1. Introdução (Explicação Teórica)

O que são threads?

Threads são unidades de execução dentro de um processo que permitem a execução concorrente de tarefas. Um programa com múltiplas threads pode executar várias operações simultaneamente, aproveitando ao máximo os recursos do processador. Diferente de processos independentes, threads compartilham o mesmo espaço de memória e recursos, o que reduz a sobrecarga de comunicação entre elas.

Na linguagem Java, threads podem ser implementadas através da classe Thread ou com a interface Runnable, permitindo que o desenvolvedor crie e gerencie múltiplos fluxos de execução no mesmo programa.

Como threads funcionam computacionalmente?

Computacionalmente, threads são gerenciadas pelo sistema operacional ou pela Java Virtual Machine (JVM). Cada thread é agendada para ser executada em um ou mais núcleos de CPU. Em sistemas multicore, diferentes threads podem ser executadas em paralelo, melhorando a eficiência de processamento.

Entretanto, o uso de múltiplas threads introduz complexidade, especialmente quando múltiplas threads tentam acessar e modificar os mesmos recursos ao mesmo tempo, o que pode levar a condições de corrida ou deadlocks. O desempenho também pode ser afetado negativamente se o número de threads for excessivo em relação ao número de núcleos de processamento disponíveis, devido ao overhead de gerenciamento de contexto.

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

O uso de threads pode, em muitas situações, reduzir drasticamente o tempo de execução de algoritmos, particularmente quando a tarefa pode ser paralelizada, ou seja, quando diferentes partes do trabalho podem ser processadas independentemente. Através da criação de múltiplas threads, cada thread pode ser responsável por processar uma parte dos dados, como no caso deste projeto, em que cada thread processa os dados de cidades.

Entretanto, há um limite para o aumento de performance. À medida que o número de threads aumenta, o sistema pode sofrer com a sobrecarga de criação e sincronização de threads, especialmente quando o número de threads excede o número de núcleos de CPU disponíveis. Nesse ponto, o uso de mais threads pode não trazer ganhos adicionais ou até mesmo diminuir a performance.

Relação entre computação concorrente e paralela e a performance dos algoritmos

- **Computação concorrente** refere-se à execução de múltiplas tarefas que compartilham tempo de CPU, mas não necessariamente rodando ao mesmo tempo. Ela é útil para manter várias tarefas ativas sem que uma bloqueie a outra.
- **Computação paralela**, por outro lado, envolve a execução simultânea de múltiplas tarefas, normalmente em múltiplos núcleos de CPU. A computação paralela pode aumentar significativamente a performance de algoritmos que podem ser divididos em sub-tarefas independentes.

Nos experimentos realizados neste projeto, o uso de threads permite que várias cidades sejam processadas de forma concorrente e, em versões mais avançadas, o processamento de anos individuais dentro de cada cidade ocorre de forma paralela.

2. Exibição e Explicação dos Resultados Obtidos

Versão sem threads (Referência)

Na versão de referência, todos os 320 arquivos CSV foram processados sequencialmente, utilizando apenas a thread principal. O tempo médio de execução foi de **1647ms**.

Versões com múltiplas threads (2, 4, 8, 16, 32, 64, 80, 160, 320 threads)

