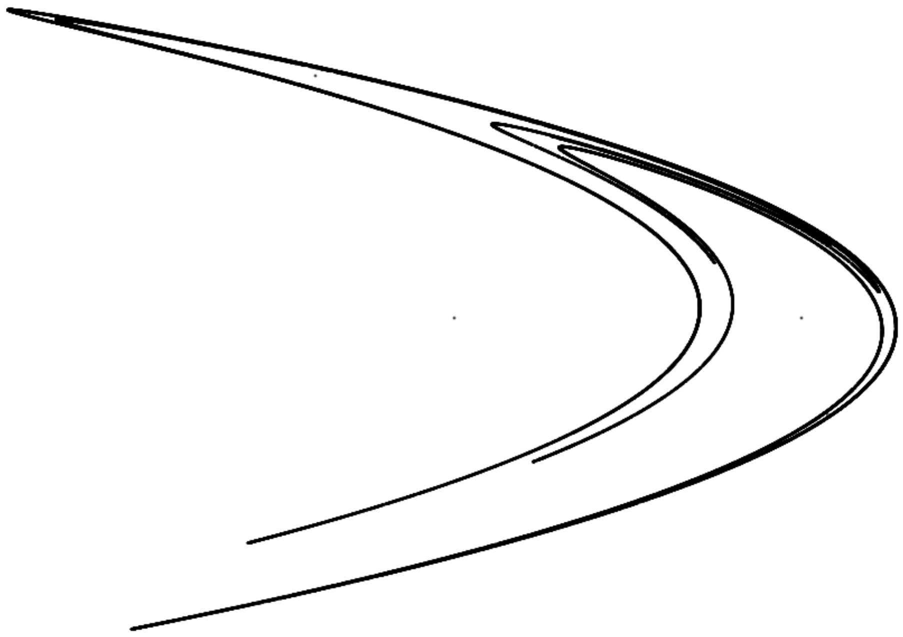


Henon Map

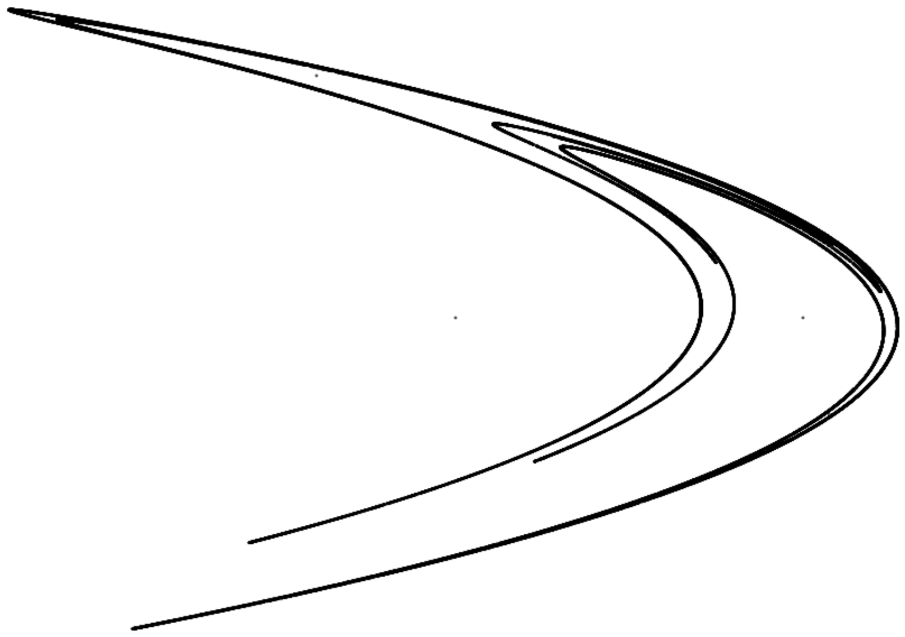
A THEORETICAL ANALYSIS

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.

$$\begin{cases} x_{n+1} = 1 - ax_n^2 + y_n \\ y_{n+1} = bx_n . \end{cases}$$

The Henon map may also be deconstructed into a one-dimensional map, defined similarly to the Fibonacci Sequence.

