Relatório

O que são threads?

Threads são unidades básicas de execução que permitem que diferentes partes de um programa rodem simultaneamente. Cada thread opera dentro do contexto de um processo e compartilha os mesmos recursos, como memória e arquivos abertos. Diferente de processos independentes, threads são mais leves e têm uma sobrecarga menor para serem criadas e gerenciadas, já que compartilham o mesmo espaço de memória.

Threads são amplamente utilizadas em aplicações que precisam executar múltiplas tarefas ao mesmo tempo, como servidores de web que precisam processar várias requisições de clientes simultaneamente. Em um ambiente com múltiplos núcleos de processamento, elas podem ser distribuídas entre os núcleos, permitindo um aumento significativo na performance.

Como threads funcionam computacionalmente?

Do ponto de vista computacional, um thread é controlado pelo sistema operacional, que decide quando ela deve ser executada, pausada ou finalizada. O sistema operacional alterna a execução dos threads de maneira tão rápida que, para o usuário, parece que todas estão sendo executadas ao mesmo tempo. Essa alternância é conhecida como multitasking.

Quando um programa é dividido em múltiplas threads, cada uma é responsável por uma tarefa específica. Os threads podem ser executados de forma sequencial ou concorrente, dependendo de como o sistema operacional gerencia os threads e do número de núcleos disponíveis no processador.

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

O uso de threads pode melhorar significativamente o tempo de execução de um algoritmo em determinados cenários. Se o programa é capaz de dividir seu trabalho em tarefas independentes, que podem ser executadas ao mesmo tempo, o uso de múltiplas threads pode diminuir o tempo total de execução.

Por exemplo, ao processar grandes volumes de dados, como nos casos em que temos várias cidades com registros de temperatura, o uso de threads permite que diferentes partes do processamento ocorram paralelamente. Isso é especialmente vantajoso em máquinas com múltiplos núcleos de processamento, pois vários threads podem ser executados ao mesmo tempo em núcleos diferentes.

Entretanto, é importante notar que nem sempre o aumento do número de threads resulta em melhorias na performance. O custo de criar e gerenciar muitos threads pode causar overhead, principalmente se o número de threads exceder o número de núcleos

disponíveis. Em sistemas com muitos threads, pode ocorrer o fenômeno de "contento", onde os threads competem pelos mesmos recursos, diminuindo a eficiência global.

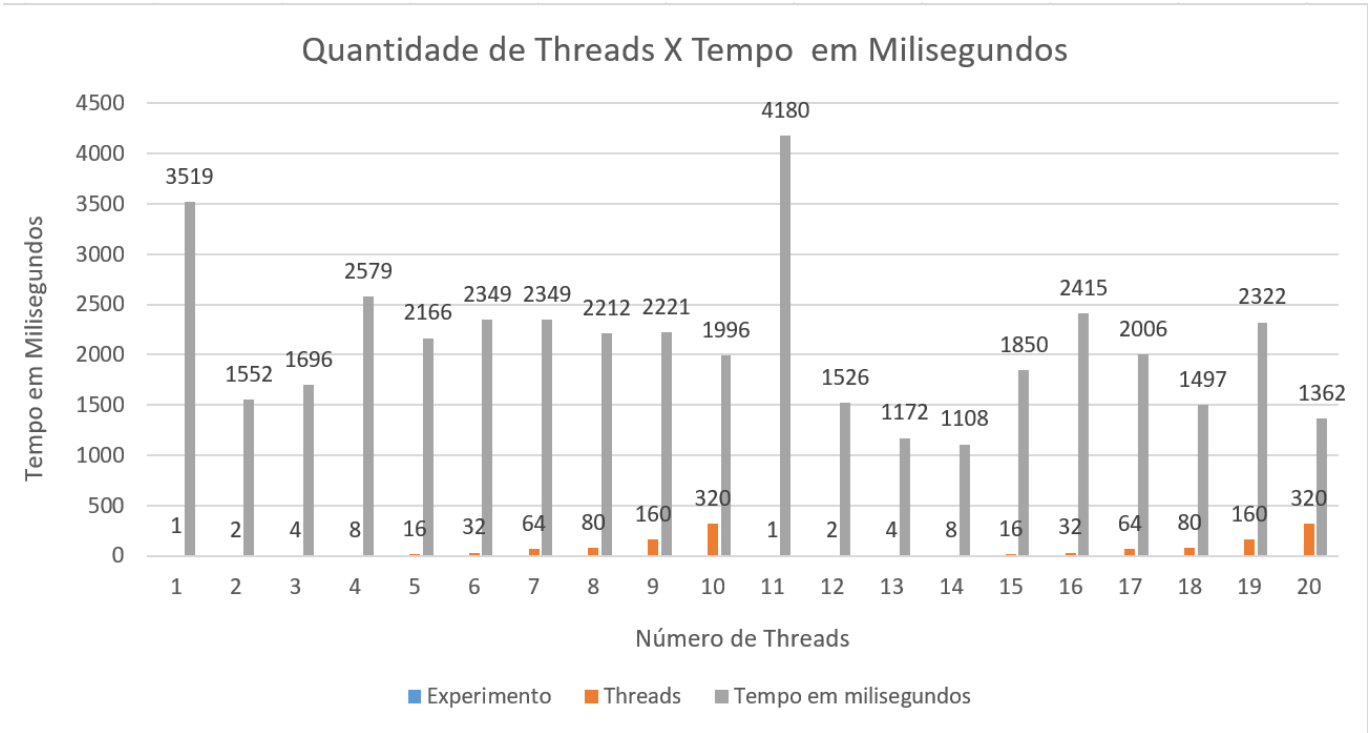
Qual a relação entre os modelos de computação concorrente e paralelo e a performance dos algoritmos?

