Relatório

Nome: João Vítor Rocha de Carvalho

RA: UC22201130

Threads são pequenas unidades de execução que podem ser programadas para rodar independentemente dentro de um processo maior. Nos sistemas operacionais atuais, um processo pode conter várias threads que dividem recursos como memória e espaço de endereçamento, porém possuem contextos de execução individuais. As threads possibilitam a execução simultânea de diversas partes de um programa, o que resulta em uma utilização mais eficiente das CPUs com múltiplos núcleos e melhora a capacidade de resposta em aplicações multitarefa.

Do ponto de vista **computacional**, as threads são controladas pelo sistema operacional e pelo ambiente de execução da linguagem de programação. Cada thread tem sua própria configuração de execução, que engloba um contador de programa, registros e uma pilha de execução. O agendamento das threads para rodarem nas CPUs disponíveis é feito pelo sistema operacional, o que pode resultar em situações de concorrência quando diversas threads tentam interagir com os mesmos recursos compartilhados.

O uso de threads pode impactar o tempo de execução de um algoritmo devido à capacidade de realizar múltiplas tarefas ao mesmo tempo. Ao adaptar um algoritmo para utilizar threads, partes independentes podem ser executadas simultaneamente em diferentes threads, aproveitando melhor o poder de processamento de CPUs multi-core e reduzindo o tempo total de execução, especialmente para algoritmos com partes paralelizáveis. No entanto, o benefício do uso de threads depende da natureza do algoritmo e do paralelismo possível de ser explorado. Algoritmos com muitas dependências entre partes podem não se beneficiar tanto do paralelismo, já que algumas partes podem precisar esperar pela conclusão de outras.

Os modelos de computação concorrente e paralela afetam diretamente a performance dos algoritmos em sistemas modernos com múltiplos núcleos de CPU.

A computação concorrente envolve a execução intercalada de múltiplas tarefas (threads), compartilhando recursos e escalonadas pelo sistema operacional. Algoritmos adaptados para concorrência podem melhorar sua performance ao permitir a execução

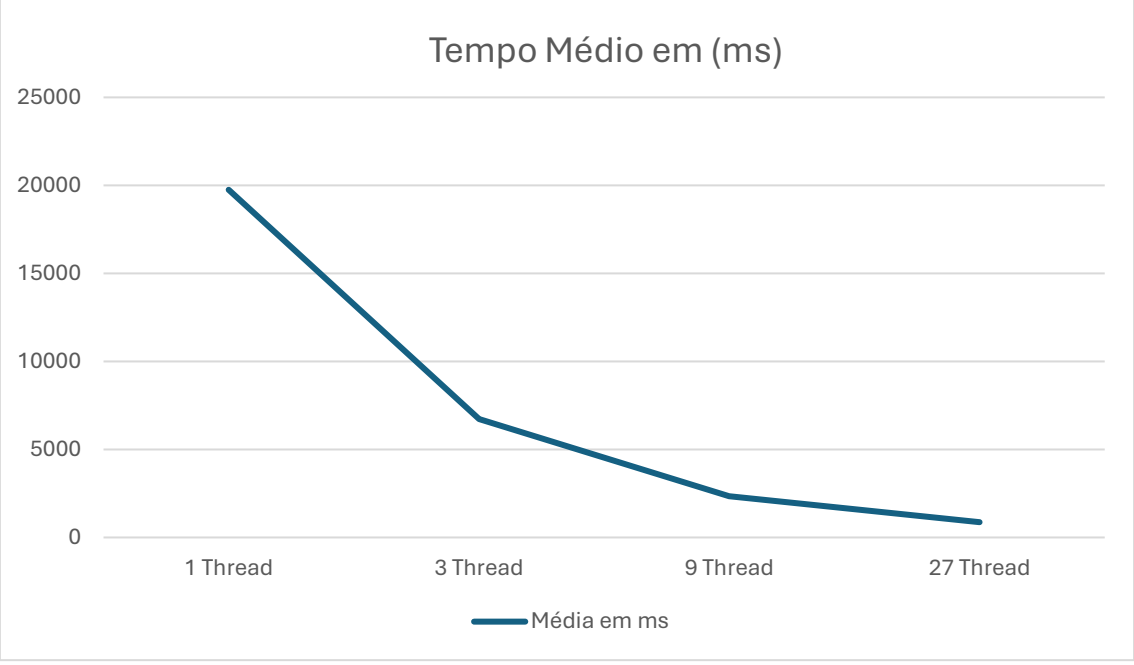
simultânea de partes independentes. Por outro lado, a **computação paralela** refere-se à execução simultânea real de múltiplas tarefas em diferentes núcleos de CPU. Algoritmos paralelizados podem alcançar significativo aumento de desempenho ao executar várias partes simultaneamente, sem dependências que as impeçam. A escolha entre os dois modelos depende da estrutura do problema e da arquitetura do hardware disponível. Algoritmos paralelizáveis eficientemente podem tirar proveito máximo dos CPUs multi-core, reduzindo o tempo de execução. Já algoritmos com muitas dependências sequenciais podem não se beneficiar tanto da paralelização, sendo mais indicados para computação concorrente ou sequencial.