$$x_{n+1} = 1 - ax_n^2 + bx_{n-1}$$

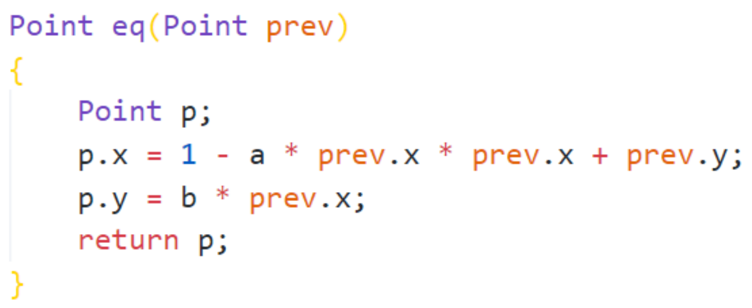
Code

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

Structure to hold the coordinates which are generated from the mathematical formulas.

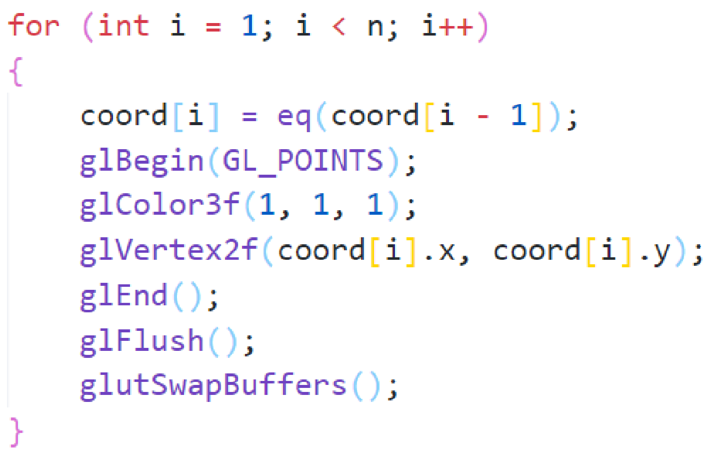
```
int start = 0, end = 10000000;
double a = 1.4;
double b = 0.3;
static GLfloat theta[] = {0.0, 0.0, 0.0};
GLint axis = 1;
Point initial = {1.0, 1.0};
```

Some constant which are used in the program, a and b are the constants that are used in the mathematical formula which is used to find the coordinates.



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

The function where I have used the mathematical equation to find the coordinates. So here we are using the previous points to fine the next points.



```
for (int i = 1; i < n; i++)
{
    coord[i] = eq(coord[i - 1]);
    glBegin(GL_POINTS);
    glColor3f(1, 1, 1);
    glVertex2f(coord[i].x, coord[i].y);
    glEnd();
    glFlush();
    glutSwapBuffers();
}
```

The loop which prints the points on the glut canvas (screen) which is implemented using OpenGL.

Entire 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 = 10000000;
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 p;
    p.x = 1 - a * prev.x * prev.x + prev.y;
    p.y = b * prev.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);
    for (int i = 1; i < n; i++)
    {
        coord[i] = eq(coord[i - 1]);
        glBegin(GL_POINTS);
        glColor3f(1, 1, 1);
        glVertex2f(coord[i].x, coord[i].y);
        glEnd();
        glFlush();
        glutSwapBuffers();
    }
    // For profiling
    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)
{
    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();
    glutLeaveMainLoop();
}
```

Profiling

In a performance engineering context performance profiling means to relate performance metric measurements to source code execution. Data sources are typically either operating system, execution environments or measurement facilities in the hardware.

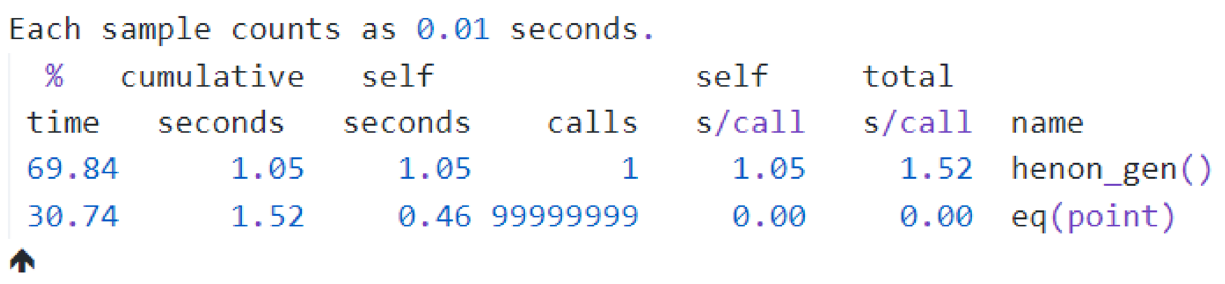
Tools used for profiling:

- Function-based profiling using gprof
- Line-based profiling using gcov
- Hardware profiling using likwid

GPROF

Gprof is used to get the frequency of each function calls, to determine the computation intensive function.

- Enable profiling during the compilation (-pg)
- Execute the binary to generate the profiling data
- The data is stored in a file "gmon.out" in the root directory.
- Use gprof -b <executable file>



