PROFILING ON

LORENZ ATTRACTOR

High Performance Computing Project Report

By,

SAI KAUSHIK SUDHAKARAN

CED18I044

Department of Computer Science and Engineering,

Indian Institute of Information Technology Design and Manufacturing

Kancheepuram, Chennai

August 2021

TABLE OF CONTENT

INTRODUCTION

The Lorenz System is a system of differential equations, noted to have chaotic solutions for given parameters and the initial conditions, first studied by Edward Lorenz in 1962 [1].

Lorenz attractor is the set of chaotic solutions of the Lorenz System. It is a part of an area of study, Chaos Theory.

The shape of the Lorenz attractor, when plotted graphically, represents that of a butterfly. The butterfly effect is a real-world implication of the Lorenz attractor.

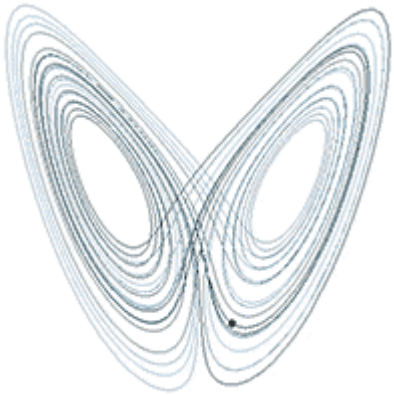


Figure 1: *Lorenz Attractor Animation*

This project is designed to implement parallel programs for the simulation of objects following Lorenz attractor satisfying the Lorenz system.

EQUATION

In 1963, with the help of Ellen Fetter, Lorenz simplified the above mathematical model to a system of differential equations, now known as the Lorenz equations.

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = x(\rho - z) - y$$

$$\frac{dz}{dt} = xy - \beta z$$

Here, x, y, z are the coordinates on the 3-dimensional cartesian plane

σ = 10.0

ρ = 28.0

β = 8.0 / 3.0

dt = 0.01

CODE

```
typedef struct point
{
    double x;
    double y;
    double z;
    float t;
} Point;
```

Structure to hold the coordinates of the generated point and the time it is being generated.

```
double sigma = 10.0;
double rho = 28.0;
double beta = 8.0 / 3.0;

Point initial = {1.0, 1.0, 1.0, 0.0};
double dt = 0.01;

int start = 0;
int end = 1000000;
```

Some useful constants used throughout the execution.

Sigma, rho and beta are the constants in the Lorenz Equation. initial is the starting point of the equation. start and end hold the number of iterations.

```
Point differential(Point curr)
{
    Point diff;
    diff.x = (sigma * (curr.y - curr.x)) * dt + curr.x;
    diff.y = (curr.x * (rho - curr.z) - curr.y) * dt + curr.y;
    diff.z = (curr.x * curr.y - beta * curr.z) * dt + curr.z;
    diff.t = curr.t + dt;
    return diff;
}
```

A function to compute the next point using the current point.

Multiply dt to dx / dt, dy / dt, dz / dt, to get the difference in the points and add this difference to the current point to generate the next point.

This function takes a current point structure and returns the next current point structure.

```
for (int i = start; i < end; i++)
{
    coord = differential(coord);
    glBegin(GL_POINTS);
    glColor3f(1, 1, 1);
    glVertex3f(coord.x, coord.y, coord.z);
    glEnd();
    glFlush();
    glutSwapBuffers();
}
```

This section of the code plots the generated points on the glut canvas using OpenGL.

The points and color are added by glColor3f and glVertex3f. glFlush and glutSwapBuffers flushes the display buffer causing it to redisplay the updated graph for a real-time visualization.

The entire code:

```
/*
gcc -o lorenz lorenz.c -lGL -lGLU -lglut -lGLEW -lm
./lorenz
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <GL/freeglut.h>
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/gl.h>

typedef struct point
{
    double x;
    double y;
    double z;
    float t;
} Point;

double sigma = 10.0;
double rho = 28.0;
double beta = 8.0 / 3.0;

Point initial = {1.0, 1.0, 1.0, 0.0};
double dt = 0.01;
```

```
int start = 0;
int end = 100000;

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

Point differential(Point curr)
{
    Point diff;
    diff.x = (sigma * (curr.y - curr.x)) * dt + curr.x;
    diff.y = (curr.x * (rho - curr.z) - curr.y) * dt + curr.y;
    diff.z = (curr.x * curr.y - beta * curr.z) * dt + curr.z;
    diff.t = curr.t + dt;
    return diff;
}

void lorenzGenerator()
{
    Point coord = 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(1.0);
    for (int i = start; i < end; i++)
    {
        coord = differential(coord);
        glBegin(GL_POINTS);
        glColor3f(0, 0, 0);
        glVertex3f(coord.x, coord.y, coord.z);
        glEnd();
        glFlush();
        glutSwapBuffers();
    }
    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();
    glClearColor(255, 255, 255, 255);
    glOrtho(-50.0, 50.0, -50.0, 50.0, -50.0, 50.0);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(1280, 720);

    glutCreateWindow("Lorenz");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(lorenzGenerator);
    glutIdleFunc(spinCube);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

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>

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.70     46.21    46.21                             lorenzGenerator
  0.00     46.21     0.00   100000     0.00     0.00  differential

Call graph:

granularity: each sample hit covers 2 byte(s) for 0.02% of 46.21 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0   46.21    0.00                 lorenzGenerator [1]
                0.00    0.00  100000/100000      differential [2]
-----------------------------------------------
                0.00    0.00  100000/100000      lorenzGenerator [1]
[2]      0.0    0.00    0.00  100000         differential [2]
-----------------------------------------------
Index by function name

   [2] differential            [1] lorenzGenerator
```

