Calculadora de Fibonacci Paralela

Enzo Sodré, Viceleno Barros, Lucas Tardin e
Matheus Lima

Introdução

Nossa solução tem como proposta facilitar o cálculo de Fibonacci a partir da posição de um número enviado pelo usuário do programa. Para isso é necessário o entendimento de como esse cálculo é realizado e a utilização de threads para otimizar a performance do código que desenvolvemos.

Implementação

A sequência de Fibonacci, de maneira geral, é a ordem de números inteiros, que parte, normalmente, de zero e um no qual cada número seguinte é o resultado da soma dos dois anteriores. Uma curiosidade é que essa continuidade pode ser vista em vários fenômenos da natureza.

Seu surgimento ocorreu no século XIII, mais precisamente em 1202, com o matemático Leonardo de Pisa, conhecido como Fibonacci, nome pela qual a operação matemática foi apelidada.

Em termos matemáticos, essa operação pode ser definida pela fórmula Fn = Fn - 1 + Fn - 2, na qual esse método é aplicado não somente na ciência da computação, como se faz presente também na análise de mercados financeiros e teoria de jogos. A seguir será demonstrado um pequeno exemplo da ordem a ser seguida conforme o cálculo deste modelo:

Posição:	1°	2°	3°	4°	5°	6°	7°	8°	9°	10°	11°	12°	13°	14°	15°	16°	17°	18°
Valor:	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597…

Em relação ao código, foi utilizada a linguagem Python, com a inserção de threads, que tem o objetivo de paralelizar o código, mantendo uma maior eficiência com a divisão de trabalhos entre os núcleos do processador da máquina utilizada.

Fez-se necessária a utilização de bibliotecas (threading e time), para inicializar e finalizar o uso de threads e calcular o tempo de execução respectivamente. A função ‘calculadora_fibonacci()’, apresenta passagem do parâmetro (n), que é o número inteiro digitado pelo usuário para que seja feita a operação do n-ésimo termo da sequência de Fibonacci, na qual é realizada a soma dos dois termos anteriores para saber o termo atual, utilizando variáveis para representar os números anteriores (n1, n2) para descobrir o resultado, assim como um loop que faz o cálculo com um recurso do Python chamado de atribuição múltipla, retornando o n-ésimo número da sequência de Fibonacci.

```
import threading
import time

def calculadora_fibonacci(n):
    """
      Função para calcular o n-ésimo número da sequência de
Fibonacci.

    Args:
        n (int): O número da posição na sequência.

    Returns:
        int: O valor da posição n na sequência de Fibonacci.
    """
    if n == 1:
        return 0
    elif n == 2:
        return 1


    n1 = 0
    n2 = 1


    for _ in range(2, n):
        n1, n2 = n2, n1 + n2


    return n2
```

A função ‘calcular_blocos()’ realiza o cálculo dos números de Fibonacci em um intervalo específico, com passagem de parâmetro (start, end) que representam o início e fim do intervalo, respectivamente. É feita uma lista para armazenar os cálculos e um loop que chama a função ‘calculadora_fibonacci()’, retornando a lista com os resultados dessa operação.

```
def calcular_blocos(start, end):
    """
       Função para calcular os números de Fibonacci em um
intervalo específico.


    Args:
        start (int): O início do intervalo (inclusive).
        end (int): O final do intervalo (exclusive).


    Returns:
         list: Uma lista dos números de Fibonacci no intervalo
especificado.
    """
    resultados = []
    for i in range(start, end):
        resultado = calculadora_fibonacci(i)
        resultados.append(resultado)
    return resultados
```

Nesta parte do código, há o cálculo dos números de Fibonacci utilizando threads, com passagem de parâmetro (n, num_threads) que representam o número que o usuário digitou e o número de threads que o mesmo quer realizar a operação paralelizada. São criadas duas listas para armazenar a criação e os resultados calculados pelas threads.

Um loop é criado para dividir as operações entre as threads, em que o ‘start’ e ‘end’ determinam o intervalo de cada bloco de cálculo, havendo a comunicação com a função ‘calcular_blocos()’ e alocando os resultados em uma lista, para que posteriormente aconteça a agregação dos resultados utilizando o método ‘join()’, com o retorno da lista ‘resultados’.

```
def paralelizar(n, num_threads):
    """
     Função para calcular os números de Fibonacci em paralelo
usando threads.
```

```
    Args:
        n (int): O número da posição na sequência de Fibonacci
a ser calculada.
        num_threads (int): O número de threads a serem usadas
para paralelizar o cálculo.


    Returns:
            list: Uma lista contendo os números de Fibonacci
calculados em paralelo.
    """
    threads = []
    resultados = []

    # Dividir o cálculo em blocos para cada thread
    tamanho_bloco = n // num_threads
    for i in range(num_threads):
        start = i * tamanho_bloco + 1
        end = (i + 1) * tamanho_bloco + 1 if i < num_threads -
1 else n + 1
            t = threading.Thread(target=lambda s=start, e=end:
resultados.extend(calcular_blocos(s, e)))
        threads.append(t)
        t.start()

    # Aguardar todas as threads terminarem
    for t in threads:
        t.join()

    return resultados
```

Neste trecho de código, há a interação com o usuário, em que é solicitado o número de threads que o cálculo será realizado, assim como a posição da sequência de Fibonacci.

```
num_threads = int(input('Digite o número de Threads para o cálculo de Fibonacci: '))


# Número para calcular a sequência de Fibonacci
n = int(input('Digite um número para o cálculo de Fibonacci: '))


print(f'Este é o cálculo usando a iteração do número {n} com {num_threads} Threads!')
```

Aqui é feita a marcação do tempo de execução para o cálculo do programa, utilizando as variáveis ‘start_time’ e ‘end_time’ para marcá-las e posteriormente obter o tempo gasto nessa operação fazendo a subtração das mesmas.

```
start_time = time.time()  # Marca o tempo de início
resultado = paralelizar(n, num_threads)
end_time = time.time()  # Marca o tempo de término

execution_time = end_time - start_time  # Calcula o tempo total de execução
```

Por fim, o resultado é demonstrado ao usuário, imprimindo o último elemento da lista ‘resultado’ contendo o cálculo da operação e o tempo total gasto em segundos, fornecendo informações benéficas para a obtenção de desempenho e otimização.

```
print("Resultado:", resultado[-1])  # Exibe o último resultado da sequência
print("Tempo de execução:", execution_time, "segundos")  # Exibe o tempo de execução
```

Conclusão

É possível destacar que essa abordagem paralela pode ser mais rápida do que calcular a soma sequencialmente, especialmente para valores grandes de ‘n’ (número da posição inserido pelo usuário) e quando há recursos de hardware disponíveis para executar as threads simultaneamente. Destacamos também que cada resultado tem variação de acordo com a máquina e quantidade de threads utilizadas.

Citações e Referências

<https://www.educamaisbrasil.com.br/enem/matematica/sequencia-de-fibonacci>. Acesso em: 04 de maio de 2024.