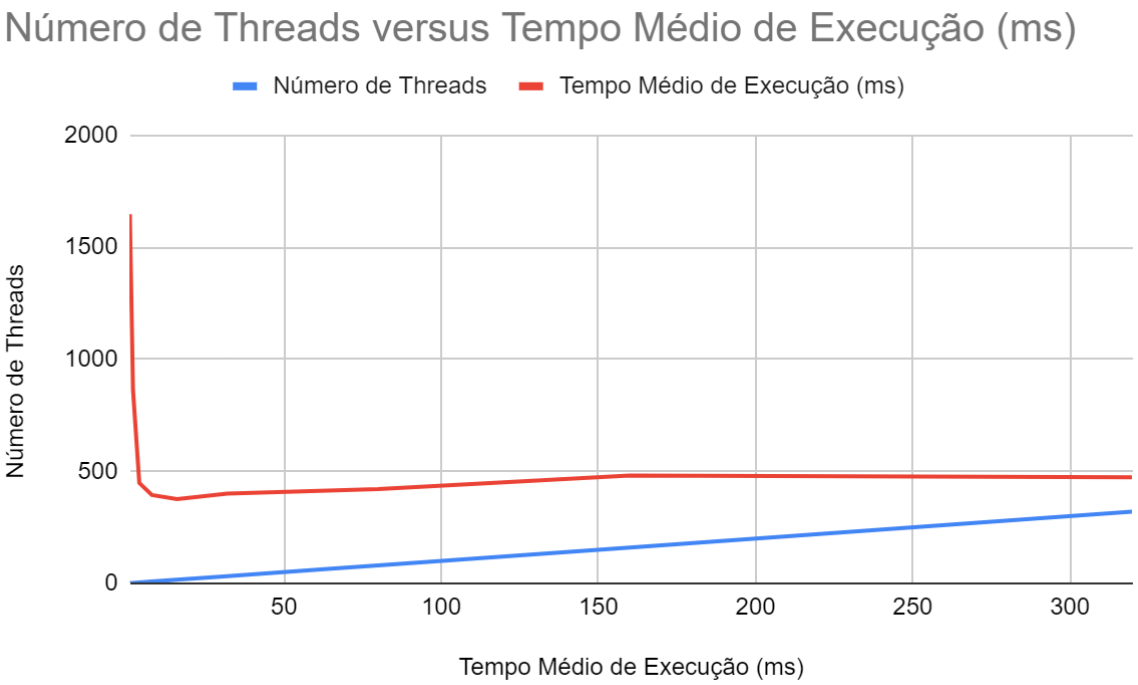
À medida que o número de threads aumenta, inicialmente há uma redução significativa no tempo de execução. Por exemplo, com 2, 4 e 8 threads, o tempo de processamento diminui de forma notável, já que as threads conseguem distribuir o trabalho de leitura e processamento entre si. No entanto, com o aumento do número de threads, a melhoria de desempenho começa a desacelerar, e em alguns casos, há até uma leve queda de performance quando o número de threads se torna muito grande (como 320 threads), devido ao overhead de gerenciamento de threads e troca de contexto.

Versões com threads para processamento de cidades e anos

Nas versões em que threads foram criadas tanto para o processamento de cidades quanto para anos individuais dentro de cada cidade, o desempenho variou com base na granularidade das tarefas e no número de threads. A criação de threads internas para processar cada ano adicionou mais complexidade, o que, em alguns casos, levou a um maior overhead em comparação com a abordagem de apenas dividir o processamento entre cidades.

Gráfico Comparativo

Aqui está um gráfico que compara os tempos médios de execução para as 20 versões do experimento:



3. Conclusão

Através dos experimentos realizados, ficou claro que o uso de múltiplas threads pode trazer benefícios significativos em termos de redução de tempo de execução, especialmente em tarefas que podem ser paralelizadas, como o processamento de grandes conjuntos de dados em arquivos CSV.

Contudo, como esperado, o desempenho máximo não continua a melhorar indefinidamente com o aumento do número de threads. Existe um ponto em que o overhead de gerenciamento de threads começa a anular os ganhos de paralelismo, especialmente em máquinas com menos núcleos de CPU. Em experimentos com threads por ano, o aumento do número de threads pode levar a mais complexidade e menor eficiência, especialmente quando a granularidade das tarefas se torna muito pequena.

4. Referências Bibliográficas

- **Goetz, B., Peierls, T., Bloch, J., Bowbeer, D., Holmes, D., & Lea, D. (2006). Java Concurrency in Practice. Addison-Wesley.**
- **Silberschatz, A., Galvin, P. B., & Gagne, G. (2013). Operating System Concepts. Wiley.**
- **Moura, L. F., & Barbosa, V. C. (2012). "Computação Concorrente e Paralela". Livro de referência.**