```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 69.84      1.05     1.05        1     1.05     1.52  henon_gen()
 30.74      1.52     0.46 99999999     0.00     0.00  eq(point)
```

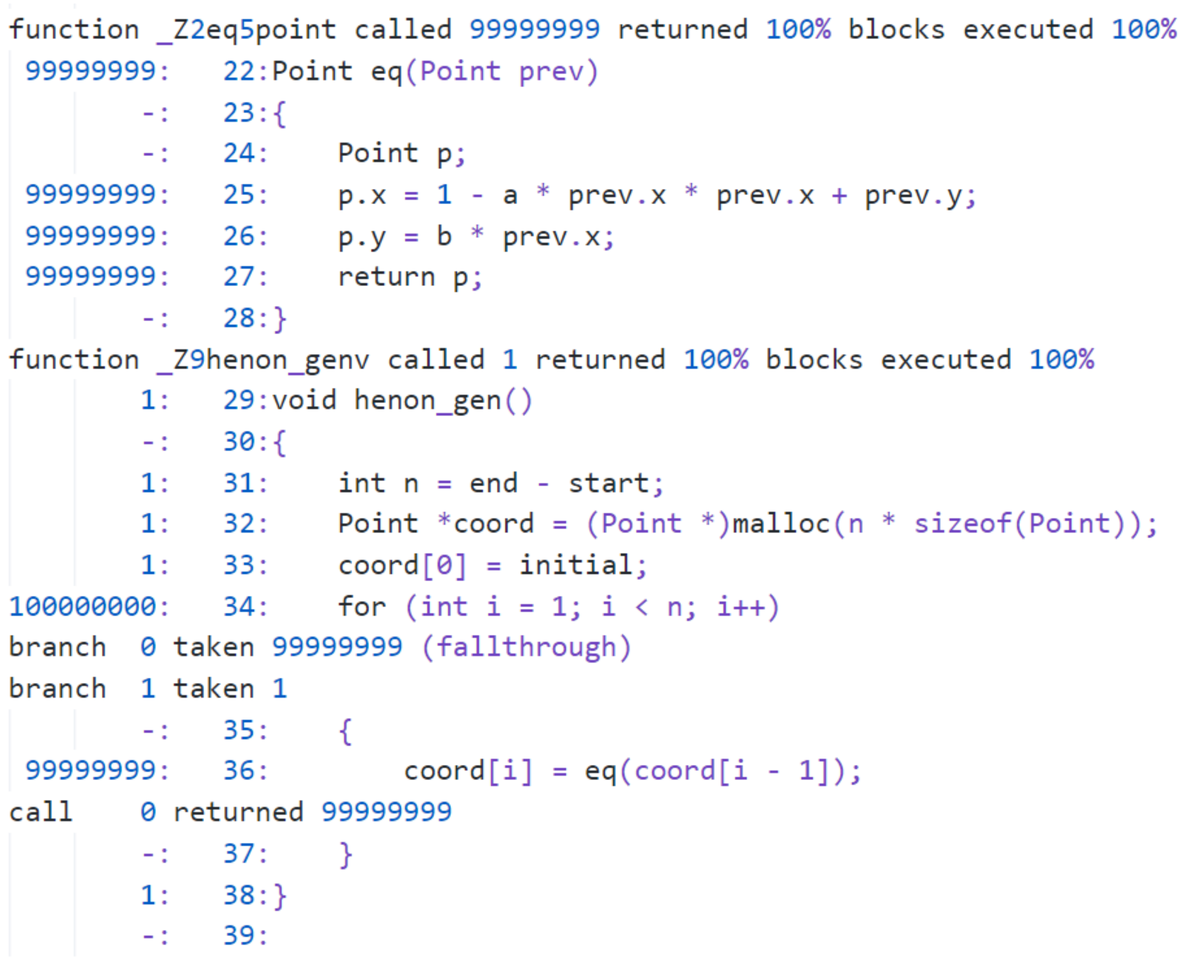
```
Call graph

