Católica
Universidade Católica de Brasília

Avaliação substitutiva - Programação Concorrente e Distribuída

Aluno: Nickolas Carvalho de Azevedo
Matrícula: UC22100480
Curso: Ciência da Computação

O que são threads

Threads são unidades de execução dentro de um processo. Em um programa, um processo pode ser dividido em várias threads, permitindo a execução de múltiplas tarefas simultaneamente no mesmo processo. Cada thread possui seu próprio conjunto de instruções e pode executar de forma independente, compartilhando recursos como memória e arquivos com outras threads do mesmo processo.

De acordo com OAKS e WONG (1999), historicamente, o uso de threads surgiu pela primeira vez para facilitar a escrita de certos programas. Se um programa puder ser dividido em tarefas separadas, geralmente é mais fácil programar o algoritmo como tarefas ou threads independentes. Programas nessa categoria normalmente são especializados e lidam com múltiplas tarefas independentes. A relativa raridade desses tipos de programa torna o uso de threads nessa categoria uma habilidade especializada. Frequentemente, esses programas eram escritos como processos separados, utilizando ferramentas de comunicação dependentes do sistema operacional, como sinais e espaços de memória compartilhada, para se comunicarem entre si. Essa abordagem aumentava a complexidade do sistema.

Como threads funcionam computacionalmente

Computacionalmente, as threads são gerenciadas tanto pelo sistema operacional quanto pelo ambiente de execução da linguagem - no caso de Java, a JVM - Java Virtual Machine. Quando um programa é iniciado, ele possui pelo menos uma thread principal. Threads adicionais podem ser criadas para executar tarefas paralelas. O sistema operacional alterna a execução das threads usando técnicas como divisão de tempo (time-slicing) ou preempção para garantir que cada thread receba tempo de CPU. As threads compartilham o mesmo espaço de memória, o que facilita a comunicação entre elas, mas também pode levar a problemas de concorrência, como condições de corrida e deadlocks.

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

O uso de threads pode influenciar o tempo de execução de um algoritmo de várias maneiras:

- **Melhoria do Desempenho:** Threads podem melhorar o desempenho de um algoritmo ao permitir a execução paralela de tarefas, especialmente em sistemas multicore. Por

exemplo, um algoritmo que processa grandes volumes de dados pode ser dividido em partes menores, cada uma processada por uma thread diferente.

- **Concorrência:** Threads permitem que um programa responda a múltiplos eventos simultaneamente, como em aplicações web ou interfaces gráficas, onde a interação com o usuário e o processamento de dados ocorrem ao mesmo tempo.
- **Sobrecarga:** No entanto, criar e gerenciar threads envolve uma sobrecarga computacional. O tempo de mudança de contexto (context switching), sincronização e comunicação entre threads pode introduzir latência, especialmente se não for gerido adequadamente.

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

- **Computação Concorrente:** Na computação concorrente, múltiplas tarefas são progressivamente executadas, possivelmente ao mesmo tempo, mas não necessariamente simultaneamente. A concorrência está mais relacionada à estruturação de programas para lidar com múltiplas atividades simultâneas, podendo melhorar a responsividade e a eficiência.
- **Computação Paralela:** Na computação paralela, múltiplas tarefas são executadas exatamente ao mesmo tempo, aproveitando múltiplos processadores ou núcleos. A paralelização pode aumentar significativamente o desempenho de algoritmos que podem ser divididos em sub-tarefas independentes.

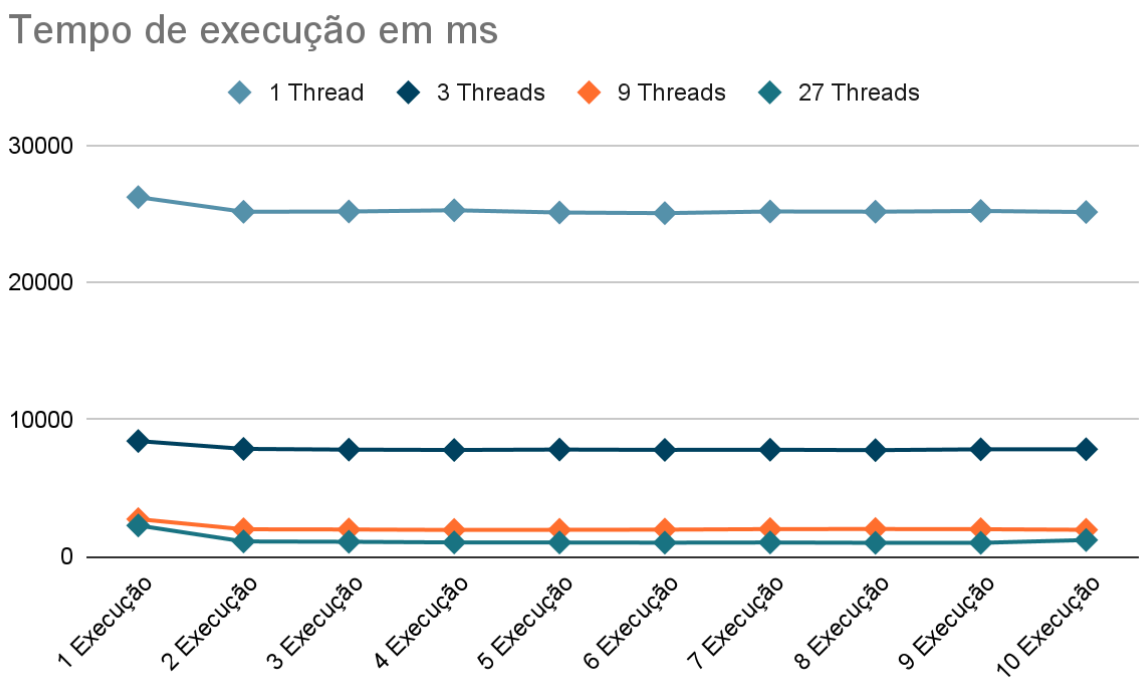
A relação entre esses modelos e a performance dos algoritmos é que a concorrência pode melhorar a eficiência e a responsividade de programas, especialmente em sistemas onde múltiplas tarefas dependem de recursos compartilhados. A paralelização, por outro lado, pode aumentar a velocidade de execução de algoritmos computacionalmente intensivos ao aproveitar a capacidade de processamento de múltiplos núcleos. Ambos os modelos requerem técnicas adequadas de sincronização e comunicação para evitar problemas como condições de corrida e deadlocks, que podem degradar a performance e a confiabilidade dos programas.

