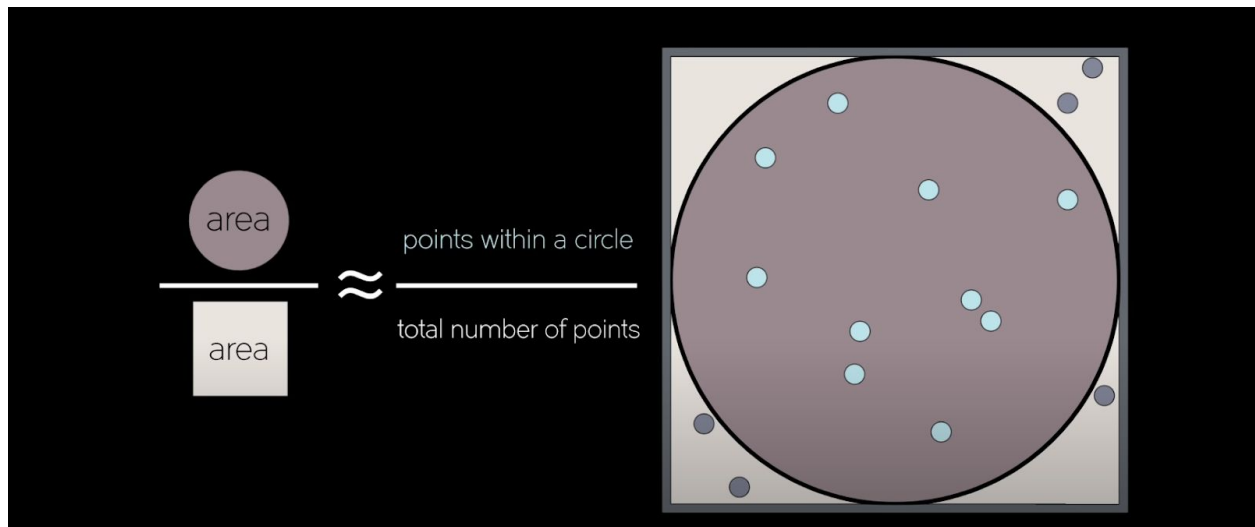


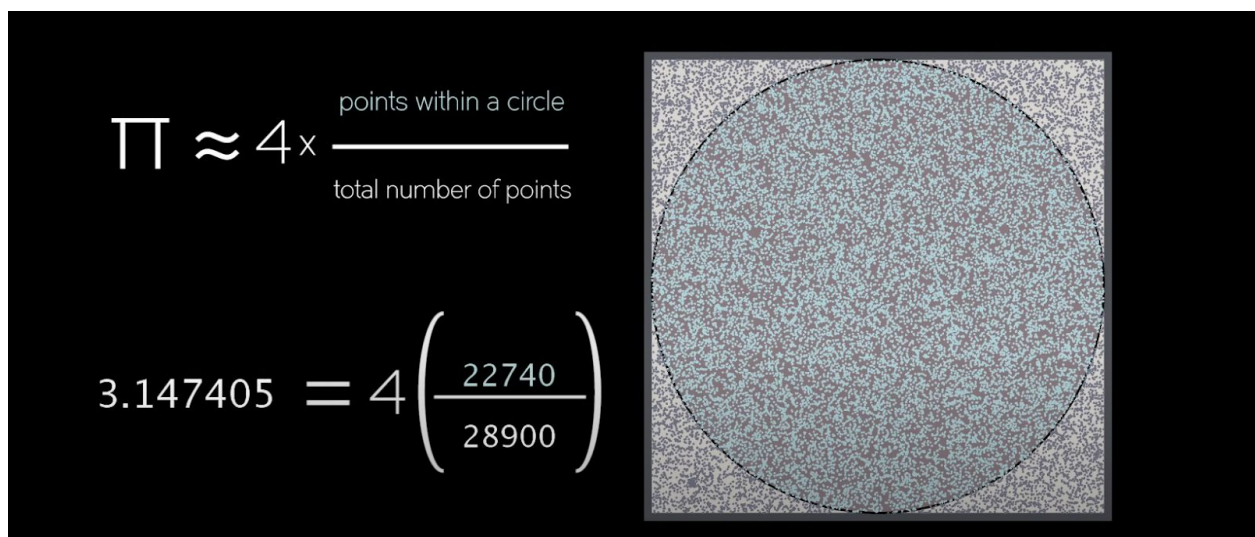
O problema a ser resolvido é o do Monte Carlo, Os métodos dele são uma ampla classe de algoritmos computacionais que dependem de amostragem aleatória repetida para obter resultados numéricos. A ideia é simular pontos aleatórios (x, y) em um plano 2-D com domínio como um quadrado de lado 1 unidade. Imagine um círculo dentro do mesmo domínio com o mesmo diâmetro e inscrito no quadrado. Em seguida, calculamos a proporção de pontos numéricos que se encontram dentro do círculo e o número total de pontos gerados. É possível ver na imagem a seguir uma melhor demonstração de como funciona a equação com base nesses pontos.

Figura 1



Em si é apenas uma conta simples, quando olhada dessa maneira. Podemos ver que a maneira de calcular o pi é os pontos dentro do círculo dividido pelos total de pontos que foram colocados de maneira randômica, vezes 4. Na próxima imagem é possível ver a fórmula em ação.

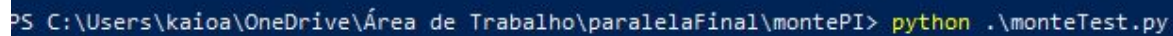
Figura 2



Como é possível ver, para uma grande aproximação foram colocados muitos pontos.