granularity: each sample hit covers 2 byte(s) for 0.66% of 1.52 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    1.52                 main [1]
                1.05    0.46       1/1           henon_gen() [2]
-----------------------------------------------
                1.05    0.46       1/1           main [1]
[2]    100.0    1.05    0.46       1         henon_gen() [2]
                0.46    0.00 99999999/99999999     eq(point) [3]
-----------------------------------------------
                0.46    0.00 99999999/99999999     henon_gen() [2]
[3]     30.6    0.46    0.00 99999999         eq(point) [3]
-----------------------------------------------

Index by function name

   [3] eq(point)               [2] henon_gen()
```

It's clear from above call graph, the eq function which generates the next point using the previous point is called most frequently.

GCOV

Gcov is used to get the frequency of each line, to determine the computation intensive section.

- Enable profiling during the compilation (-fprofile-arcs -ftest-coverage)
- Execute the binary to generate the profiling data
- Run gcov to generate the coverage data (gcov -b -c <file-name>.c)
- The data is stored in a file "<file-name>.c.gcov" in the root directory.

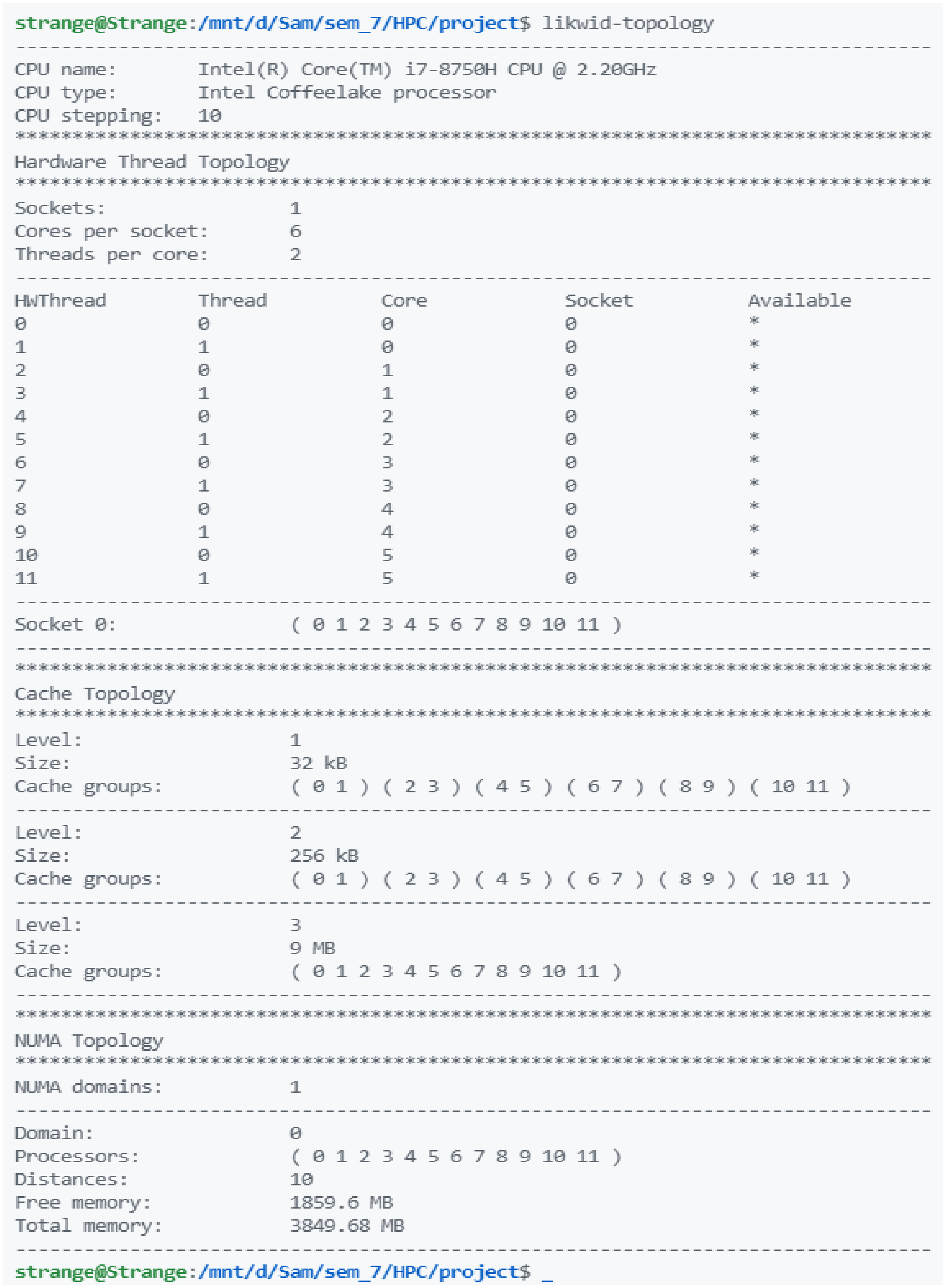


