Universidade Católica de Brasília

Pró-Reitoria Acadêmica
Curso de Ciência da Computação

AT2/N1 - Relatório

Autores: Guilherme Bemvindo Santana Martins,
Matheus Anderson de Sousa Gomes,
Rafael Alves de Araujo,
Scarllet Gomes Rodrigues,

Orientador: João Robson

Brasília – DF
2023

Sumário

○ **O que são *threads***

Para entendermos o conceito de *Threads* primeiramente precisamos nos familiarizar com o conceito de processo, segundo Tanenbaum, (2015, P. 59) "[...]processo: uma abstração de um programa em execução.". Uma das mais antigas e significativas abstrações que os sistemas operacionais oferecem é o processo. Os computadores modernos usualmente executam diversas tarefas ao mesmo tempo, eles transformam uma única CPU em múltiplas CPUs virtuais, permitindo assim operações (pseudo) concorrentes mesmo quando há apenas uma CPU disponível. Um *thread* é um fluxo de controle independente executado no mesmo espaço de endereço que outros fluxos de controle independentes dentro de um processo. Conforme Silberschatz, (2000, P. 82) "Um thread, às vezes chamado de processo leve (lightweighl ftrocess), é uma unidade básica de utilização de CPU; [...]".

○ **Como *threads* funcionam computacionalmente:**

Os *threads* funcionam compartilhando o esboço de memória e recursos do processo que as criou. Elas permitem que múltiplas tarefas sejam executadas simultaneamente dentro do mesmo processo, o que facilita a troca de informações entre elas. No entanto, como os *threads* de um processo acessam a mesma memória, é necessário, um controle rigoroso para evitar erros como condições de corrida, onde vários *threads* tentam acessar ou modificar a mesma variável ao mesmo tempo. Para evitar esses problemas, são usados mecanismos de sincronização, como *locks* e semáforos, que ajudam a controlar o acesso à memória compartilhada (IBM, 2023).

○ **Como o uso de *threads* pode afetar o tempo de execução de um algoritmo:**

O uso de *threads* pode acelerar a execução de algoritmos, dividindo o trabalho em tarefas menores e independentes que podem ser processadas paralelamente em diferentes núcleos da CPU. Por exemplo, enquanto um thread espera por uma operação de entrada/saída (I/O), outra pode continuar processando dados. No entanto, o uso de *threads* também traz desafios, como o *overhead*, caudado pela necessidade de sincronização entre elas e a troca constante de contexto, ou o *deadlock*, causado quando *threads* ficam presas em um ciclo de dependência em que cada uma está esperando que a outra libere um recurso para poder continuar e como nenhuma dos threads pode prosseguir, todas ficam paradas indefinidamente. Se essas questões não forem gerenciadas corretamente, o uso de *threads* pode, na verdade, aumentar o tempo de execução em vez de reduzi-lo (Tanembaum, 2008).

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

Na computação concorrente, múltiplas tarefas (*threads* ou processos) são executadas "simultaneamente", mas na realidade, elas alternam entre si em um único núcleo de CPU, o que melhora a responsividade do sistema sem necessariamente diminuir o tempo total de execução. Já na computação paralela, essas tarefas são realmente executadas ao mesmo tempo, pois utilizam múltiplos núcleos da CPU ou várias máquinas em paralelo. Isso pode melhorar significativamente o desempenho de algoritmos, especialmente em sistemas com muitos núcleos de processamento. Contudo, para obter ganhos reais em performance, é importante garantir que as tarefas estejam bem balanceadas entre as *threads* ou processos. Se um thread estiver sobrecarregado enquanto outras ficam ociosas, o desempenho geral será prejudicado (Silberschatz, 2000)

