UNIVERSIDADE CATÓLICA DE BRASÍLIA

PRÓ-REITORIA ACADÊMICA

ESCOLA DE EDUCAÇÃO,TECNOLOGIA E COMUNICAÇÃO

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE PROGRAMAÇÃO CONCORRENTE DISTRIBUÍDA

ARTHUR LOPES MOURA-UC20190001666

PEDRO XAVIER LODÔNIO-UC21200125

YGOR MACHADO GONÇALVES DE OLIVEIRA-UC18200363

RELATÓRIO:ATIVIDADE PRÁTICA COLETIVA

BRASÍLIA-DF

2024

SUMÁRIO

1 INTRODUÇÃO

A atividade proposta tem como objetivo principal a comparação da capacidade de leitura de arquivos de um algorítimo a partir da adição de *threads.* Nesse sentido, é necessário elaboração de um projeto em Java que leia 320 arquivos no formato Comma-Separated Values (CSV), valores separados por vírgula -em tradução livre-, os quais contém registros de temperaturas diárias de 320 cidades diferentes. A partir dessa leitura , o programa deve ser capaz de calcular as temperaturas médias,mínimas e máximas de cada mês,para cada cidade. Ademais, para atingir o objetivo principal da atividade, foram realizados 20 experimentos, os quais se diferem em relação ao número de *threads* utilizados.

2 THREADS

Thread é o fluxo sequencial de execução por qual um programa percorre. “Os sistemas operacionais modernos possuem o conceito de processos que , de forma simplicada, são programas diferentes e independentes executados pelo sistema operacional” (Cordeiro,2004), a citação é importante para a compreensão do conceito de thread, visto que ,ainda segundo Cordeiro, as *threads* são classificadas como “processos leves”. Essa classificação deve-se ao fato de que, assim como os processos, as *threads* são independentes, possuem pilha de execução própria, *program counter* próprio e também suas variáveis locais. A vantagem do uso de *threads* está na capacidade da coexistência de diversas atividades em um mesmo processo, aumentando assim o desempenho.

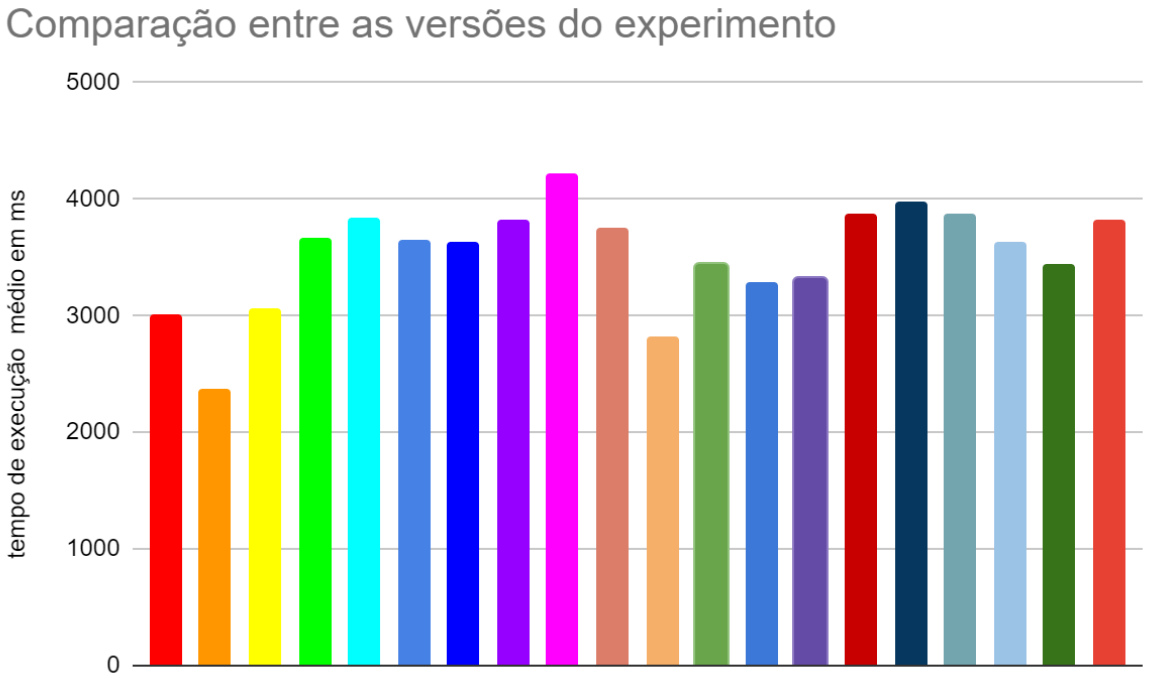
Segundo Tanenbaum e BOS (2015), o uso de *threads* é mais eficiente em sistemas com múltiplos núcleos de CPU, uma vez que é possível a execução em paralelo, ou seja, o uso de *threads* habilita o sistema, com multiplos nucleos de CPU, a executar diversas tarefas ao mesmo tempo. O sistema operacional fica responsável pela distribuição da execução das *threads* entre os núcleos e alterna entre elas mediante o *context switching* .Consequente a isso, as tarefas são realizadas com eficiência e aumentam o desempenho, principalmente nas que podem ser desmembradas e executadas em paralelo.

