Documentação Ordenação Paralela com Threads

Professor: Aldo Henrique

Alunos: Felipe de Macedo, Kaio de Oliveira, Ryan Barcelos

1. Descrição:

Este código implementa o algoritmo de ordenação QuickSort em Python. Ele oferece a opção de ordenar uma lista de dados de forma paralela, utilizando threads, ou de forma sequencial. O usuário fornece o nome de um arquivo contendo os dados a serem ordenados, e pode escolher o número de threads a serem utilizadas na ordenação paralela.

2. Funções:

- quick_sort(arr): Implementa o algoritmo de ordenação QuickSort para ordenar uma lista de elementos.
- escrever_dados_ordenados(nome_arquivo, dados): Escreve os dados ordenados em um arquivo de texto.
- escrever_tempo_execucao(nome_arquivo, tempo_execucao): Escreve o tempo de execução em um arquivo de texto.
- ordenar_dados(dados): Função auxiliar que chama a função quick_sort() para ordenar uma lista de dados.
- analisar_desempenho(tempo_inicio, tempo_fim): Calcula o tempo de execução com base no tempo de início e fim.

3. Fluxo de Execução:

- O programa solicita ao usuário o nome do arquivo que contém os dados a serem ordenados.
- Os dados são lidos do arquivo e convertidos para uma lista de inteiros (se possível).
- O usuário é questionado se deseja executar o código.
- Se o usuário optar por executar o código:
    - É perguntado quantas threads deseja utilizar para a ordenação paralela.
    - Os dados são divididos em partes iguais para cada thread.

- Um ThreadPoolExecutor é iniciado para paralelizar a ordenação dos dados.
- Os dados ordenados usando threads são escritos em um arquivo.
- O tempo de execução da abordagem paralela é gravado em um arquivo.
- Os dados são ordenados e o tempo de execução é gravado em um arquivo.
- Se o usuário optar por não executar o código, o programa é encerrado.

4. Utilização de Threads:

O código utiliza a biblioteca concurrent.futures para paralelizar a ordenação dos dados, aproveitando os recursos de multithreading oferecidos pelo Python. Isso permite uma execução mais rápida da ordenação em sistemas com múltiplos núcleos de CPU.

5. Considerações:

O algoritmo QuickSort é eficiente para ordenar grandes conjuntos de dados. A utilização de threads proporciona uma abordagem paralela para acelerar o processo de ordenação, aproveitando o poder de processamento de múltiplos núcleos da CPU.

A escolha entre a abordagem paralela e sequencial depende do tamanho dos dados e da disponibilidade de recursos do sistema.

6. Observações:

O código está estruturado de forma modular, facilitando a compreensão e manutenção. Comentários foram adicionados para explicar o propósito de cada função e bloco de código, tornando o código mais legível e acessível.