Universidade Católica de Brasília

Curso de Ciência da Computação

THREADS

Autor: Júlia Corrêa de Souza Oliveira - UC22200753
Professor: João Robson Santos Martins

Brasília - DF
2024

Threads em Java: Estrutura, Funcionamento e Impacto na Performance

As threads são unidades de execução dentro de um processo que compartilham o mesmo espaço de memória, permitindo que diferentes partes de um programa sejam executadas simultaneamente. Essa característica é fundamental em ambientes de programação modernos, como o Java, que oferece um suporte robusto ao uso de threads. Em um editor de texto, por exemplo, uma thread pode ser responsável pela formatação do texto enquanto outra cuida da impressão. Essa capacidade de realizar multitarefas não só aumenta a eficiência das aplicações, mas também melhora a experiência do usuário, tornando os programas mais responsivos.

No nível computacional, as threads são gerenciadas pelo sistema operacional, que aloca frações de tempo de CPU para cada uma delas. Em sistemas de um único núcleo, o sistema operacional alterna rapidamente entre as threads, criando a ilusão de que elas estão sendo executadas simultaneamente. Em contrapartida, em ambientes multicore, várias threads podem realmente ser processadas em paralelo, aproveitando melhor os recursos de hardware disponíveis. No entanto, esse gerenciamento requer um controle eficiente para garantir a integridade dos dados compartilhados, uma vez que as threads podem ser interrompidas a qualquer momento, levando a potenciais conflitos e inconsistências.

O uso de threads pode ter um impacto significativo no desempenho de algoritmos, especialmente aqueles que lidam com grandes volumes de dados ou que exigem computação intensiva. Ao permitir a execução simultânea de múltiplas tarefas, as aplicações podem utilizar o tempo ocioso dos recursos do sistema de maneira mais eficiente. Contudo, é importante destacar que a introdução de múltiplas threads também traz uma sobrecarga adicional, como a necessidade de sincronização e comunicação entre elas. Se essas operações não forem bem implementadas, podem reduzir os ganhos de desempenho esperados, resultando em latências maiores e competições por recursos do processador.

A computação concorrente refere-se à execução de várias tarefas de maneira intercalada, permitindo que partes do código sejam executadas de forma não sequencial. Já a computação paralela envolve a execução real de tarefas simultâneas em múltiplos núcleos de processamento. Essa distinção é crucial, pois enquanto a concorrência oferece a possibilidade de maior responsividade, o paralelismo pode resultar em ganhos de desempenho significativos, especialmente em tarefas que podem ser divididas em partes independentes.

Entretanto, nem todos os problemas são adequados para paralelização. Aqueles que possuem dependências entre etapas de execução podem não se beneficiar do uso de múltiplas threads. A escolha do modelo de computação mais apropriado deve ser baseada no comportamento do aplicativo, que pode variar durante a execução, e na configuração do sistema que o suporta. Assim,

um design adaptativo que ajuste dinamicamente a utilização de threads, seja virtuais ou de sistema, pode resultar em um desempenho superior e mais eficiente.

As threads são uma ferramenta poderosa na programação moderna, especialmente em Java, onde sua implementação é projetada para facilitar a criação de aplicações eficientes e responsivas. Ao compreender como as threads funcionam, seu impacto no desempenho dos algoritmos e a diferença entre computação concorrente e paralela, os desenvolvedores podem otimizar suas aplicações de maneira eficaz. Essa flexibilidade, quando combinada com um design adaptativo, é essencial para maximizar o uso dos recursos computacionais disponíveis, especialmente em um cenário onde os desafios de desempenho são cada vez mais complexos.

Análise Comparativa dos Resultados das 20 Versões do Experimento

O experimento realizado, que envolve o uso de threads para processar 320 arquivos CSV com dados de temperaturas diárias, nos fornece uma perspectiva clara sobre o impacto do uso de múltiplas threads no tempo de execução de algoritmos. O objetivo foi analisar como a variação no número de threads impacta o tempo de execução do processamento. Os resultados foram coletados em 10 rodadas para cada versão, com o tempo médio de execução sendo calculado para melhor comparação.

Análise dos Resultados:

Na **Versão 1**, onde não foi utilizado nenhum tipo de thread, o tempo médio de execução foi de 1228 ms. Essa versão serviu como base para as comparações posteriores. O uso de threads começou na **Versão 2**, onde a execução média caiu drasticamente para 643 ms, mostrando uma melhoria significativa e indicando que a paralelização inicial trouxe grandes benefícios em termos de desempenho.

A partir da **Versão 3**, a tendência de redução no tempo médio de execução continuou, alcançando 365 ms. Aqui, o uso de 4 threads demonstrou ser muito eficiente, o que sugere que a divisão de tarefas em múltiplos threads foi eficaz na exploração da capacidade de processamento.
Conforme avançamos para as **Versões 4 e 5**, os tempos médios de 282 ms e 277 ms, respectivamente, mostraram que o desempenho continuava a melhorar com o aumento do número de threads. Essas versões indicam que, até esse ponto, o processamento de dados se beneficiava claramente da concorrência.

Na **Versão 6**, o tempo médio de 270 ms reafirmou a eficácia da abordagem com threads, mas a queda no tempo começou a desacelerar. A **Versão 7** apresentou um tempo médio similar de 270 ms, indicando que o sistema já estava próximo de sua capacidade ideal de processamento.