Ademais, é de suma importância diferenciar os modelos de computação concorrente e paralela, tais diferenças são discutidas por Silberschatz, Galvin e Gagne no livro “Operating Systems Concepts”. Os autores descrevem que em um sistema com um núcleo único a execução de *threads* será intercalada, já que o sistema consegue executar apenas uma *thread* por vez (concorrente). Por outro lado , em um sistema com mais de um núcleo , o sistema consegue processar algumas *threads* em paralelo, já que é possível atribuir uma thread a um núcleo específico (paralelo).

Portanto, *threads* são unidades básicas de execução dentro de um processo e podem ser consideradas processos leves por compartilharem características como sua independência,sua pilha de execução própia e suas variáveis locais. Além disso, algoritmos que utilizam *threads* conseguem executar tarefas em paralelo o que otimiza o tempo de execução e aumenta o desempenho . Cabe ressaltar que o uso de *threads* é mais eficiente em sistemas com vários núcleos , os quais permitem a execução paralela - na qual cada *thread* é atribuida a um núcleo. Por fim , no modelo de execução concorrente a execução das *threads* ocorre de maneira intercalada , devido a falta de nucleos para a reatribução de *threads* - o que impossibilita as atividades de serem realizadas de maneira simultânea e paralela.

3 RESULTADOS DO EXPERIMENTO

Com a intenção de comparar a performance do algorítimo com a implementação de *threads* foram desenvolvidas 20 versões diferentes , cada uma com uma quantidade diferente de *threads*. O gráfico a seguir mostra a diferença do tempo de execução dos experimentos. Deve ser levado em consideração que nos experimentos de 1-10 as *threads* foram utilizadas somente para leitura de cidades e a partir do experimento 11 foram utilizadas threads para o processamento de dados de cada ano registrado. A comparação dos resultados sugere que, enquanto a utilização de *threads* pode melhorar a eficiência em tarefas de leitura, seu impacto em tarefas de processamento pode variar, dependendo da implementação e da natureza da carga de trabalho.



Quando a quantidade de *threads* ultrapassa significativamente o número de núcleos físicos e lógicos disponíveis (no caso, um processador com 6 núcleos físicos e capacidade de *hyper-threading*, resultando em 12 *threads* possíveis), o sistema tende a gastar um tempo excessivo alternando entre as *threads* e gerenciando-as. Essa sobrecarga é a explicação para o aumento do tempo de execução observado com 320 *threads*. Para um processador com 6 núcleos e 12 *threads* lógicas, o desempenho ideal é alcançado com um número de *threads* variando entre 6 e 12. Além disso, não se observam ganhos significativos além desse intervalo, e o desempenho começa a se deteriorar devido à sobrecarga causada pela alternância excessiva entre as *threads*.

A máquina utilizada para realização dos testes é um MacBook Pro, com processador Intel Core i7 6-Core 2.6ghz, placa gráfica AMD Radeon Pro 5300M 4GB VRAM e 16GB de memória RAM. O sistema operacional é o macOS Sonoma.

4 CONCLUSÃO

Quando se utiliza um número de *threads* muito maior do que a quantidade de núcleos disponíveis em um processador — por exemplo, um processador com 6 núcleos físicos e 12 *threads* lógicas — o sistema pode acabar gastando muito tempo alternando entre essas *thread*s e gerenciando-as. Isso resulta em um aumento do tempo de execução, já que essa sobrecarga prejudica a eficiência do processamento.

Para alcançar um desempenho ideal, é recomendado manter o número de *threads* entre 6 e 12. Essa faixa permite que o processador trabalhe de maneira mais eficaz, aproveitando ao máximo seus núcleos sem sofrer penalidades associadas à troca constante de contextos. Quando se ultrapassa esse número, os benefícios em termos de desempenho se tornam insignificantes, e a situação pode piorar, levando a uma queda no desempenho geral.

Portanto, é fundamental que, ao implementar o uso de *threads*, se leve em conta a arquitetura do processador e quantos núcleos ele possui. Usar muitas *threads* não só não melhora o desempenho, como pode acabar dificultando-o. Para otimizar o uso dos recursos do processador e garantir uma execução mais rápida e eficiente das tarefas, é essencial encontrar um equilíbrio entre a quantidade de *threads* e a capacidade do *hardware* disponível. Essa abordagem ajuda a maximizar a eficiência e a obter resultados melhores nas tarefas realizadas.

5 REFERÊNCIAS BIBLIOGRÁFICAS

CORDEIRO, Daniel de Angelis. Introdução ao Uso de Threads em Java. USP, [S. l.], p. 1-12, 26 mar. 2004. Disponível em: Introdução ao uso de Threads

TANENBAUM, Andrew S.; BOS, Hebert. THREADS. In: TANENBAUM, Andrew S.; BOS, Hebert. MODERN OPERATING SYSTEMS. [S. l.: s. n.], 2015. Disponível em: Andrew S. Tanenbaum - Modern Operating Systems. Acesso em: 20 set. 2024.

SILBERSCHATZ, ABRAHAM; GALVIN, PETER BAER; GAGNE, GREG. Operating Systems Concepts. [S. l.: s. n.], 2018. Disponível em: Operating Systems Concepts. Acesso em: 21 set. 2024.