UNIVERSIDADE CATÓLICA DE BRASÍLIA – UCB

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

THREADS: Avaliação substitutiva

ISABELA THIFFANY RODRIGUES PEREIRA

BRASÍLIA – DF

2024

SUMARIO

INDRODUÇÃO

Threads, ou fios de execução, são uma técnica essencial em sistemas operacionais modernos que permite a execução simultânea de múltiplas atividades dentro de um mesmo processo. Este trabalho explora como as threads funcionam computacionalmente, seu impacto no tempo de execução de algoritmos e a relação entre modelos de computação concorrente e paralela na melhoria da performance dos sistemas. Além disso, é apresentado um experimento que compara diferentes configurações de threads em um projeto de previsão climática, analisando o efeito no tempo médio de execução e na eficiência do processamento paralelo.

O QUE SÃO THREADS

Threads, ou fios de execução, são uma técnica utilizada em sistemas operacionais para permitir que múltiplas atividades ocorram simultaneamente dentro de um único processo. Em sistemas operacionais tradicionais, cada processo tem seu próprio espaço de endereçamento e um único thread de controle. No entanto, em muitos casos, é desejável ter vários threads de controle dentro do mesmo espaço de endereçamento executando quase em paralelo.

Essas múltiplas threads de controle são como sub-processos dentro de um processo, compartilhando o mesmo espaço de endereçamento e recursos, mas com sua própria linha de execução. Isso permite que vários threads sejam executados em paralelo, melhorando a eficiência e a performance do sistema. Além disso, as threads também podem ser utilizadas para melhorar a resposta do sistema, permitindo que um processo seja interrompido e retomado em outro ponto, sem perder a sessão de trabalho.

COMO THREADS FUNCIONAM COMPUTACIONALMENTE

Computacionalmente as Threads funcionam dividindo o tempo de processamento em pequenas unidades, chamadas de ticks, e atribuindo cada tick a um thread específico. No estado de execução, o thread atua como um thread operário, executando o código de instruções do programa. Quando o thread operário não tem trabalho, ele entra no estado de bloqueio, esperando receber uma nova solicitação.

O despachante é o thread responsável por ler as requisições de trabalho e atribui-las aos threads operários disponíveis. Quando um thread operário está pronto para ser executado, o despachante o escolhe e passa a solicitação para ele. O thread operário verifica se a solicitação pode ser satisfeita a partir do cache da página da web e, se necessário, começa uma operação de leitura para obter a página do disco.

Se o thread operário está bloqueado aguardando por uma nova solicitação, outro thread é escolhido para ser executado, talvez o despachante ou outro operário esteja pronto para ser executado. Esse modelo permite que o servidor seja escrito como uma coleção de threads sequenciais, com o programa do despachante consistindo em um laço infinito para obter requisições de trabalho e entrega a um operário.

COMO O USO DE THREADS PODE AFETAR O TEMPO DE EXECUÇÃO DE UM ALGORITMO

O uso de threads pode melhorar a performance do sistema por permitir que múltiplas tarefas sejam executadas simultaneamente. Isso é especialmente útil em sistemas em que as tarefas possuem requisitos de tempo de resposta extremamente baixos, como em aplicações de tempo real. Quando um algoritmo é paralelizado, ele pode ser dividido em partes menores que sejam executadas ao mesmo tempo, o que pode reduzir significativamente o tempo de execução total.

No entanto, o uso de threads também pode levar a uma complexidade e sobrecarga aumentadas, o que pode negativamente impactar a performance. Isso porque threads adicionais podem criar mais recordações de memória, aumentar a pressão sobre a CPU do sistema e aumentar a complexidade do código. Além disso, a implementação adequada de threads pode ser desafiadora, pois requer a gestão correta de recursos compartilhados e a sincronização adequada entre os threads.

Além disso, o uso excessivo de threads pode também levar a problemas de concorrência, como paralelismo pré-maturo, onde os threads tentam acessar recursos compartilhados ao mesmo tempo, o que pode causar atrasos e erros. Portanto, a escolha adequada do nível de paralelismo e a implementação adequada dos threads são fundamentais para aproveitar o seu uso máximo.

QUAL A RELAÇÃO ENTRE OS MODELOS DE COMPUTAÇÃO CONCORRENTE E PARALELO E A PERFORMANCE DOS ALGORITMOS

Os modelos de computação concorrente e paralelo são fundamentais para entender como os algoritmos podem ser otimizados para melhorar a performance do sistema. A computação concorrente se refere ao processo de executar múltiplas tarefas ao mesmo tempo, compartilhando recursos e trocando informações entre elas. Isso pode ser alcançado através da utilização de threads, processos ou outras abordagens.

Por outro lado, a computação paralela se refere ao processo de executar múltiplas tarefas ao mesmo tempo, mas sem compartilhar recursos ou trocar informações entre elas. Isso pode ser alcançado através da utilização de processadores múltiplos, GPUs ou outras abordagens.