Comparação dos Tempos de Execução

Os resultados obtidos mostram como o tempo de execução varia com o aumento do número de threads utilizados. Abaixo está um resumo dos tempos de execução médios, mínimos e máximos para cada configuração de threads:

1. **Versão de referência, sem uso de threads:**
    - **Tempo médio de execução:** 25259.7 ms
    - **Tempo mínimo de execução:** 25048 ms
    - **Tempo máximo de execução:** 26217 ms
2. **Versão com 3 threads:**
    - **Tempo médio de execução:** 7860.6 ms
    - **Tempo mínimo de execução:** 7758 ms
    - **Tempo máximo de execução:** 8423 ms
3. **Versão com 9 threads:**
    - **Tempo médio de execução:** 2046.7 ms
    - **Tempo mínimo de execução:** 1936 ms
    - **Tempo máximo de execução:** 2728 ms
4. **Versão com 27 threads:**
    - **Tempo médio de execução:** 1170.6 ms
    - **Tempo mínimo de execução:** 995 ms
    - **Tempo máximo de execução:** 2263 ms

Gráfico Comparativo



Análise dos Resultados

A análise dos resultados revela algumas tendências importantes sobre o impacto do uso de threads no tempo de execução de um algoritmo:

- **Redução do Tempo de Execução:** Conforme o número de threads aumenta, há uma redução significativa no tempo médio de execução do algoritmo. Comparando a execução com uma única thread (25259.7 ms) com a execução com 27 threads (1170.6 ms), observamos uma redução drástica no tempo de execução. Isso demonstra o benefício do paralelismo, onde múltiplas threads permitem a divisão do trabalho em partes menores que podem ser processadas simultaneamente.
- **Eficiência do Paralelismo:** A eficiência do paralelismo é evidente quando comparamos os tempos médios de execução. A mudança de uma única thread para 3 threads já resulta em uma grande melhoria (de 25259.7 ms para 7860.6 ms). A eficiência continua a aumentar com 9 threads (2046.7 ms) e atinge um ponto ótimo em 27 threads (1170.6 ms). No entanto, a redução do tempo de execução tende a diminuir à medida que mais threads são adicionadas, indicando um ponto de saturação além do qual adicionar mais threads não proporciona melhorias proporcionais.
- **Sobrecarga de Gerenciamento de Threads:** Embora o tempo médio de execução diminua com o aumento do número de threads, a variabilidade nos tempos de execução (diferença entre o tempo mínimo e máximo) também aumenta. Isso é mais evidente em 9 threads e 27 threads, onde o tempo máximo de execução apresenta um desvio significativo em relação ao tempo médio. Essa variabilidade pode ser atribuída à sobrecarga de gerenciamento de múltiplas threads e ao tempo de mudança de contexto (context switching).
- **Estabilidade dos Tempos de Execução:** Em configurações de 3 threads e 9 threads, a variação entre os tempos mínimos e máximos é relativamente menor, indicando uma execução mais estável. No entanto, em 27 threads, embora o tempo médio de execução seja o menor, a variação é maior (tempo máximo de 2263 ms), o que pode ser resultado da maior sobrecarga de gerenciamento de threads.

Conclusão

Em resumo, observa-se que o uso de threads pode melhorar significativamente o tempo de execução de algoritmos, especialmente em sistemas multicore. No entanto, há um equilíbrio a ser atingido entre o número de threads e a sobrecarga de gerenciamento. A utilização de um número ótimo de threads (neste caso, entre 9 e 27) proporciona uma

melhoria considerável no desempenho, mas é essencial monitorar a variabilidade dos tempos de execução para evitar problemas de desempenho devido à sobrecarga de gerenciamento.

Bibliografia

1. OAKS, Scott; WONG, Henry. Java Threads. 3. ed. Sebastopol: O'Reilly, 1999.
2. OPEN-METEO. Historical Weather API. Disponível em: https://open-meteo.com/en/docs/historical-forecast-api#start_date=2024-01-01&end_date=2024-01-31&hourly=temperature_2m&daily=temperature_2m_max,temperature_2m_min&timezone=America%2FSao_Paulo. Acesso em: 27 jun. 2024.