Camilla Pozer de Matos
Programação Paralela e Multi-Core: Trabalho Final

Problema 2: **Cálculo do PI**

Para encontrar um valor aproximado de PI, existe uma função chamada Monte Carlo, que consiste em gerar números aleatórios em um quadrado sobreposto por uma circunferência. Com todos esses pontos, é possível fazer a contagem de quantos estão dentro da circunferência e aplicar na fórmula abaixo:

$$PI = 4 * NC/NT$$

Dado que NC refere-se à quantidade de pontos dentro da circunferência e NT o número total de pontos, é possível chegar no valor aproximado de PI.
Para executar essa função, foi utilizado paralelismo em Python, através da biblioteca multiprocessing, que suporta processos simultâneos localmente e remotamente. Desta biblioteca, o paralelismo foi feito através do Manager e do Process, que controla a criação dos processos paralelamente.

Em relação ao código, ele foi dividido em 4 partes, sendo 3 funções e a implementação principal que faz chamadas a elas. Na parte principal são instanciadas as variáveis de quantidade de workers e quantidades de pontos por worker, além da execução de cada um paralelamente, imprimindo os resultados obtidos no final.

A primeira função chamada é a “worker”, que fica responsável por chamar a criação dos pontos, imprimir informações sobre cada task e retornar os resultados.

```
def worker(worker_id, return_dict, total_points ):
    pi = createPoints(total_points, worker_id);
    print("Task " + str(worker_id) + ": " + str(pi[0]) + " pontos dentro do circulo")
    return_dict[worker_id] = pi
```

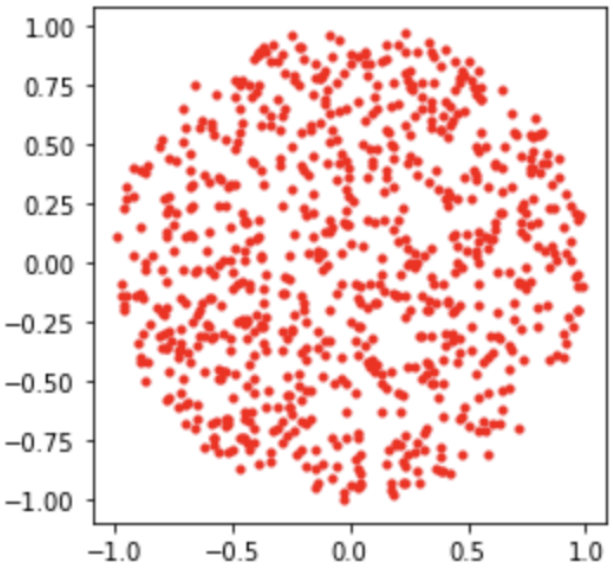
A segunda função, que é chamada pelo “worker”, é a criação dos pontos. Nessa função, são gerados pontos aleatórios através da biblioteca random. Após isso, é verificado se a posição destes pontos em um plano ficaria dentro ou fora da circunferência, que caso seja dentro é somado 1 ao número total de pontos que ficam dentro. Ainda, é adicionado a um array a posição X e Y do ponto para uma posterior visualização em gráficos.

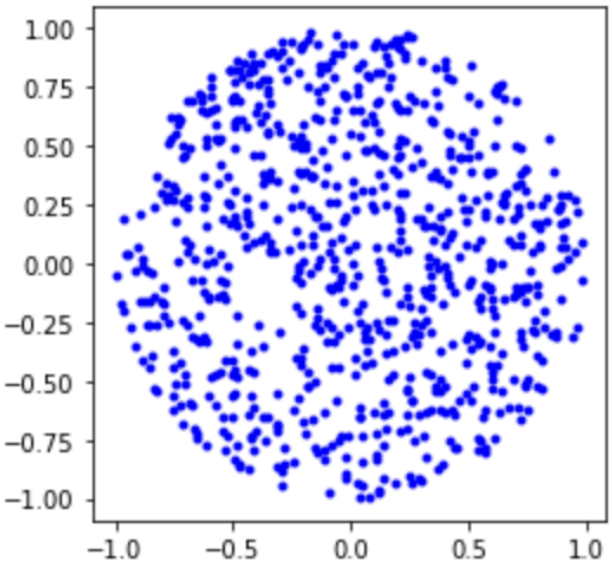
```
def createPoints(total_points, worker_id):
  num_hits = 0
  inside = []
  np.random.seed(worker_id)

  for i in range(total_points):
    x = np.random.uniform(-1, 1)
    y = np.random.uniform(-1, 1)

    if np.sqrt(x**2 + y**2) < 1:
        inside.append((x, y))
        num_hits += 1

  return [num_hits, total_points, inside]
```

