Cálculo do PI proposto por Monte Carlo

Para a resolução do problema utilizando programação paralela, foi utilizado o pacote *multiprocessing*, disponibilizado nativamente na linguagem Python. O pacote *multiprocessing* disponibiliza o objeto Pool, em que é possível paralelizar a execução de uma função em vários valores de entrada, distribuindo os dados de entrada entre os processos (paralelismo de dados).

O método de aproximação do valor do PI proposto por Monte Carlo utiliza um grande número de entradas aleatórias gerada uniformemente entre os valores de 0 e 1. A cada execução é gerado um valor aleatório para x e y entre 0 e 1. A partir dos valores de x e y é possível calcular a distância entre esses dois valores utilizando o teorema de Pitágoras, uma vez que a distância da origem de uma circunferência e a sua borda é sempre igual ao raio, que nesse caso é igual a 1.

Dessa forma, quando a distância calculada é maior que o valor do raio, esse ponto estará fora da área do quadrante da circunferência, caso seja menor que o valor do raio, então, o *plot* desse ponto será dentro da circunferência. Quando gerado um grande número de valores aleatórios é possível obter um valor de aproximação do π utilizando a fórmula $\pi = 4 \times \frac{N_{interno}}{N_{total}}$, onde $N_{interno}$ é a quantidade de pontos em que a distância entre os valores gerados para x e y é menor que 1 e N_{total} é a quantidade total de pontos gerados.

Para utilizar o modelo de dados múltiplos de programa único, como também o balanceamento de carga, foi definido 8 processos *worker*, e que serão gerados 20.000 pontos aleatórios, dessa forma, cada *worker* executou a função que gera valores aleatórios e calcula a distância 2.500 vezes.

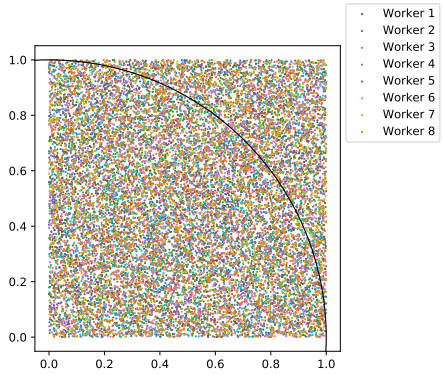

Figura 1: *Plots* para cada *worker*

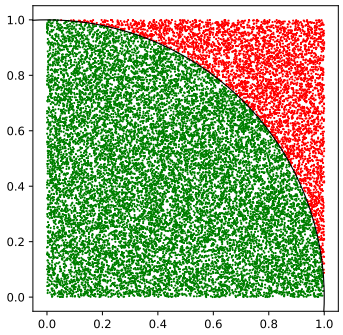

Figura 2: *Plot* total dos pontos

As figura acima representam os *plots* dos pontos, sendo que a Figura 1 contêm uma cor para cada ponto gerado por um *worker*, enquanto a Figura 2 contêm os pontos dentro e fora da circunferência.

O valor de aproximação do π para 8 processos *workers* e 20.000 pontos foi 3.14959.

Abaixo é apresentado a impelementação utilizando Python e o pacote *multiprocessing*, como também a geração dos *plots*. Entre a linha 13 e 36, é realizado o balanceamento de carga e executado a função principal que retorna a quantidade de pontos gerados dentro da circunferência, como também vetores contendo as coordenadas dos pontos para os *plots*. Na linha 36 é realizado o *map* com a função e os parâmetros da função. O retorno do *map* é um vetor de tuplas, em que cada tupla contêm o retorno da função *pi*. O restante do código contêm as funções de *plots*, onde na linha 50 é populado um vetor com a quantidade de pontos gerados de um processo *worker* dentro de uma circunferência. Esse vetor é utilizado para calcular a aproximação d π na linha 87.

```
1  #!/usr/bin/python
2  # -*- coding: UTF-8 -*-
3
4  import matplotlib
5  matplotlib.use('Agg')
6
7  import matplotlib.pyplot as plt
8
9  import random
10 import multiprocessing
11 from multiprocessing import Pool
12
13 def pi(n):
14     contador = 0
15     x_dentro = []
16     y_dentro = []
17     x_fora = []
18     y_fora = []
19     for i in range(n):
20         x = random.uniform(0, 1)
21         y = random.uniform(0, 1)
22         if (x ** 2 + y ** 2) < 1.0:
23             contador += 1
24             x_dentro.append(x)
25             y_dentro.append(y)
26         else:
27             x_fora.append(x)
28             y_fora.append(y)
```

```
29
30      return contador, x_dentro, y_dentro, x_fora, y_fora
31
32  workers = 8
33  totalPontos = 20000
34  divisaoWorker = [totalPontos/workers for i in range(workers)]
35  pool = Pool(processes = workers)
36  resultado = pool.map(pi, divisaoWorker)
37
38  dict = {
39      0: { 'cor': 'tab:blue', 'nome': 'Worker 1' },
40      1: { 'cor': 'tab:green', 'nome': 'Worker 2' },
41      2: { 'cor': 'tab:cyan', 'nome': 'Worker 3' },
42      3: { 'cor': 'tab:purple', 'nome': 'Worker 4' },
43      4: { 'cor': 'tab:brown', 'nome': 'Worker 5' },
44      5: { 'cor': 'tab:pink', 'nome': 'Worker 6' },
45      6: { 'cor': 'tab:olive', 'nome': 'Worker 7' },
46      7: { 'cor': 'tab:orange', 'nome': 'Worker 8' },
47  }
48
49  fig, ax = plt.subplots()
50  circulo = plt.Circle((0, 0), 1, fill=False)
51  ax.set_aspect(1)
52  ax.add_artist(circulo)
53
54  contador = []
55  for idx, tupla in enumerate(resultado):
56      contador.append(tupla[0])
57      ax.set_aspect('equal')
58      legenda = dict.get(idx)['nome']
59
60      ax.scatter(tupla[1], tupla[2],
61          color='{}'.format(dict.get(idx)['cor']),
62          marker='s',
63          s=0.5,
64          label=legenda)
65
66      ax.scatter(tupla[3], tupla[4],
67          color='{}'.format(dict.get(idx)['cor']),
68          marker='s',
69          s=0.5)
70
71  fig.legend()
72  fig.savefig('Workers.pdf')
73
74
```

```
75 fig, ax = plt.subplots()
76 circulo = plt.Circle((0, 0), 1, fill=False)
77 ax.set_aspect(1)
78 ax.add_artist(circulo)
79
80 for idx, tupla in enumerate(resultado):
81     ax.set_aspect('equal')
82     ax.scatter(tupla[1], tupla[2], color='green', marker='s', s=0.5)
83     ax.scatter(tupla[3], tupla[4], color='red', marker='s', s=0.5)
84
85 fig.savefig('Total.pdf')
86
87 print('~ PI: {:.20f}'.format(sum(contador)/float(totalPontos)*4))
```