Pró-Reitoria Acadêmica
Curso de Ciência da Computação
Trabalho de Programação Concorrente e distribuída

Atividade prática coletiva

Autores:
Caio Vinícius Rodrigues Moreira
Gabriel de Paula Alvarenga Adeodato
Lucas Sabino Assis

Orientador(a): Prof. João Robson Santos Martins

Brasília – DF
2024

Threads são unidades de execução dentro de um processo que permitem a execução concorrente de tarefas, oferecendo um fluxo de controle independente. Segundo Higor, threads permitem que um aplicativo opere como se tivesse várias partes atuando em paralelo, aumentando a eficiência e a responsividade do sistema. Renato Ramos da Silva e Roberto Sadao Yokoyama definem uma thread como um fluxo de controle dentro de um processo, onde "linhas de execução concorrem, dividindo o mesmo contexto". Isso significa que threads compartilham o mesmo espaço de endereçamento, o que facilita a comunicação entre elas e reduz o custo de criação, tornando-as uma solução ideal para aproveitar melhor os recursos em sistemas multiprocessados.

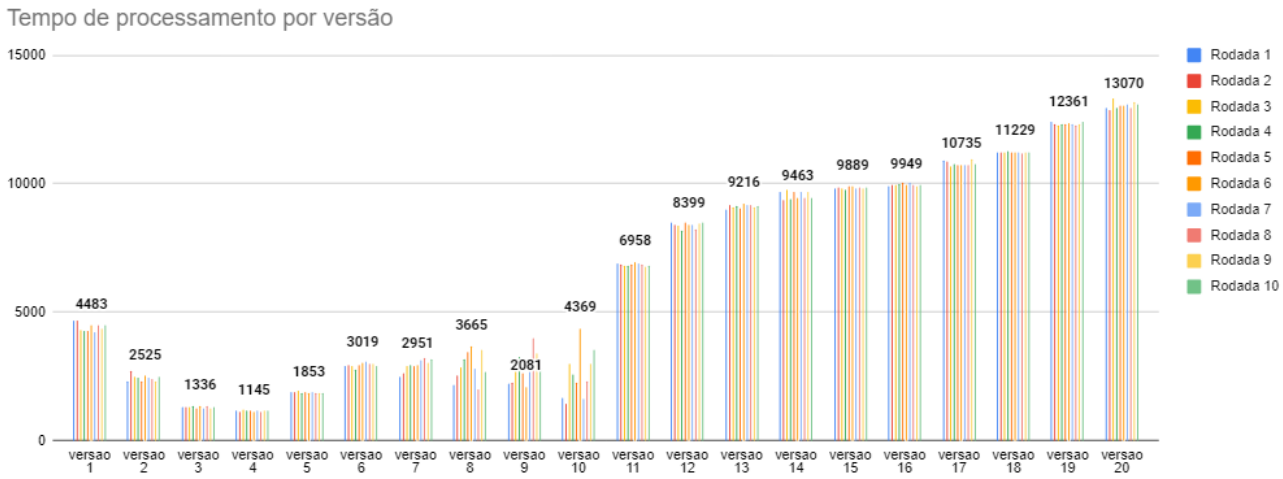
Quando divididas em multi threads, elas podem executar tarefas de forma concorrente ou sequencial, dependendo do comando utilizado, do software em teste e da maneira como o sistema operacional distribui e gerencia esses processos. Isso permite maior flexibilidade e desempenho, aproveitando melhor os recursos disponíveis.

O uso de threads pode reduzir o tempo de execução de um algoritmo ao permitir a execução simultânea de tarefas que seriam processadas sequencialmente. Quando um algoritmo é dividido em sub-tarefas que podem ser executadas em paralelo, o tempo total para completar o trabalho pode diminuir, especialmente em sistemas com múltiplos núcleos de CPU. Segundo Stallings (2021), essa concorrência permite que as threads sejam escalonadas em diferentes núcleos, maximizando o uso do hardware.

No entanto, é importante considerar que o gerenciamento de threads pode introduzir uma sobrecarga que afeta o desempenho. Essa sobrecarga se refere ao tempo e recursos necessários para criar, destruir e sincronizar threads. Se muitas threads forem criadas ou se a comunicação entre elas for excessiva, isso pode levar a um aumento no tempo de espera e à contenção de recursos, resultando em um desempenho pior do que o esperado.

A computação concorrente e paralela impacta a performance dos algoritmos de forma diferente. Enquanto a concorrente alterna a execução de várias tarefas em um único núcleo e está mais ligada a linguagens de alto nível, a paralela divide essas tarefas para serem processadas simultaneamente em múltiplos núcleos e geralmente está necessita de linguagens

com suporte explícito para programação paralela , como destacado pelo professor Dr. Leandro Franco.



O gráfico acima mostra os resultados dos experimentos realizados em uma máquina com Windows 11, processador Intel i9-9900 e 32 GB de RAM, onde o eixo X representa as 20 versões de teste e o eixo Y o tempo médio para a execução de cada versão.

O gráfico apresenta algo interessante: em um cenário ideal, onde o usuário possui 320 núcleos de processador e pode alocar cada thread em um núcleo, o melhor tempo deveria ser da versão 10. No entanto, isso não acontece. Nas primeiras versões, o tempo de execução diminui gradativamente, mas a partir da versão 5, com 16 threads, o desempenho começa a variar bastante, trazendo imprevisibilidade ao tempo de execução, já que a cada rodada, até a versão 10, o tempo se altera cada vez mais.

A partir da versão 11, um novo elemento foi adicionado ao código, as subthreads. O que inicialmente parecia bom para a performance, após análise, mostrou o efeito contrário, pois esse aumento no número de threads e subthreads gerou uma sobrecarga de gerenciamento, comprometendo o desempenho. O experimento revela que o limite de eficiência do código é físico, pois, mesmo com menor quantidade de threads, as versões iniciais se mostraram mais eficientes. Vale destacar que, embora a versão 10 tenha apresentado um bom tempo em algumas execuções, ao submeter o código a mais rodadas de testes, ela se mostrou inconstante, com tempos de execução muito altos em alguns casos.

Esse experimento revela que o limite de eficiência do código é físico, pois mesmo que os testes iniciais possuíssem menor quantidade de threads eles se mostraram os mais eficientes ao longo do experimento e vale constar que mesmo a versão 10 estando com um bom tempo nesse recorte do experimento, ao colocar o código em mais rodadas de testes, ele se mostra extremamente inconstante, podendo chegar a valores bem maiores.

SILVA, R. R. da; YOKOYAMA, R. S. Avaliação do Desempenho da Utilização de Threads em user level em Linux. **Revista de Informática Teórica e Aplicada**, *[S. l.]*, v. 18, n. 1, p. 112–132, 2011. DOI: 10.22456/2175-2745.12643. Disponível em: https://seer.ufrgs.br/index.php/rita/article/view/rita_v18_n1_p112. Acesso em: 21 sep. 2024.

HIGOR. Programação com threads. DevMedia, 2007. Disponível em: https://www.devmedia.com.br/programacao-com-threads/6152. Acesso em: 21 set. 2024.

SOUZA, Leandro Franco de. Introdução ao paralelismo. Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2009. Disponível em: https://sites.icmc.usp.br/lefraso/parallel.html. Acesso em: 21 set. 2024.

MICROSOFT. *Threads and Threading*. Disponível em: https://learn.microsoft.com/pt-br/dotnet/standard/threading/threads-and-threading. Acesso em: 22 set. 2024.