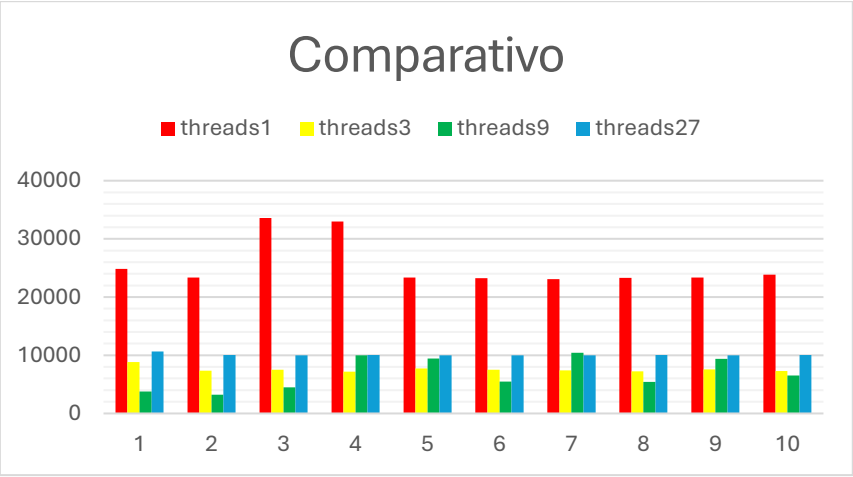
Ambos os modelos podem melhorar a performance do sistema, pois permitem que múltiplas tarefas sejam executadas simultaneamente. No entanto, a escolha do modelo a ser utilizado depende do algoritmo específico e do problema sendo abordado.

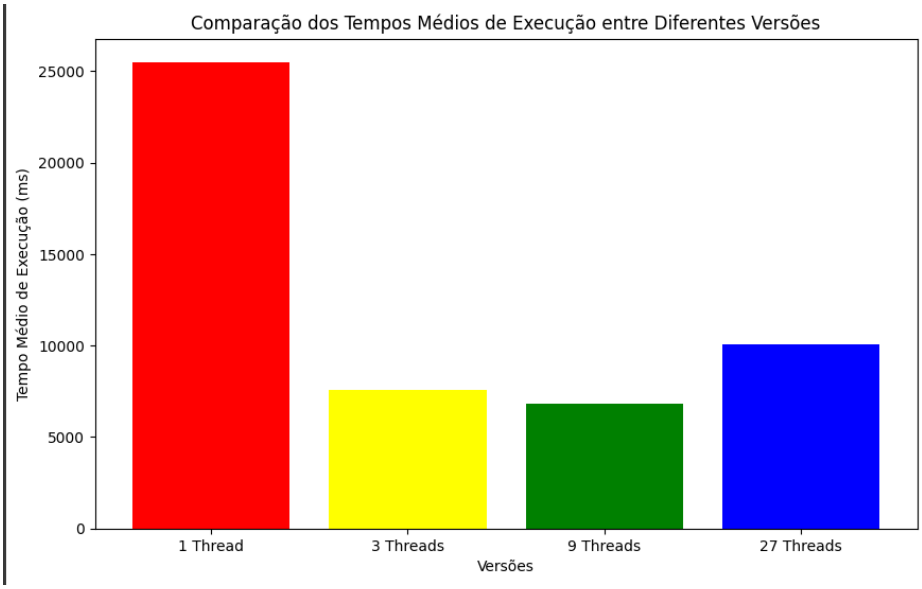
Segundo Tanenbaum "a escolha entre a computação concorrente e a computação paralela depende do tipo de problema que está sendo resolvido. Se o problema é altamente dependente de memória, a computação concorrente pode ser mais adequada, pois permite que os threads compartilhem recursos e troquem informações. No entanto, se o problema é altamente dependente de processamento, a computação paralela pode ser mais adequada, pois permite que os processadores múltiplos trabalhem em paralelo."

EXPERIMENTO

O experimento consiste na comparação de quatro versões do mesmo projeto, variando apenas o número de threads utilizadas, com o objetivo de avaliar o impacto no tempo médio de execução e na eficiência do processamento paralelo. Para assegurar a consistência dos resultados, todas as versões do projeto têm a mesma estrutura base e foram executadas dez vezes cada. Além disso, utilizaram-se as mesmas cidades em todas as versões do projeto, mudando apenas o número de threads.

Gráficos:





Na versão com 1 thread, apresentou o maior tempo médio de execução, 25.504,8 ms, isso aconteceu, pois, a execução sequencial não aproveita o processamento paralelo, assim gerando um desempenho mais lento.

Na versão com 3 threads, houve melhoria significativa no tempo médio de execução, reduzindo para 7.562,4 ms, como foi dividida as tarefas entre 3 threads, o projeto teve benefícios consideráveis, diminuindo o tempo médio de execução passando para 7562.4 ms.

Na versão com 9 threads, a eficiência no tempo médio de execução aumentou ainda mais, chegando a 6.814,9 ms, demonstrando que o aumento no número de threads continuou melhorando a eficiência do processamento paralelo.

Por outro lado, na versão com 27 threads, houve um aumento no tempo médio de execução para 10.084,2 ms, indicando que aumentar o número de threads além de certo ponto pode causar sobrecarga devido à competição por recursos e ao gerenciamento excessivo de threads.

Por fim, os resultados evidenciam que aumentar o número de threads de maneira controlada pode significativamente melhorar o desempenho do sistema, reduzindo o tempo médio de execução. No entanto, um aumento adicional de threads não resulta em melhorias proporcionais e pode até ocasionar degradação no desempenho devido à sobrecarga de gerenciamento de threads. Para este projeto, a utilização de 9 threads demonstrou ser a mais eficiente, enquanto o uso de 27 threads resultou em sobrecarga e tempos de execução mais longos.

REFERENCIAS

- Tanenbaum, Andrew S. Sistemas operacionais modernos (https://archive.org/details/SistemasOperacionaisModernosTanenbaum4Edio/page/n7/mode/2up)
- Open-Meteo (https://open-meteo.com/en/docs)