Por fim, a última função serve apenas para criar o círculo de pontos graficamente. Para isso, são utilizados apenas os pontos que foram considerados dentro da circunferência, formando um círculo perfeito conforme a quantidade de pontos.

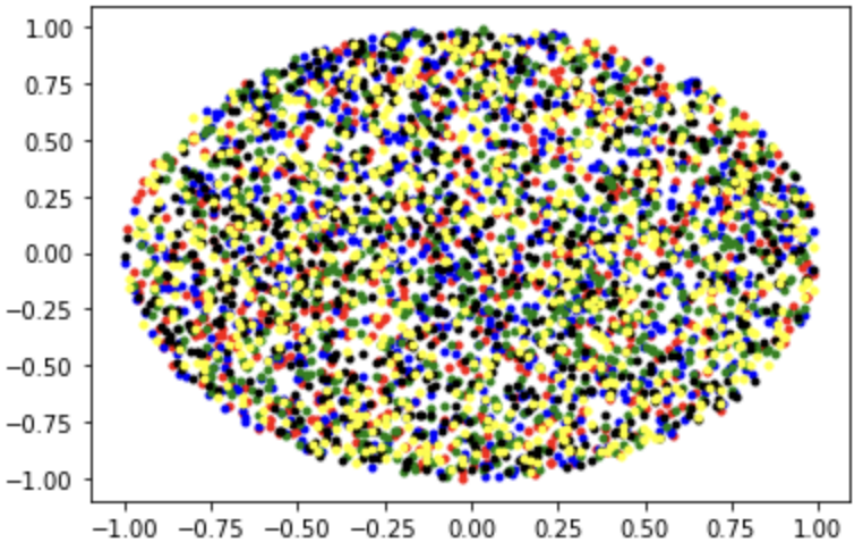
Para realizar os testes e obter os resultados, são utilizados 5 workers, identificados pelas cores vermelho, azul, verde, preto e amarelo. Em relação a quantidade de pontos, o ideal é possuir a maior quantidade possível, visto que o valor estimado chegará mais próximo do valor real do PI. Porém, para uma melhor visualização, foi iniciado a execução com a criação de 1000 pontos por worker.

Nas duas figuras abaixo, pode-se ver o círculo formado pelos pontos do worker azul e vermelho, sendo que os outros workers ficaram bem parecidos:



