Assignment No 4
High Performance Computing
Prof. *Alejandro Cárdenas*
Facultad de Matemáticas e Ingenierías
Fundación Universitaria Konrad Lorenz

Ricardo Cano
Code: *614141007*
Programa de Matemáticas
Date: 7 de marzo de 2019

Solution

1. Using kernprof, line_profiler and memory Profiler to analyse the given code as it is (V1.ipnyb). What is that code doing and where? Where is it spending the majority of the time? How much memory is being used and where is the most used one?

Solution:
This code is divided into two functions, the first function (HenonMap) receives the parameters and then enters them in the returning function here this function spends the 100 % of the time (See figure 1).

The second function (myfunction) is the main function, where the whole process of the program is performed as such, having made use of the decorator line_profiler I found that the program spends 53 % of the time on the function that is inside the while cycle, and that spends 23 % of the time on the conditional that while requires for the program to continue performing the validation, then spends 10 % of the time on the other validation that the while has after the break, finally the counter of the variable aux spends 9,7 % of the time. In these fourth items already mentioned, the program spends a greater percentage of time (See figure 2) *I didn't show myfunction part 1 'cause does not show something relevant for this estudy.*

The time running of **V1 was about** 782,497*s* in minuts it is approx. 13 min. I tried to make *memory profile and the computer spent more than* 20 *minutes, so I made the simulation for* 10 *iterations and the program uses more memory is when it is plotting the Henon Map (Spends 160.859 MiB (See figure 3)).*

```
Total time: 149.375 s
File: V1.py
Function: HenonMap at line 10

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    10                                           @profile
    11                                           def HenonMap(x,y,a,b,alpha,beta):
    12  55466629  149374519.0      2.7    100.0      return a- alpha*x**2+b*y, beta*x
```

Figura 1: HenonMap

```
52        1         83.0     83.0      0.0         qgrid=np.linspace(-4,0,npoints)
53        1         26.0     26.0      0.0         values=np.zeros([npoints,npoints])
54      601        869.0      1.4      0.0         for i in range(npoints):
55   360600     547455.0      1.5      0.1                 for j in range(npoints):
56   360000     747262.0      2.1      0.1                         xtemp = pgrid[i]
57   360000     599027.0      1.7      0.1                         ytemp = qgrid[j]
58   360000     474356.0      1.3      0.1                         aux=0
59 55825597  186222343.0      3.3     23.8                         while (xtemp**2+ytemp**2)<R:
60 55465629  419901501.0      7.6     53.7                                 xtemp, ytemp = HenonMap(xtemp,ytemp,a,b,alpha,beta)
61 55465629   81431035.0      1.5     10.4                                 if aux>1000:
62       32         38.0      1.2      0.0                                         break
63 55465597   76025341.0      1.4      9.7                                 aux+=1
64   360000     1093565.0      3.0      0.1                         values[i,j]=aux
65
66        1   1258968.0 1258968.0      0.2         plt.imshow(values, interpolation='nearest', cmap=cm.hot)
67        1   1845217.0 1845217.0      0.2         plt.savefig("HenonMap.png",dpi=300)
68        1   4547616.0 4547616.0      0.6         show()
```

Figura 2: myfunction part 2

```
46
47  140.984 MiB    0.000 MiB        R=100
48  140.984 MiB    0.000 MiB        p=4
49  140.984 MiB    0.000 MiB        q=4
50  140.984 MiB    0.000 MiB        npoints=10
51  140.984 MiB    0.000 MiB        pgrid=np.linspace(0,4,npoints)
52  140.984 MiB    0.000 MiB        qgrid=np.linspace(-4,0,npoints)
53  140.988 MiB    0.004 MiB        values=np.zeros([npoints,npoints])
54  140.988 MiB    0.000 MiB        for i in range(npoints):
55  140.988 MiB    0.000 MiB                for j in range(npoints):
56  140.988 MiB    0.000 MiB                        xtemp = pgrid[i]
57  140.988 MiB    0.000 MiB                        ytemp = qgrid[j]
58  140.988 MiB    0.000 MiB                        aux=0
59  140.988 MiB    0.000 MiB                        while (xtemp**2+ytemp**2)<R:
60  140.988 MiB  140.988 MiB                                xtemp, ytemp = HenonMap(xtemp,ytemp,a,b,alpha,beta)
61  140.988 MiB    0.000 MiB                                if aux>1000:
62                                                                  break
63  140.988 MiB    0.000 MiB                                aux+=1
64  140.988 MiB    0.000 MiB                        values[i,j]=aux
65
66  141.246 MiB    0.258 MiB        plt.imshow(values, interpolation='nearest', cmap=cm.hot)
67  156.586 MiB   15.340 MiB        plt.savefig("HenonMap.png",dpi=300)
68  160.859 MiB    4.273 MiB        show()
```

Figura 3: Memory Profile

2. Write a version if that code that does not use numpy. (Call it V2.py)

> **Solution:**
> You can watch the solution adjoin in this set file **(See V2.py)**

3. Optimise the code as much as possible using at least (Call it V3.py):
   - Numba (Use the decorator @jit with at least one argument)
   - Broadcasting

> **Solution:**
> You can watch the solution adjoin in this set file **(See V3.py)**

4. Compare the three version of the codes and comment on your results.

Solution:
Of the three versions the one that ran faster was $V3.py$, this is because I am using the $@jit$ decorator in $HenonMap$, so it ran faster the cycles in $Func1$ and $Func2$.