Relatório Técnico

Análise de Performance com Uso de Threads em Java

1. Explicação Teórica

"O que são threads?"

Threads são unidades básicas de execução dentro de um processo. Um único processo pode conter diversas threads que compartilham o mesmo espaço de memória, mas executam tarefas de forma independente. Isso permite que diferentes partes de um programa sejam executadas em paralelo ou de forma concorrente, dependendo da arquitetura do sistema operacional e do hardware. O uso de threads melhora a eficiência dos programas, especialmente em aplicações que envolvem múltiplas tarefas simultâneas, como acesso à rede, leitura de arquivos e processamento paralelo.

"Como threads funcionam computacionalmente?"

Computacionalmente, uma thread é gerenciada pelo sistema operacional, que controla sua criação, execução e finalização. Em sistemas com múltiplos núcleos de CPU, threads podem ser executadas em paralelo, cada uma em um núcleo distinto. Já em sistemas com núcleo único, as threads compartilham o tempo de processamento por meio de escalonamento. As threads de um mesmo processo compartilham variáveis globais, o que reduz o custo de criação em relação aos processos tradicionais, mas exige cuidados com sincronização e concorrência, para evitar erros como condições de corrida.

"Como o uso de threads pode afetar o tempo de execução de um algoritmo?"

O uso de múltiplas threads pode reduzir significativamente o tempo de execução de algoritmos que envolvem tarefas independentes ou paralelizáveis, como chamadas a APIs ou processamento em lote. No entanto, o ganho de desempenho depende de fatores como a quantidade de núcleos da CPU, o custo de criação e gerenciamento das threads e o tipo da tarefa executada. Se houver muitas threads competindo por

poucos núcleos, pode ocorrer overhead, tornando o desempenho pior do que uma execução sequencial.

"Qual a relação entre os modelos de computação concorrente e paralelo e a performance dos algoritmos?"

A computação concorrente permite que múltiplas tarefas estejam ativas ao mesmo tempo, ainda que não sejam executadas simultaneamente. Já na computação paralela, diferentes tarefas são de fato processadas ao mesmo tempo, cada uma por um núcleo distinto da CPU. Quando bem aplicada, a paralelização tende a melhorar significativamente a performance dos algoritmos, pois reduz o tempo total de execução. No entanto, nem todas as tarefas podem ser paralelizadas, e há um limite prático de eficiência: aumentar o número de threads além de certo ponto pode não trazer ganhos adicionais ou até gerar sobrecarga e piorar o desempenho.

2. Resultados e Análise Comparativa

Foram desenvolvidas quatro versões de um algoritmo em Java, cada uma utilizando uma quantidade diferente de threads para coletar dados climáticos das 27 capitais brasileiras por meio da API Open-Meteo. As versões são:

- **TemperaturaCapitais.java** (1 thread)
- **TemperaturaCapitais3Threads.java** (3 threads)
- **TemperaturaCapitais9Threads.java** (9 threads)
- **TemperaturaCapitais27Threads.java** (27 threads)

O experimento foi executado em 10 rodadas para cada versão, sendo calculado o tempo médio de execução para cada caso. Os resultados foram os seguintes:

Comparação do Tempo Médio de Execução por Quantidade de Threads

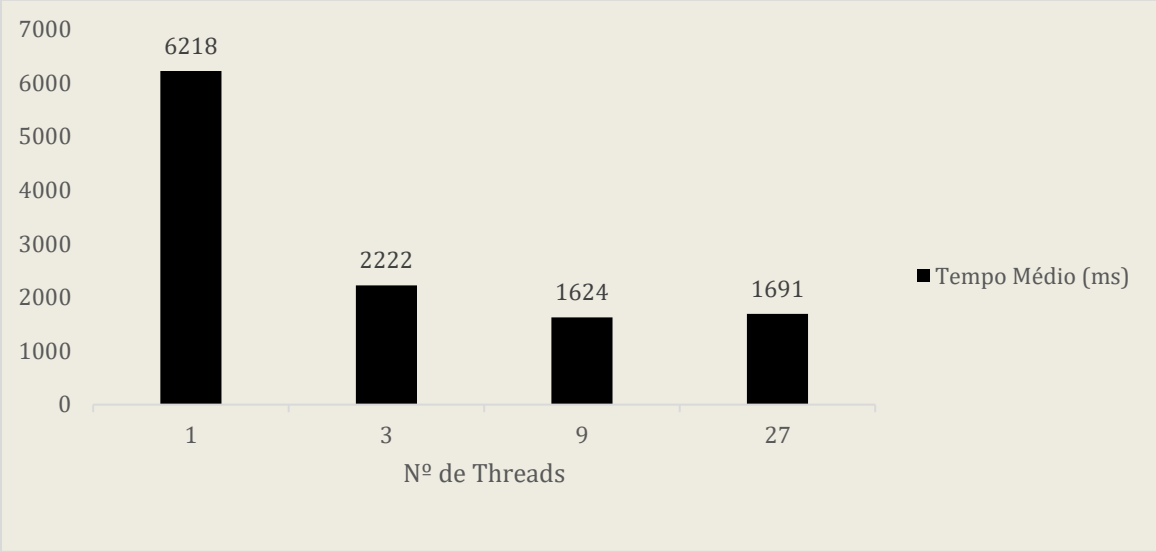

Figura 1 – Comparação do Tempo Médio de Execução por Quantidade de Threads
Fonte: Resultados do experimento manualmente registrados por Vitor Hugo.

Observa-se uma redução significativa no tempo de execução à medida que o número de threads aumenta de 1 para 3 e, depois, de 3 para 9. Esse comportamento demonstra o ganho de desempenho proporcionado pela paralelização. No entanto, ao

passar de 9 para 27 threads, o tempo médio se mantém praticamente estável, o que indica um ponto de saturação.

Esse fenômeno pode estar relacionado à limitação da quantidade de núcleos disponíveis no processador ou ao overhead gerado pelo gerenciamento de um número excessivo de threads. É necessário equilíbrio, pois mais threads nem sempre significam maior desempenho.

Esses resultados reforçam a importância do uso estratégico de threads para melhorar a performance, mas também alertam que o número ideal depende da arquitetura do sistema e da natureza das tarefas executadas.

3. Referências

- **SCHEFFER, R. Uma visão geral sobre threads. *Campo Digital*, v. 2, n. 1, p. 7–12, 2007. Disponível em: https://revista2.grupointegrado.br/revista/index.php/campodigital/article/download/311/145.**
- **TANENBAUM, A. S.; BOS, H. *Modern Operating Systems*. 4. ed. Pearson, 2015.**
- **SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Operating System Concepts*. 9. ed. Wiley, 2018.**
- Open-Meteo API: https://open-meteo.com/