As shown in the outputs, the most frequently called function is the differential function, that generates the next point of the Lorenz equation.

GCOV

Gprof 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.

```
100000:   33:Point differential(Point curr)
     -:   34:{
     -:   35:    Point diff;
100000:   36:    diff.x = (sigma * (curr.y - curr.x)) * dt + curr.x;
100000:   37:    diff.y = (curr.x * (rho - curr.z) - curr.y) * dt + curr.y;
100000:   38:    diff.z = (curr.x * curr.y - beta * curr.z) * dt + curr.z;
100000:   39:    diff.t = curr.t + dt;
100000:   40:    return diff;
     -:   41:}
```

```
  100001:   52:    for (int i = start; i < end; i++)
branch  0 taken 100000
branch  1 taken 1 (fallthrough)
        -:   53:    {
   100000:   54:        printf("%d\n", i);
call    0 returned 100000
   100000:   55:        coord = differential(coord);
call    0 returned 100000
   100000:   56:        glBegin(GL_POINTS);
call    0 returned 100000
   100000:   57:        glColor3f(1, 1, 1);
call    0 returned 100000
   100000:   58:        glVertex3f(coord.x, coord.y, coord.z);
call    0 returned 100000
   100000:   59:        glEnd();
call    0 returned 100000
   100000:   60:        glFlush();
call    0 returned 100000
   100000:   61:        glutSwapBuffers();
call    0 returned 100000
        -:   62:    }
        1:   63:    glutLeaveMainLoop();
call    0 returned 1
        1:   64:}
```

As we can see in the above outputs, the most frequent lines are in the loop that generates the coordinates and plots it and also the lines in the differential function.

LIKWID

likwid-topology output

```
thegamingbot@LAPTOP-7ONLVAJ1:/mnt/d/College/Sem-7/HPC Project/Lorenz$ likwid-topology
--------------------------------------------------------------------------------
CPU name:       AMD Ryzen 7 5800H with Radeon Graphics
CPU type:       nil
CPU stepping:   0
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                1
Cores per socket:       8
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               *
12              0               6               0               *
13              1               6               0               *
14              0               7               0               *
15              1               7               0               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 ) ( 8 9 ) ( 10 11 ) ( 12 13 ) ( 14 15 )
--------------------------------------------------------------------------------
Level:                  2
Size:                   512 kB
Cache groups:           ( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 ) ( 8 9 ) ( 10 11 ) ( 12 13 ) ( 14 15 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   16 MB
Cache groups:           ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           1
--------------------------------------------------------------------------------
Domain:                 0
Processors:             ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 )
Distances:              10
Free memory:            5085.13 MB
Total memory:           7880.87 MB
--------------------------------------------------------------------------------
```

```
********************************************************************************
Graphical Topology
********************************************************************************
Socket 0:
+-------------------------------------------------------------------------------------------------+
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  0 1   | |  2 3   | |  4 5   | |  6 7   | |  8 9   | | 10 11  | | 12 13  | | 14 15  | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| |  32 kB | |  32 kB | |  32 kB | |  32 kB | |  32 kB | |  32 kB | |  32 kB | |  32 kB | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| | 512 kB | | 512 kB | | 512 kB | | 512 kB | | 512 kB | | 512 kB | | 512 kB | | 512 kB | |
| +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ +--------+ |
| +---------------------------------------------------------------------------------------+ |
| |                                        16 MB                                          | |
| +---------------------------------------------------------------------------------------+ |
+-------------------------------------------------------------------------------------------------+
```

likwid-pin -c output: As the program is not currently parallelized, the program just runs in the core 0.

```
thegamingbot@LAPTOP-7ONLVAJ1:/mnt/d/College/Sem-7/HPC Project/Lorenz$ likwid-pin -c 0,1,2,3 ./main
Sleeping longer as likwid_sleep() called without prior initialization
NVD3D10: CPU cyclestats are disabled on client virtualization
[pthread wrapper]
[pthread wrapper] MAIN -> 0
[pthread wrapper] PIN_MASK: 0->1  1->2  2->3
[pthread wrapper] SKIP MASK: 0x0
        threadid 139888905840384 -> core 1 - OK
        threadid 139888897447680 -> core 2 - OK
        threadid 139888889054976 -> core 3 - OK
Roundrobin placement triggered
        threadid 139888880662272 -> core 0 - OK
NVD3D10: CPU cyclestats are disabled on client virtualization
```