```
function _Z2eq5point called 99999999 returned 100% blocks executed 100%
 99999999:   22:Point eq(Point prev)
        -:   23:{
        -:   24:    Point p;
 99999999:   25:    p.x = 1 - a * prev.x * prev.x + prev.y;
 99999999:   26:    p.y = b * prev.x;
 99999999:   27:    return p;
        -:   28:}
function _Z9henon_genv called 1 returned 100% blocks executed 100%
        1:   29:void henon_gen()
        -:   30:{
        1:   31:    int n = end - start;
        1:   32:    Point *coord = (Point *)malloc(n * sizeof(Point));
        1:   33:    coord[0] = initial;
100000000:   34:    for (int i = 1; i < n; i++)
branch  0 taken 99999999 (fallthrough)
branch  1 taken 1
        -:   35:    {
 99999999:   36:        coord[i] = eq(coord[i - 1]);
call    0 returned 99999999
        -:   37:    }
        1:   38:}
        -:   39:
```

From the above outputs, it is observed that the loop that generates the coordinates are the lines which are most frequently run.

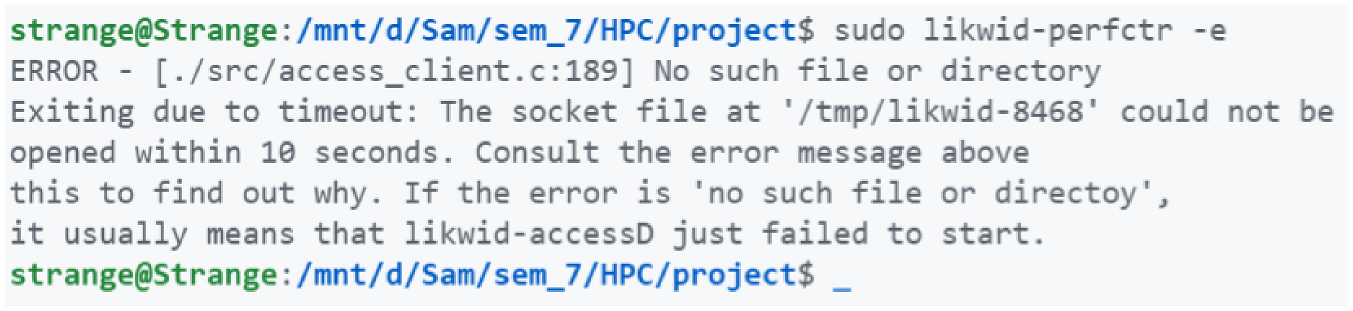
LIKWID

likwid-topology output



