Programação Paralela e Multicore
Problema 2: Cálculo do PI

Luan Bruno de Melo

O problema consiste em apresentar um algoritmo que seja capaz de calcular o valor de PI utilizando o método de Monte Carlo, que foi inventado por volta da segunda guerra mundial, com o objetivo de calcular áreas abaixo de curvas ou calcular integrais com múltiplas dimensões. Além disso, o código precisa funcionar de forma paralela.

Para realizar esse cálculo, é utilizado a ideia de aleatoriedade, onde o código cria pontos em posições aleatórias (x e y), que podem cair dentro ou fora de um círculo. Tendo esses dados, é possível chegar a um valor aproximado de PI, pegando o número de pontos que caíram dentro do círculo, multiplicando por quatro (4) e dividindo pelo número total de pontos.

A linguagem de programação escolhida para a realização do trabalho foi a *Python*. Para que o código funcione de forma paralela foi utilizado o pacote *multiprocessing*, possibilitando a geração de pontos, de forma paralela.

Para gerar o valor que representa o número de pontos que caíram dentro do círculo, foi feito uma função que irá receber primeiramente, como parâmetro, o número total de tentativas por processos (número de pontos gerados). Tendo esse valor, é criado uma variável (inside) para armazenar o número de pontos que caíram no círculo e um *for* que irá rodar baseado no valor do parâmetro da função. Dentro do for, será gerado dois valores aleatórios, que representando os valores de x e y, para serem utilizados na equação: x * x + y * y. Por fim, o resultado será verificado, sendo incrementado +1 à variável inside, toda vez que o valor for menor ou igual a 1, retornando o valor dessa variável.

```
def monte_compute(iterations):
    inside = 0
    for _ in range(iterations):
        a = random.random()
        b = random.random()
        c = math.pow(a, 2.0) + math.pow(b, 2.0)
        if c <= 1.0:
            inside += 1
    return inside
```

Para funcionar de forma paralela, foi desenvolvido uma função que recebe como parâmetros o número de pontos gerados e número de máximo de CPUs presentes na máquina, sendo esse valor gerado de forma automática, para que

assim, o código possa funcionar com o máximo de eficiência, em diferentes máquinas, sem precisar fazer ajustes. Isso foi possível usando a biblioteca *psutil*.

```
num_cpus = psutil.cpu_count(logical=True)
```

Obtendo esses parâmetros, é criado um *with* e utilizando a biblioteca *multiprocessing* e a classe pool, que irá executar recebendo o número de CPUs. Dentro dele, é incrementado valores a duas variáveis, a primeira é receber o valor de acertos de cada processo, em forma de uma lista, utilizando a função map, que possibilita aplicar uma lista (que no caso irá conter os valores máximos de tentativas por processos) a uma função. A segunda variável, irá somar todos os valores que estão dentro da primeira variável, chegando assim ao valor total de acertos de todos os processos.

```
def proc_monte_carlo_pi(num_cpus, iterations):
    with multiprocessing.Pool(num_cpus) as p:
        results = p.map(monte_compute, [iterations] * num_cpus)
        total_inside = sum(results)
    return total_inside
```

Tendo o valor de total de acertos, é possível fazer o cálculo de pi, utilizando o método de monte carlo, da seguinte forma: PI = PontosAcertados / PontosTotais * 4.

```
pi = total_inside / total_iterations * 4
```

Exemplo de uma saída com 1 milhão de pontos

```
Valor de PI: 3.14277
Tempo de execução: 1.32
```

Exemplo de uma saída com 10 milhões de pontos

```
Valor de PI: 3.1415774
Tempo de execução: 12.55
```