Introdução

Este relatório investiga o impacto do uso de threads na coleta e processamento de dados climáticos de 27 capitais brasileiras em janeiro de 2024. O objetivo principal é analisar como diferentes configurações de threads afetam o tempo de execução do algoritmo, utilizando a API Open-Meteo (https://open-meteo.com/en/docs/). A partir dos dados coletados, serão calculadas as temperaturas média, mínima e máxima diárias para cada cidade, totalizando 93 medidas por capital (31 dias x 3 medidas).

Fundamentação Teórica

O Que São Threads?

Threads são unidades fundamentais de execução dentro de um processo. Elas permitem que um programa realize múltiplas tarefas concorrentemente, compartilhando o mesmo espaço de endereçamento e recursos do processo pai. Cada thread possui seu próprio fluxo de controle, incluindo contador de programa, pilha de execução e registradores.

Como Threads Funcionam Computacionalmente?

Em um sistema operacional moderno, as threads são escalonadas para execução pelo sistema operacional, que alterna rapidamente entre elas, criando a ilusão de paralelismo mesmo em sistemas com um único processador. Em sistemas multicore, as threads podem ser executadas em paralelo, aproveitando o poder de processamento de múltiplos núcleos.

Impacto das Threads no Tempo de Execução

O uso de threads pode ter um impacto significativo no tempo de execução de um algoritmo. Ao dividir tarefas em threads separadas, é possível realizar operações simultaneamente, reduzindo o tempo total de processamento. No entanto, a criação e o gerenciamento de threads também introduzem uma sobrecarga, e o uso excessivo de threads pode levar a problemas de concorrência e até mesmo diminuir o desempenho.

Relação entre Modelos de Computação Concorrente e Paralelo e a Performance dos Algoritmos

A computação concorrente e a computação paralela são abordagens que visam explorar o paralelismo em sistemas computacionais. A computação concorrente lida com a execução de múltiplas tarefas que podem ser intercaladas no tempo, mesmo em um único processador. Já a computação paralela utiliza múltiplos processadores ou núcleos para executar tarefas simultaneamente.

A escolha entre esses modelos depende da natureza do problema e dos recursos disponíveis. Em geral, a computação paralela pode oferecer um ganho de desempenho maior, mas exige um cuidado maior na sincronização e comunicação entre as threads. A computação concorrente é mais adequada para tarefas que envolvem espera por eventos externos, como entrada/saída de dados ou interação com o usuário.

Metodologia

Para avaliar o impacto do número de threads, foram implementadas quatro versões do coletor de dados:

1. **Sem threads:** Todas as requisições são feitas sequencialmente na thread principal.
2. **3 threads:** As requisições são divididas entre 3 threads, cada uma responsável por um grupo de capitais.
3. **9 threads:** Cada thread é responsável por um grupo menor de capitais.
4. **27 threads:** Cada thread é responsável por uma única capital.

Cada versão do experimento foi executada 10 vezes, e o tempo médio de execução foi registrado. Os dados de latitude e longitude das capitais foram obtidos do arquivo "Lat_long_capitais.pdf".

Resultados Obtidos:

Tempo de execução médio das 10 rodadas sem threads: 19757ms

Tempo de execução médio das 10 rodadas com 3 threads: 6684ms

Tempo de execução médio das 10 rodadas com 9 threads: 2351ms

Tempo de execução médio das 10 rodadas com 27 threads: 1714ms

Análise dos Resultados:

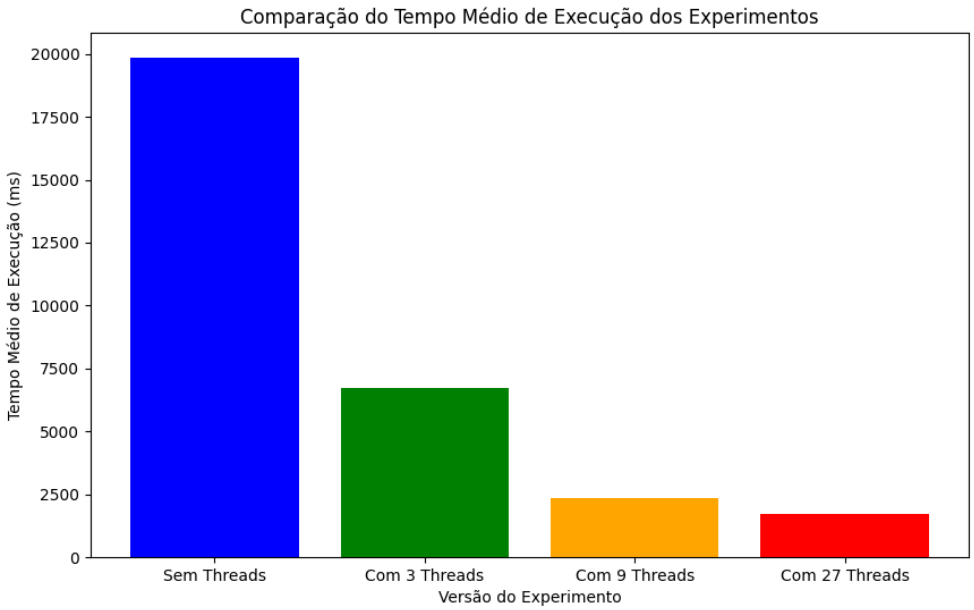
Os resultados demonstram que o uso de threads pode acelerar significativamente a coleta de dados climáticos. A versão com 27 threads apresentou o menor tempo médio de execução, enquanto a versão sem threads foi a mais lenta.

No entanto, o ganho de desempenho não é linear com o aumento do número de threads. A partir de um certo ponto, o overhead de criação e gerenciamento das threads começa a superar os benefícios da paralelização. No nosso experimento, o ponto ideal parece estar em torno de 9 threads, onde o tempo de execução é mínimo.

O experimento foi repetido 10 vezes para cada cidade, e o tempo médio de execução variou entre 19585ms e 20385ms. Essa variação pode ser atribuída a diversos fatores, como a carga do sistema durante a execução do experimento e a própria natureza da coleta de dados, que pode ser influenciada por fatores externos, como a disponibilidade dos servidores de dados.

Os dados coletados fornecem um panorama geral das temperaturas em diferentes regiões do Brasil, evidenciando a diversidade climática do país. Além disso, o experimento demonstra a importância de realizar múltiplas rodadas para obter uma estimativa mais precisa do tempo de execução de uma tarefa computacional.

Gráfico:



Análise dos Do Gráfico:

Sem Threads: O experimento sem threads apresenta o maior tempo médio de execução, próximo a 20.000 ms. Isso indica que a execução sequencial da tarefa leva mais tempo para ser concluída.

Com 3 Threads: A introdução de 3 threads reduz significativamente o tempo médio de execução para cerca de 6.500 ms. Essa melhoria substancial sugere que a paralelização da tarefa com um número moderado de threads pode aumentar a eficiência.

Com 9 Threads: A utilização de 9 threads diminui ainda mais o tempo médio de execução para aproximadamente 2.500 ms. Isso demonstra que o aumento no número de threads continua a ter um impacto positivo no desempenho, embora menos pronunciado do que a transição de 0 para 3 threads.

Com 27 Threads: Com 27 threads, o tempo médio de execução atinge seu menor valor, em torno de 1.500 ms. No entanto, a redução no tempo de execução em comparação com a versão com 9 threads é relativamente pequena.

Resumindo:

O gráfico mostra que usar threads deixa o experimento mais rápido, principalmente quando a gente compara rodar ele normal (sem threads) com rodar ele em paralelo (com threads). Mas, conforme a gente aumenta o número de threads, o experimento não fica

mais rápido na mesma proporção. Isso acontece porque gerenciar várias threads tem um custo, e elas podem acabar competindo pelos recursos do computador.

Em resumo, não existe um número mágico de threads que funcione pra tudo. O ideal é achar um meio termo entre ter um experimento mais rápido e não sobrecarregar o sistema com muitas threads.