Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal Catarinense
Campus Videira

VINICIUS SCHULTE

IMPLEMENTAÇÃO DO MÉTODO DE APROXIMAÇÃO DO PI DE MONTE CARLO

Videira
2021

RESUMO

Este trabalho tem como objetivo apresentar a criação de um algoritmo que realiza o cálculo do valor de Pi (Π) através do método de Monte Carlo, fazendo uso da linguagem Python. Este método faz uso de números aleatórios e estatísticas para resolver problemas. Com a geração destes números aleatórios e possível alcançar os valores que poderão ser utilizados para resolver o problema.

1 INTRODUÇÃO

A letra grega Π (pi) é usada para representar uma constante matemática. O valor de Π é um número decimal com infinitas casas decimais, e essa constante na maioria dos casos é arredondada para 3,14, para ser facilitado o seu uso. (MATERIA, 2020)

Como as calculadoras limitam o números de casas decimais, pois não cabem tantas casas nelas. A descoberta de mais casas, além das encontradas por calculadoras, tornou-se possível graças aos computadores. (MATERIA, 2020)

Com isso este trabalho apresenta a criação de um algoritmo que desempenha a função de calcular o valor estimado do Pi fazendo uso do método de Monte Carlo e o uso da tecnologia Python, utilizando a programação paralela para ajudar na resolução .

2 DESENVOLVIMENTO

2.1 MÉTODO DE MONTE CARLO

O valor de PI (Π) pode ser calculado de diversas formas, e uma delas é o método de estimativa do valor de PI, proposto por Monte Carlo. É possível observar na Figura 1 que os valores dos eixos X e Y são delimitados utilizando o método de gerar números randômicos (random), que pode gerar números com o intervalo entre 0 e 1.

Figura 1 – Geração de coordenadas

```
deltaX = random.random()
deltaY = random.random()
```

Fonte: O Autor, 2021

Para determinar se um ponto está contido na circunferência é necessário calcular a distância entre dois pontos, como apresentado na Figura 2. Fazendo uso do teorema de pitágoras é possível determinar qual a distância entre a origem e o ponto, determinando se o ponto está dentro do círculo. A equação da distância é feita desta forma: $distncia = \sqrt{\Delta x^2 + \Delta y^2}$

Figura 2 – Calculo da Distância

```
distancia = math.sqrt(deltaX**2 + deltaY**2)
```

Fonte: O Autor, 2021

Caso o resultado deste cálculo sejá menor ou igual a 1, será considerado que o ponto está dentro do círculo, na linguagem Python é possível representar essa condição utilizando "if", que é possível ver na Figura 3.

Figura 3 – Condição do Círculo

```
# Verifica se o ponto ta dentro do circulo
if distancia <= 1:
    pontosCirc += 1
```

Fonte: O Autor, 2021

2.2 MULTIPROCESSING

Como demonstra na Figura 4, foi utilizada uma biblioteca de multiprocessamento (Multiprocessing), que segundo a própria biblioteca do Python, é um pacote que suporta geração de processos usando uma API semelhante ao módulo de threading. Com isso é possível retornar o número de CPUs existentes no computador em que o código está sendo executado.

Figura 4 – Multiprocessing

```
np = multiprocessing.cpu_count()
print(F'Você tem {np} CPUs')
```

Fonte: O Autor, 2021

2.3 BALANCEAMENTO DE CARGA

Para existir um balanceamento de carga, onde todos os processos requerem trabalho igual, eles serão divididos igualmente, como a Figura 5 apresenta. Aonde será criado um array de n valores chamado totalPontos, que irá receber os valores do número de pontos totais divididos pela quantidade de CPUs e multiplícados também pela quantidade de CPUs (para ampliar o tamanho do array).

Figura 5 – Balanceamento de carga

```
totalPontos = [n/np] * np
```

Fonte: O Autor, 2021

2.4 CRIAÇÃO DOS WORKERS

Após ser feito o balanceamento de carga, será utilizada uma funcionalidade (Pool), que está presente na biblioteca de multiprocessamento, os workers serão criados tendo como base a quantidade de CPUs encontradas. Aonde o resultado de cada ponto que está dentro do círculo será obtida, podendo ser vista na Figura 6, com a função (calculoDePontos), e com as iterações (totalPontos), que foram gerados no balancemanto de carga.

Figura 6 – Workers

```
pool = Pool(processes=np)

pontosCirc = pool.map(calculoDePontos, totalPontos)
print("Pontos de cada worker:", totalPontos)
print("Pontos de cada worker que esta dentro do circulo:", pontosCirc)
```

Fonte: O Autor, 2021

2.5 CÁLCULO DO PI

Com o resultado dos pontos obtidos em cada worker, é possível calcular o valor aproximado de PI, aonde será feito um somatório dos pontos de cada worker, dívidido pelo número total de pontos, e multiplicado por 4, como na Figura 7.

Figura 7 – Cálculo do PI

```
pi = sum(pontosCirc) / n * 4
print("Valor de aproximado de PI: ", pi)
```

Fonte: O Autor, 2021

3 RESULTADOS ENCONTRADOS

Como alguns valores são gerados de forma aleatória, o resultado final pode sofrer variação. Além desta, outra variação que ocorre é com a quantidade de pontos gerados, quanto mais pontos, mais a aproximação do PI irá aumentar, como visto na Tabela 1.

Tabela 1 – Resultados com mudança da quantidade de pontos

Pontos	Workers	Resultado
100	4	3.2
1000	4	3.192
10000	4	3.1748
100000	4	3.1348
1000000	4	3.142668
10000000	4	3.1411428

Fonte: O Autor, 2021

Outra váriação que pode sofrer é no tempo de execução que irá depender da quantidade de workers, como apresentado na Tabela 2.

Tabela 2 – Variação de tempo com diferentes workers

Pontos	Workers	Tempo
10000000	1	9.47s
10000000	2	4.88s
10000000	3	3.49s
10000000	4	2.91s

Fonte: O Autor, 2021

REFERÊNCIAS

MATERIA, Toda. **Numero do Pi**. 2020. <https://www.todamateria.com.br/numero-pi/>.

PYTHON, Biblioteca. **multiprocessing — Process-based parallelism**. ://docs.python.org/3/library/multiprocessing.html.