Para a resolução do serviço foi necessário implementar um serviço de Master e Worker, onde nele se tem um mestre, o líder, que pede task, ou seja trabalhadores para fazerem tarefas. Cada uma dessas tarefas é criada para fazer uma função. No nosso caso é possível passar quantos testes em cada worker serão feitos.

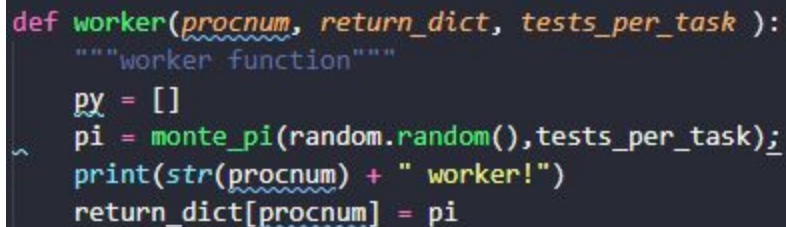
Figura 3



```
PS C:\Users\kaioa\OneDrive\Área de Trabalho\paralelaFinal\montePI> python .\monteTest.py
```

Explicando ele dentro do código seria o seguinte. Para uma melhor aproximação é necessário fazer vários testes e ir tirando os resultados dentro deles. Por conta disso é ótimo a utilização do mesmo, para todos os workers trabalharem de forma paralela e conseguirem ter um bom desempenho de tempo. Cada teste passado seria quantos pontos serão colocados, ou seja 1000 seria a quantidade de pontos e 5 seria a quantidade de workers criados. Sendo nesse caso o total de 5000 pontos. Claro que esse é apenas um exemplo, para uma melhor aproximação será necessário uma quantidade maior. Na imagem a seguir é possível ver como funcionam os workers dentro do código.

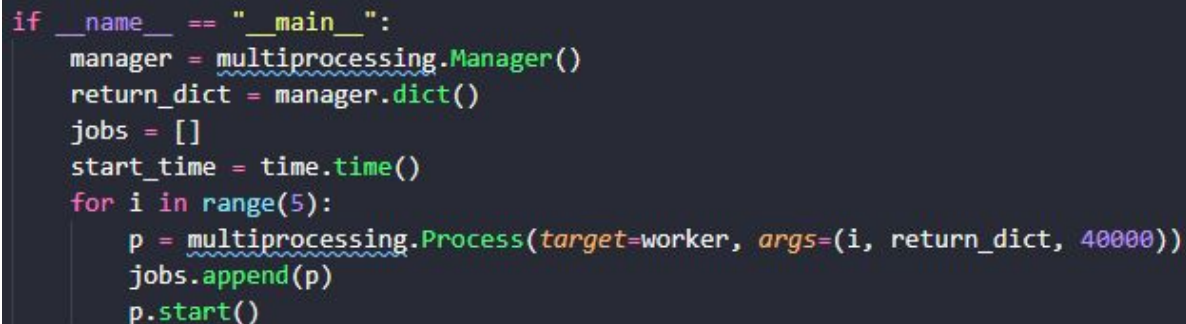
Figura 4



```
def worker(procnum, return_dict, tests_per_task):  
    """worker function"""  
    py = []  
    pi = monte_pi(random.random(), tests_per_task);  
    print(str(procnum) + " worker!")  
    return_dict[procnum] = pi
```

Nesta imagem é possível ver que estamos criando vários workers pela função de submit_task e pelo comando "for".

Figura 5



```
if __name__ == "__main__":  
    manager = multiprocessing.Manager()  
    return_dict = manager.dict()  
    jobs = []  
    start_time = time.time()  
    for i in range(5):  
        p = multiprocessing.Process(target=worker, args=(i, return_dict, 40000))  
        jobs.append(p)  
        p.start()
```

Nessa imagem está sendo mostrado o método de Monte Carlo em python, onde é possível ver a primeira função verificando se o ponto está dentro do círculo e a segunda fazendo a criação dos pontos.

Figura 6

```
def throw_dart():  
    pt = math.pow(random.random(),2) + math.pow(random.random(),2)  
    if (math.pow(random.random(),2) + math.pow(random.random(),2) ) <= 1: return 1  
    else: return 0  
  
def monte_pi(rand_seed, num_tests):  
    random.seed(rand_seed)  
    num_hits = 0  
  
    for i in range(int(num_tests)):  
        num_hits += throw_dart()  
  
    return [num_hits, num_tests]
```

Por fim, são contados os pontos que estão fora e dentro, por conta de serem vários workers é necessário somá-los. Na imagem a seguir é possível ver essa ação por conta do método “for” que passa em todos os workers.

Figura 7

```
for proc in jobs:  
    proc.join()  
print(return_dict.values())  
  
num_hits = 0  
num_tests = 0  
  
for points in return_dict.values():  
    num_hits += points[0]  
    num_tests += points[1]  
  
end_time = time.time()  
pi_estimate = 4.0 * float(num_hits)/num_tests  
  
print(("Estimate of pi:", pi_estimate))  
print(("Estimate error:", abs(pi_estimate-math.pi)))  
print(("Total time:", str(end_time-start_time)))
```

Figura 8

```
PS C:\Users\kaioa\OneDrive\Área de Trabalho\paralelaFinal\montePI> python .\monteTest.py
1 worker!
4 worker!
0 worker!
3 worker!
2 worker!
[[31379, 40000], [31442, 40000], [31381, 40000], [31524, 40000], [31473, 40000]]
('Estimate of pi:', 3.14398)
('Estimate error:', 0.002387346410206881)
('Total time:', '0.18304109573364258')
```