Ao analisarmos as **Versões 8 e 9**, notamos uma leve piora no desempenho, com tempos médios de 469 ms e 466 ms, respectivamente. Esse aumento no tempo pode ser atribuído a uma sobrecarga no gerenciamento de threads, onde o sistema começou a enfrentar limitações em termos de gerenciamento de recursos. A **Versão 10**, que ainda utilizou o mesmo número de threads que as anteriores, resultou em um tempo médio de 1225 ms, semelhante ao da Versão 1, reforçando a ideia de que a eficiência do sistema pode ter atingido um teto.

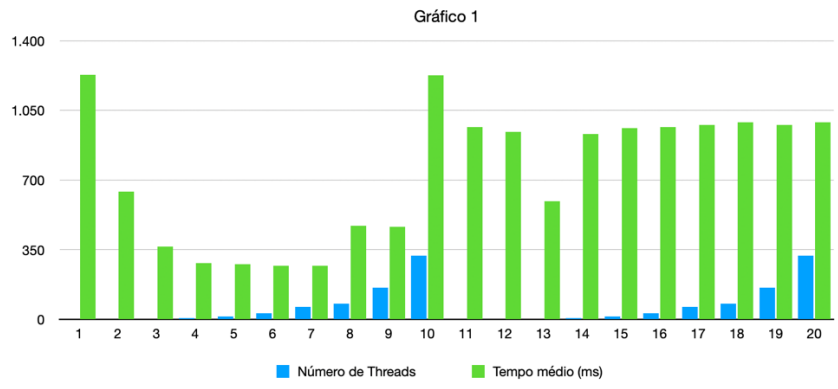
Nas **Versões 11 a 20**, onde foram aplicadas técnicas mais complexas de processamento com aumento gradual no número de threads, observamos um padrão de estabilização. Por exemplo, a **Versão 11** teve um tempo médio de 966 ms, enquanto a **Versão 12** registrou 942 ms. Contudo, nas versões subsequentes, os tempos médios começaram a variar menos, sugerindo que o sistema estava se aproximando de sua capacidade máxima. A **Versão 18** obteve um tempo médio de 989 ms, o que demonstra que, mesmo com um aumento no número de threads, a melhoria no desempenho tornou-se marginal.

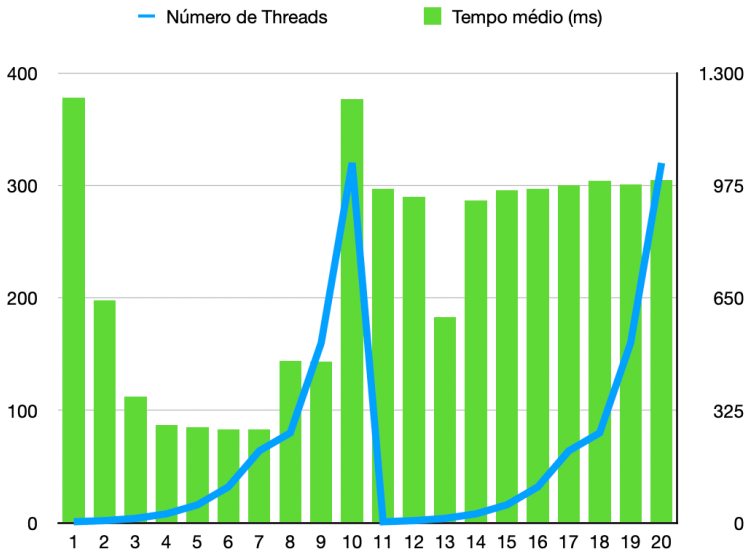
Por fim, a **Versão 20** apresentou um tempo médio de 990 ms, quase idêntico ao da Versão 18, o que reforça a conclusão de que, além de um certo ponto, a eficiência do uso de threads para processamento paralelo pode não resultar em ganhos significativos.

Os resultados obtidos demonstram que a utilização de threads traz benefícios significativos nas fases iniciais do processamento, mas esses ganhos tendem a se estabilizar à medida que mais threads são adicionadas. Esse fenômeno pode ser devido ao overhead de gerenciamento de threads e à competição por recursos limitados, o que indica que um balanceamento adequado entre o número de threads e a carga de trabalho é crucial para otimizar o desempenho. Futuros experimentos podem investigar técnicas de balanceamento de carga e gerenciamento mais eficientes para maximizar o uso de threads e melhorar ainda mais o desempenho do sistema.

Essa análise fornece uma base sólida para a compreensão dos efeitos da concorrência em tarefas de processamento intensivo e pode guiar decisões de implementação em projetos similares.

Gráficos





Referências

Soares, F. A. L., Nobre, C. N., & Freitas, H. C. (2019). Parallel Programming in Computing Undergraduate Courses: a Systematic Review of Literature.

SCHILDT, Herbert. Java para iniciantes. 6. Porto Alegre: Bookman, 2015.

K. -Y. Chen, J. M. Chang and T. -W. Hou, "Multithreading in Java: Performance and Scalability on Multicore Systems," in *IEEE Transactions on Computers*, vol. 60, no. 11, pp. 1521-1534, Nov. 2011, doi: 10.1109/TC.2010.232.