A computação concorrente e a computação paralela estão intimamente relacionadas, mas não são exatamente a mesma coisa. A concorrência se refere à execução de várias tarefas de forma intercalada, onde o sistema alterna rapidamente entre elas. Já a computação paralela implica que múltiplas tarefas estão sendo executadas ao mesmo tempo, efetivamente, em diferentes núcleos de processamento.

Nos algoritmos, a computação paralela tende a proporcionar melhorias de performance significativas, já que múltiplas tarefas podem ser processadas simultaneamente. A computação concorrente, por outro lado, se destaca em cenários onde múltiplas tarefas independentes podem ser gerenciadas sem a necessidade de paralelismo total, como em aplicações de rede ou interface gráfica, onde as operações são intercaladas para evitar que o sistema fique inativo.

Em resumo, a computação concorrente permite a alternância entre várias tarefas, enquanto a computação paralela utiliza a capacidade de múltiplos núcleos para processar várias tarefas ao mesmo tempo. A escolha entre esses modelos afeta diretamente a performance dos algoritmos, dependendo da natureza da tarefa e dos recursos disponíveis.

Experimento:



A partir do experimeto acima, conclui-se que o aumento de threads não é linear: A expectativa de que mais threads resultem em maior desempenho não é garantida, especialmente ao passar de um certo número de threads, que geralmente está relacionado ao número de núcleos disponíveis no seu processador.

Quando você usa mais threads do que os núcleos disponíveis, o sistema operacional precisa alternar entre as threads, o que adiciona overhead (custo adicional) em termos de context switching. Limitação de I/O (leitura de arquivos): Embora você esteja paralelizando a leitura de arquivos, o desempenho de leitura/escrita no disco é limitado pela velocidade do disco.

Mesmo com muitas threads, se todas estiverem tentando ler arquivos ao mesmo tempo, elas estarão competindo pelos mesmos recursos de I/O, causando um gargalo. Sobrecarga de gerenciamento de threads: Com 320 threads, o sistema operacional gasta mais tempo gerenciando o ciclo de vida dessas threads (inicialização, agendamento, e troca de contexto) do que em execução real. Isso é particularmente notável quando o número de threads excede significativamente os núcleos da CPU disponíveis.

Referências Bibliográficas

- Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems*. Pearson.
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts*. Wiley.
- Downey, A. B. (2008). *The Little Book of Semaphores*. Green Tea Press.