Referências Bibliográficas

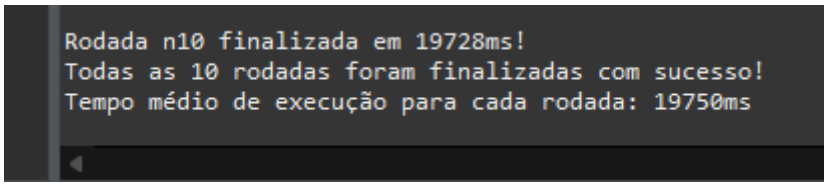
"Operating System Concepts" de Abraham Silberschatz

https://blog.pantuza.com/artigos/o-que-sao-e-como-funcionam-as-threads

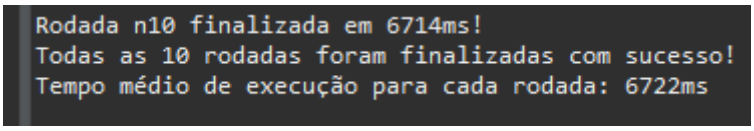
https://revista.grupointegrado.br/revista/index.php/campodigital/article/view/311/145



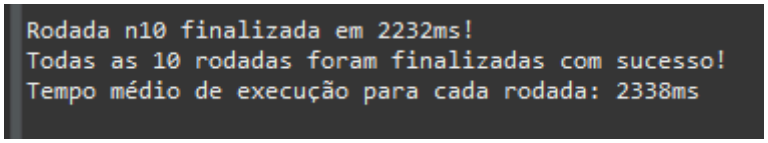
sem thread



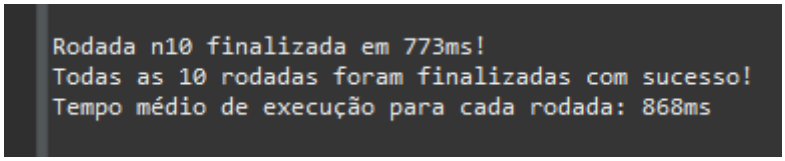
3 thread



9 thread



27 thread



Resultados

Após adquirir os resultados, claramente é perceptível a diferença de tempo em média de execução das threads, como podemos ver no gráfico e também nos prints acima (código rodando na minha máquina). No primeiro resultado tivemos um tempo em média de 19750ms e em quanto na de 27 tivemos um resultado de 868ms. Em resumo, o uso de threads em Java permite que múltiplas partes de um programa sejam executadas simultaneamente, melhorando o desempenho geral e reduzindo o tempo médio de execução.

Referências Bibliográficas para ajudar no Código

https://youtu.be/9oq7Y8n1t00

https://youtu.be/r_MbozD32eo