○ **Exibição e explicação dos resultados obtidos**

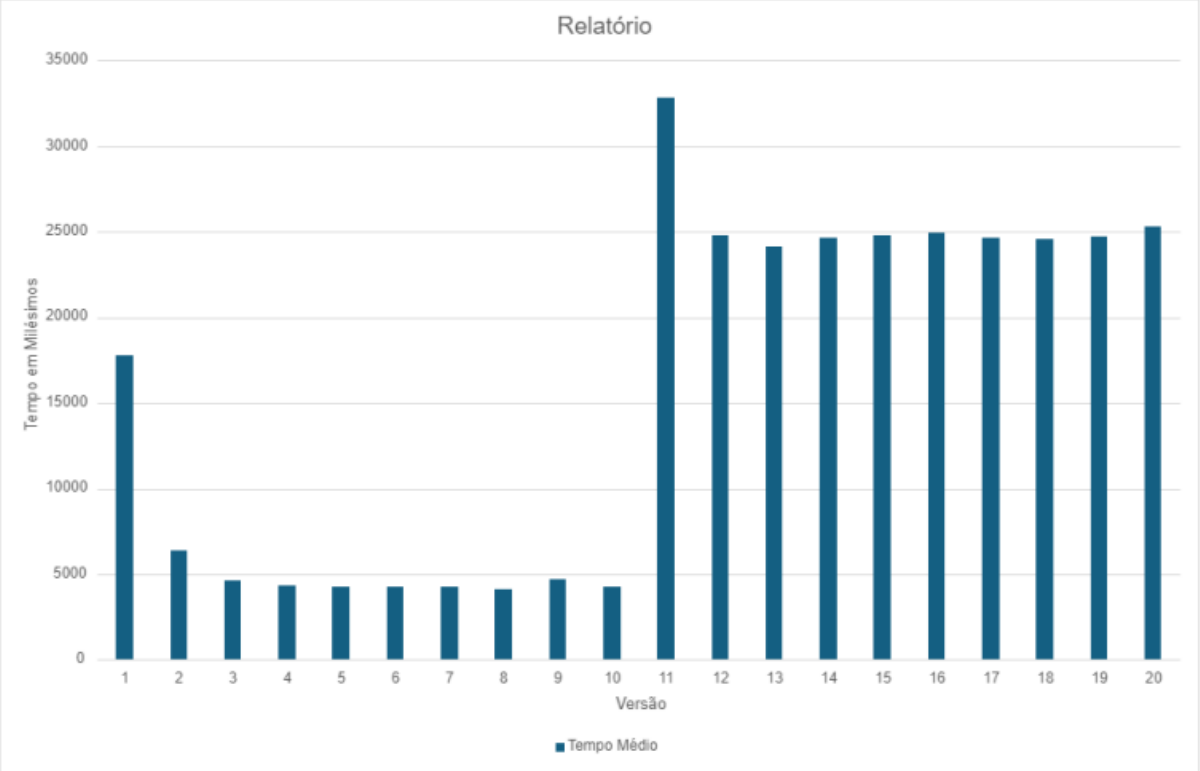
No experimento, analisamos 20 versões de execução, variando o número de *threads* e a forma como as tarefas foram distribuídas. O objetivo era avaliar o impacto dessas configurações no tempo de execução. A versão mais rápida foi a versão 8, que utilizou 80 *threads*, com cada uma responsável por processar 4 cidades, sem *threads* específicas para os anos. Essa abordagem, das versões de 2 a 10 distribuíram bem o trabalho entre os threads, resultando em um tempo de execução otimizado.

Por outro lado, a versão 11 foi a mais lenta, onde nenhum thread foi alocado para processar cidades, e cada *thread* era responsável apenas pelos anos individuais. Isso causou um aumento significativo no tempo de execução, provavelmente devido ao overhead na criação e sincronização de muitos threads para cada ano.

Curiosamente, esperávamos que as versões de 11 a 20 fossem as mais rápidas, pois aumentamos o número de *threads* e dividimos as tarefas entre anos, o que deveria permitir um processamento paralelo mais eficiente. No entanto, os resultados indicam que provavelmente não organizamos os threads de forma eficaz, resultando em um desempenho abaixo do esperado. Isso sugere que a forma como dividimos o trabalho entre os threads, principalmente no processamento por anos, não foi ideal, levando a um uso ineficiente dos recursos.

O teste foi executado em um computador rodando Windows 11, utilizando Java 22 e com um processador de 4 núcleos.

Gráfico Comparativo:



O gráfico destaca que as versões com um balanceamento adequado de *threads*, como a versão 8, obtiveram tempos de execução significativamente menores. Por outro lado, nas versões de 11 a 20, onde a organização dos threads foi menos eficiente, o tempo de execução aumentou.

Esses resultados mostram que, embora o número de *threads* seja importante, a maneira como as tarefas são organizadas entre elas é fundamental para otimizar a performance. Uma divisão mal planejada pode gerar overhead e reduzir o ganho esperado do paralelismo.

○ **Referencias**:

- IBM. Understanding threads and processes. 2023. Disponível em: https://www.ibm.com/docs/pt-br/aix/7.3?topic=programming-understanding-threads-processes. Acesso em: 22 set. 2024.
- SILBERSCHATZ, Abraham. *Sistemas Operacionais - Conceitos e Aplicações*. Rio de Janeiro: LTC, 2000. Disponível em: https://www.academia.edu/40441882/SILBERSCHATZ_Sistemas_Operacionais_Conceitos_e_Aplicacoes_pdf. Acesso em: 22 set. 2024.
- TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 4. ed. São Paulo: Pearson Education, 2015. Disponível em: https://archive.org/details/SistemasOperacionaisModernosTanenbaum4Edio/page/n85/mode/1up?view=theater. Acesso em: 22 set. 2024.