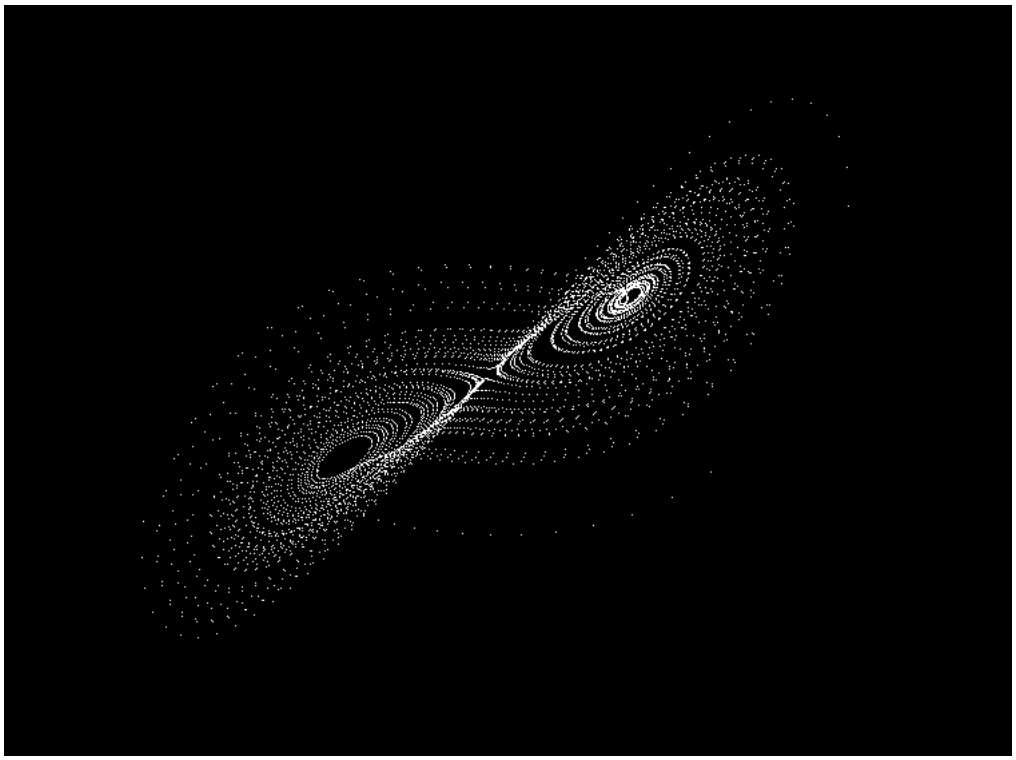
likwid-perfctr -e output:

```
thegamingbot@LAPTOP-7ONLVAJ1:/mnt/d/College/Sem-7/HPC Project/Lorenz$ sudo likwid-perfctr -e
ERROR - [./src/perfmon.c:perfmon_init_maps:1282] Unsupported Processor
ERROR - [./src/perfmon.c:perfmon_check_counter_map:720] Counter and event maps not initialized.
This architecture has 0 counters.
Counter tags(name, type<, options>):



This architecture has 0 events.
Event tags (tag, id, umask, counters<, options>):
```

On checking the list of supported architectures using “likwid-perfctr -i”, the current processor “AMD Ryzen 7 5800H” is not listed.

```
Supported AMD processors:
        AMD Opteron single core 130nm processor
        AMD Opteron Dual Core Rev E 90nm processor
        AMD Opteron Dual Core Rev F 90nm processor
        AMD K10 (Barcelona) processor
        AMD K10 (Shanghai) processor
        AMD K10 (Istanbul) processor
        AMD K10 (Magny Cours) processor
        AMD Interlagos processor
        AMD Family 16 model - Kabini processor
        AMD K17 (Zen) architecture
        AMD K17 (Zen2) architecture
```

OUTPUT



A real-time visualization of the Lorenz Attractor

CONCLUSION

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

GPROF: The most commonly called function is the deferential function that computes the location of the next point with respect to the current point.

GCOV: The most commonly run lines are the ones within the loop, that generate and plots the coordinates and the lines in the differential function.

LIKWID: No inference was drawn from these outputs as the code is not yet parallelized and likwid-perfctr does not support the student's processor.

REFERENCES

[1] Wikipedia, “Lorenz system - Wikipedia,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Lorenz_system.

[2] Wikipedia, “Chaos theory - Wikipedia,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Chaos_theory.

[3] Wikipedia, “Butterfly effect - Wikipedia,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Butterfly_effect.

[4] Wikipedia, “Atmospheric convection - Wikipedia,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Atmospheric_convection.

[5] E. Lorenz, “The statistical prediction of solutions of dynamic equations.,” Meteorological Society of Japan, Tokyo, 1962.

[6] J. Burkardt, “LORENZ_ODE - The Lorenz System,” Florida State University, [Online]. Available: https://people.sc.fsu.edu/~jburkardt/c_src/lorenz_ode/lorenz_ode.html.