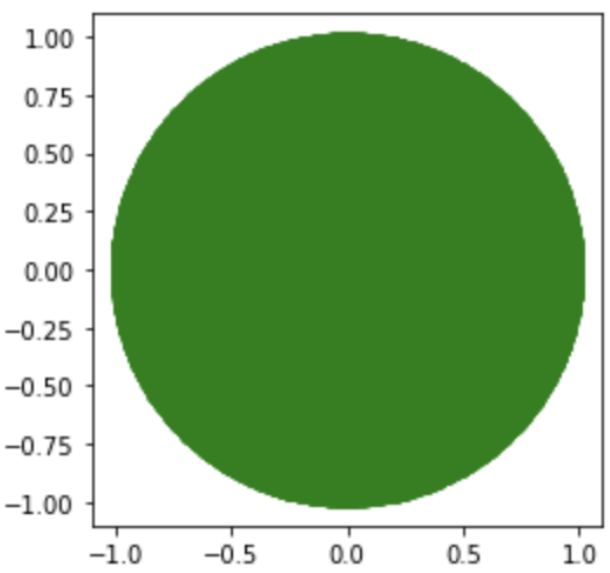


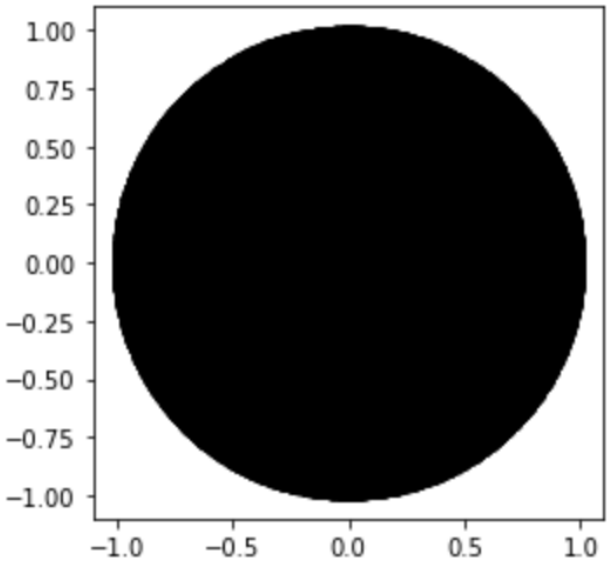
Para finalizar, a impressão de todos os pontos gerados, formando um círculo completamente visível:



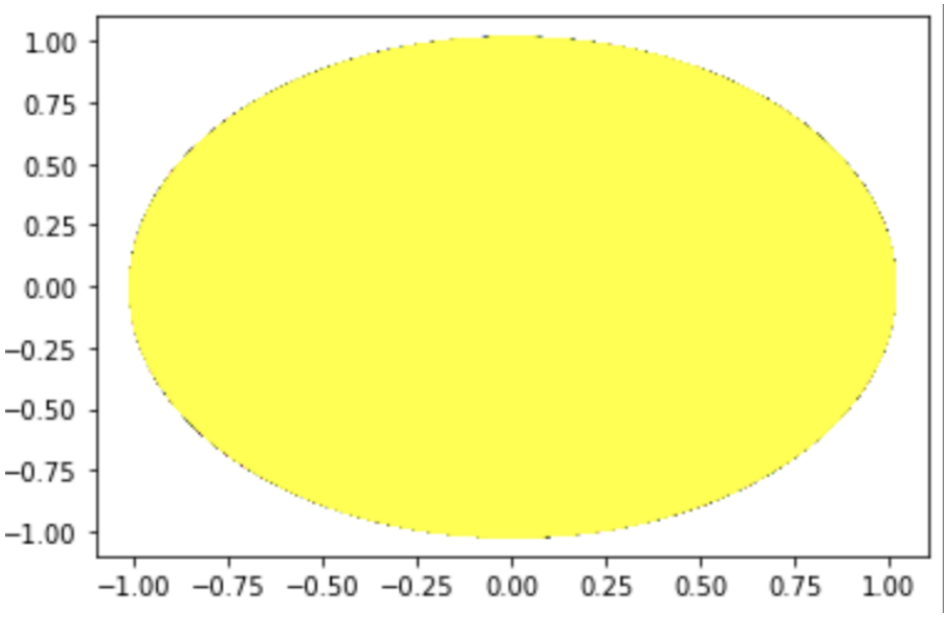
Nessa execução, o resultado obtido do PI foi de 3.15, com uma taxa de erro de 0.015.

Na segunda execução, com o objetivo de ser mais assertiva, foram criados 500000 pontos em cada worker, resultando em um círculo perfeito em cada worker, como pode ser visto o worker verde e preto nas imagens:





Já o círculo com todos os pontos ficou totalmente amarelo, pois foi o último worker a ser executado.



Nesta execução o valor de PI obtido foi de 3.1403, com uma taxa de erro de 0.001.