```
strange@Strange:/mnt/d/Sam/sem_7/HPC/project$ likwid-topology
--------------------------------------------------------------------------------
CPU name:       Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
CPU type:       Intel Coffeelake processor
CPU stepping:   10
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                1
Cores per socket:       6
Threads per core:       2
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               1               0               0               *
2               0               1               0               *
3               1               1               0               *
4               0               2               0               *
5               1               2               0               *
6               0               3               0               *
7               1               3               0               *
8               0               4               0               *
9               1               4               0               *
10              0               5               0               *
11              1               5               0               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 1 2 3 4 5 6 7 8 9 10 11 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 ) ( 8 9 ) ( 10 11 )
--------------------------------------------------------------------------------
Level:                  2
Size:                   256 kB
Cache groups:           ( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 ) ( 8 9 ) ( 10 11 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   9 MB
Cache groups:           ( 0 1 2 3 4 5 6 7 8 9 10 11 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           1
--------------------------------------------------------------------------------
Domain:                 0
Processors:             ( 0 1 2 3 4 5 6 7 8 9 10 11 )
Distances:              10
Free memory:            1859.6 MB
Total memory:           3849.68 MB
--------------------------------------------------------------------------------
strange@Strange:/mnt/d/Sam/sem_7/HPC/project$ _
```

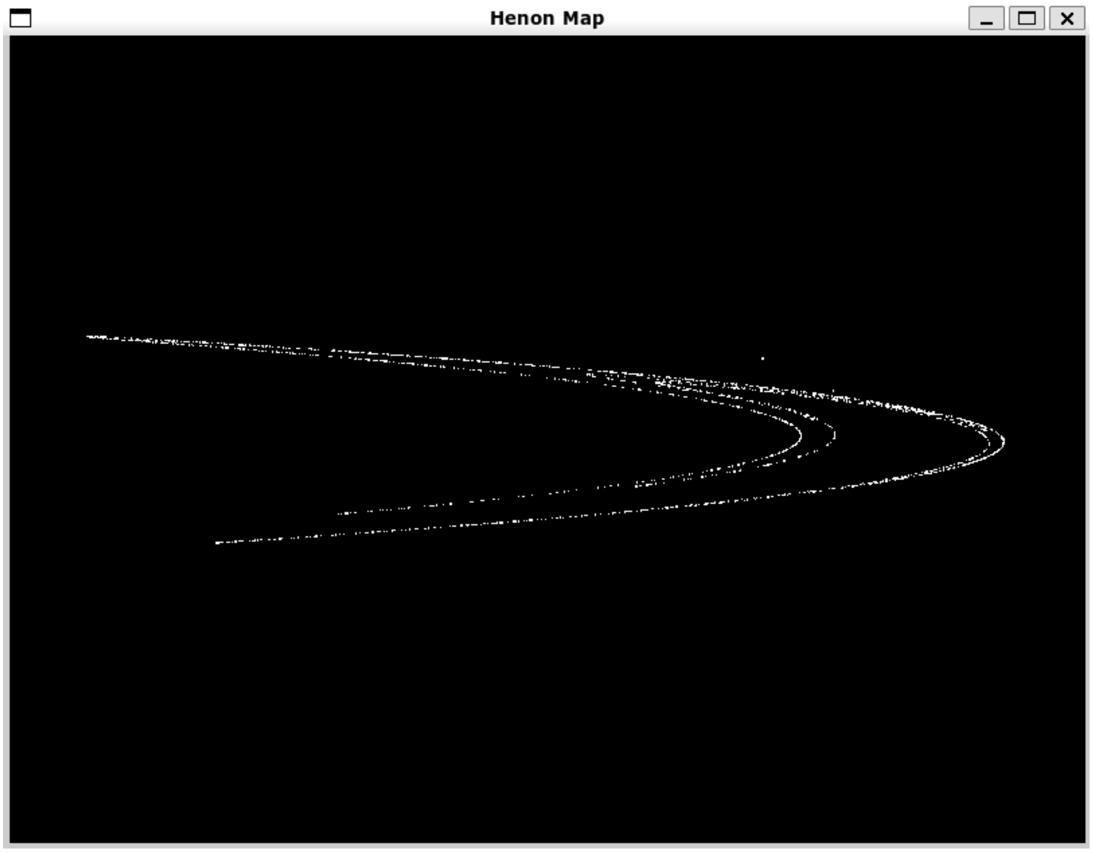
likwid-perfctr -e output:



```
strange@Strange:/mnt/d/Sam/sem_7/HPC/project$ sudo likwid-perfctr -e
ERROR - [./src/access_client.c:189] No such file or directory
Exiting due to timeout: The socket file at '/tmp/likwid-8468' could not be
opened within 10 seconds. Consult the error message above
this to find out why. If the error is 'no such file or directoy',
it usually means that likwid-accessD just failed to start.
strange@Strange:/mnt/d/Sam/sem_7/HPC/project$ _
```

Getting an error while using likwid-perfctr.

Output



Real-time implementation of Henon Map using OpenGL

Conclusion

On application of the mentioned profiling tools, namely, GPROF, GCOV, LIKWID, the critical section of the program is determined.

GPROF: It's clear from above call graph, the eq function which generates the next point using the previous point is called most frequently.

GCOV: It is observed that the loop that generates the coordinates are the lines which are most frequently run.

LIKWID: No inference was drawn from these outputs as the code is not yet parallelized and likwid-perfctr gives an error when using the command.