Relatório

O que são threads?

Threads são unidades básicas de execução em um programa. Em um ambiente multitarefa, um processo pode conter múltiplas threads, que compartilham o mesmo espaço de memória, mas podem ser executadas simultaneamente. O uso de threads permite que diferentes partes de um programa sejam executadas ao mesmo tempo, aumentando a eficiência e a velocidade de processamento.

Como threads funcionam computacionalmente?

Threads são gerenciadas pelo sistema operacional ou pela máquina virtual (no caso de Java, pela JVM). Cada thread possui seu próprio contador de programa, registradores e stack, mas compartilha a memória heap com outras threads do mesmo processo. Isso permite uma comunicação eficiente, mas também requer mecanismos de sincronização para evitar condições de corrida e inconsistências de dados.

- **Contador de Programa**: Cada thread possui seu próprio contador de programa que mantém o controle de qual instrução a thread deve executar a seguir.
- **Registradores**: As threads têm seus próprios registradores, que são pequenos locais de armazenamento de dados de alta velocidade.
- **Stack**: Cada thread possui uma stack própria, onde são armazenadas variáveis locais, chamadas de métodos, etc.
- **Heap**: A memória heap é compartilhada entre todas as threads, permitindo que elas acessem e modifiquem dados comuns.

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

O uso de threads pode reduzir significativamente o tempo de execução de um algoritmo, especialmente em sistemas com múltiplos núcleos de processamento. Ao dividir uma tarefa em várias threads, partes do trabalho podem ser executadas simultaneamente em diferentes núcleos, aumentando a utilização da CPU e reduzindo o tempo total de execução. No entanto, a sobrecarga de criar e gerenciar threads pode, em alguns casos, compensar os benefícios, especialmente se o número de threads exceder a capacidade de processamento do sistema.

Relação entre modelos de computação concorrente e paralelo e a performance dos algoritmos

Computação concorrente e paralela são modelos usados para melhorar a performance dos algoritmos. A computação concorrente permite que múltiplas tarefas sejam realizadas em progresso ao mesmo tempo, gerenciando a execução de forma que elas se intercalem. Já a computação paralela divide tarefas em sub-tarefas que são executadas simultaneamente em diferentes núcleos de processamento. A escolha do modelo depende da natureza do problema e da arquitetura do sistema.

- **Computação Concorrente**: Envolve a execução de múltiplas tarefas ao mesmo tempo, mas não necessariamente ao mesmo tempo exato. As tarefas podem ser

interrompidas e retomadas, o que é útil em ambientes onde a resposta rápida é necessária.
- **Computação Paralela**: Implica na execução simultânea de tarefas em diferentes núcleos ou processadores. Ideal para tarefas que podem ser divididas em partes menores que podem ser executadas ao mesmo tempo.

Referências

- Java Guia do Programador - 3ª Edição
- Sistemas Operacionais Com Java
- https://www.devmedia.com.br/programacao-com-threads/6152

Exibição e Explicação dos Resultados

Tempo de execução sem threads: 5863 ms

Tempo de execução com 3 threads: 2009 ms

Tempo de execução com 9 threads: 923 ms

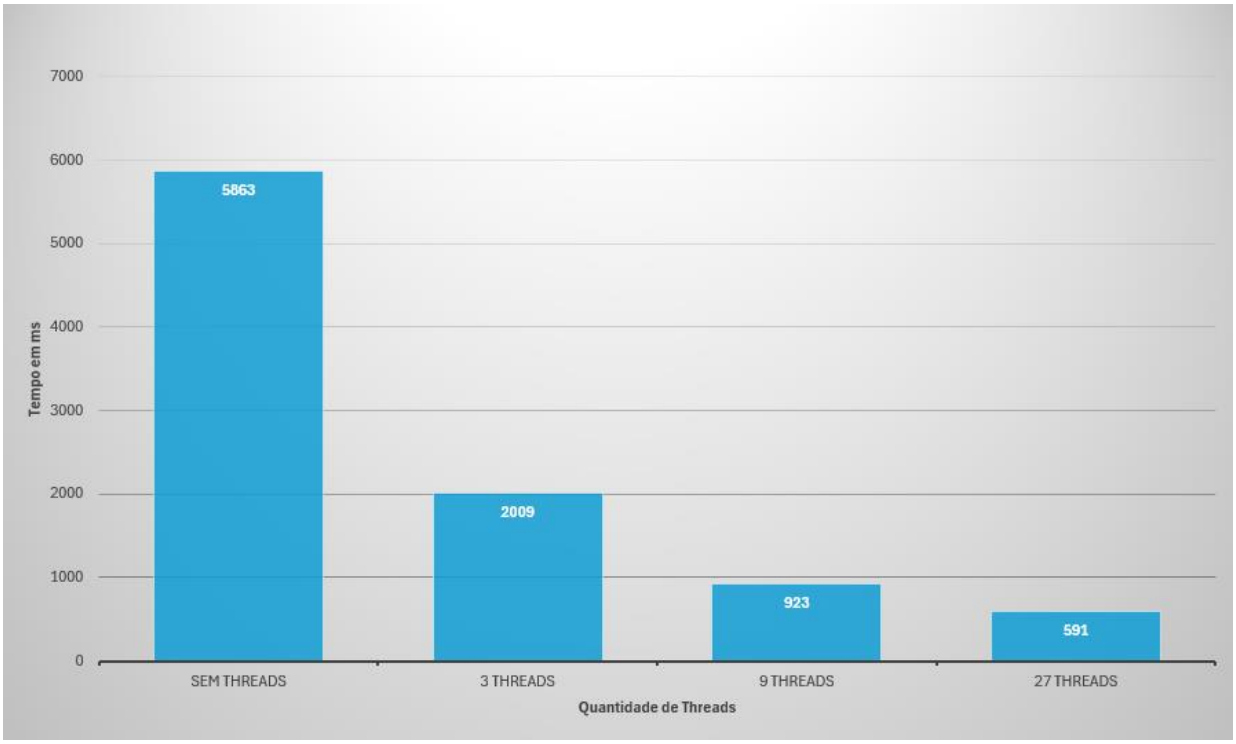
Tempo de execução com 27 threads: 591 ms

A introdução de threads no coletor de dados climáticos resultou em uma significativa redução no tempo de execução comparado ao método sequencial. Enquanto a execução sem threads levou aproximadamente 5863 milissegundos, a abordagem com 3, 9 e 27 threads reduziu os tempos para 2009 ms, 923 ms e 591 ms, respectivamente.

Essa melhoria ocorre devido à capacidade das threads de executarem múltiplas requisições HTTP simultaneamente. Isso permite um aproveitamento mais eficiente dos recursos do sistema, acelerando o processo de coleta e processamento de dados climáticos das cidades brasileiras analisadas.

Gráfico Comparativo

Aqui está um gráfico comparativo dos tempos de execução:



Este gráfico compara o tempo de execução (em milissegundos) de um programa que coleta dados climáticos para as 27 capitais brasileiras usando diferentes quantidades de threads. O eixo X representa a quantidade de threads utilizadas (SEM THREADS, 3 THREADS, 9 THREADS, 27 THREADS) e o eixo Y representa o tempo de execução em milissegundos.

1. **SEM THREADS**: O tempo de execução é de 5863 ms. Isso significa que o programa levou 5863 milissegundos para coletar os dados climáticos quando não usou threads (execução sequencial).
2. **3 THREADS**: O tempo de execução é de 2009 ms. Isso mostra que o programa levou 2009 milissegundos para coletar os dados quando usou 3 threads.
3. **9 THREADS**: O tempo de execução é de 923 ms. O programa levou 923 milissegundos para coletar os dados ao usar 9 threads.
4. **27 THREADS**: O tempo de execução é de 591 ms. Isso indica que o programa levou 591 milissegundos para coletar os dados ao usar 27 threads.

Análise:

- **Redução de Tempo com Aumento de Threads**: Observa-se uma significativa redução no tempo de execução conforme o número de threads aumenta. Usar threads permite que o programa execute várias tarefas simultaneamente, o que diminui o tempo total de execução.
- **Melhor Desempenho com 27 Threads**: O melhor desempenho é observado com 27 threads, onde o tempo de execução é o menor (591 ms). Isso faz sentido, pois cada thread pode ser responsável por uma cidade, permitindo uma execução paralela ideal.

- **Eficácia de Threads**: Mesmo com 3 threads, há uma grande redução no tempo de execução comparado à execução sequencial (sem threads), demonstrando a eficácia da multithreading.

Este gráfico destaca a importância do uso de threads para otimizar a performance de programas que necessitam executar múltiplas tarefas de maneira simultânea, como requisições HTTP para coleta de dados.