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:

Versões 1 a 10: A Versão 1, que não utiliza threads e processa todas as cidades na thread principal, apresentou o maior tempo médio de execução, com 1274 ms. Isso já era esperado, pois o processamento sequencial, sem concorrência, exige mais tempo para lidar com grandes volumes de dados. A partir da Versão 2, ao dividir o processamento em 2 threads, o tempo médio cai drasticamente para 649 ms, representando uma redução de cerca de 50%. Esse padrão de melhoria continua nas Versões 3 e 4, com 4 e 8 threads, onde os tempos médios diminuem ainda mais para 365 ms e 282 ms, respectivamente. Entretanto, na Versão 5, o ganho de performance parece estabilizar. Com 16 threads, o tempo médio foi de 276 ms, uma melhora marginal em relação à Versão 4. Da mesma forma, as Versões 6 e 7, com 32 e 64 threads, mostraram tempos médios muito semelhantes, ambos em torno de 275-277 ms. Isso indica que, apesar de aumentar o número de threads, o benefício em termos de desempenho já se aproximava de um limite. Curiosamente, a partir da Versão 8, onde são usadas 80 threads, observou-se um aumento no tempo médio para 466 ms. Esse aumento se mantém nas Versões 9 e 10, sugerindo que o uso de um número excessivo de

threads pode introduzir sobrecarga no sistema, resultando em tempos mais elevados, isso reforça a ideia de que a eficiência do sistema pode ter atingido um teto.

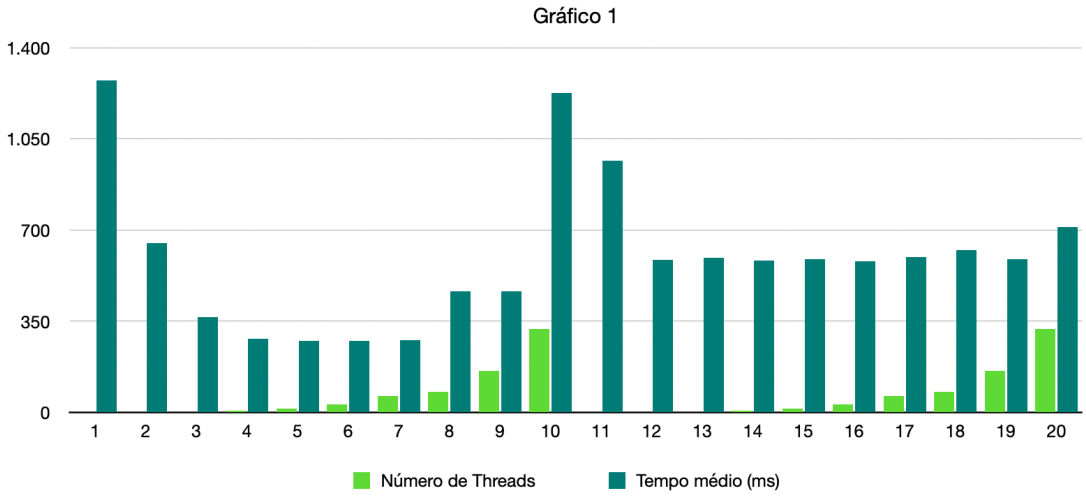
Versões 11 a 20: As Versões 11 a 20 introduziram o processamento de threads por cidade e ano, o que traz uma complexidade maior na gestão das threads. Na Versão 11, o tempo médio foi de 966 ms, indicando que, com threads adicionais para processar os anos, a sobrecarga começou a afetar negativamente o desempenho. Nas Versões 12 a 16, o desempenho melhora novamente, com tempos médios variando de 585 ms a 579 ms, um indicativo de que o uso controlado de threads por cidade e ano ainda resulta em melhorias de desempenho quando o número de threads é moderado. Contudo, a partir da Versão 17, com mais threads sendo usadas para o processamento, o tempo médio voltou a subir. A Versão 18 apresentou um aumento significativo para 623 ms, sugerindo uma maior sobrecarga de threads e uma possível saturação dos recursos disponíveis. As Versões 19 e 20 mostraram tempos mais elevados, especialmente a Versão 20, que teve um tempo médio de 712 ms. Isso pode ser explicado pelo fato de que o uso excessivo de threads para processar tanto cidades quanto anos, aliado à sobrecarga de gerenciamento de threads, leva a um desempenho inferior em comparação com o número ideal de threads.

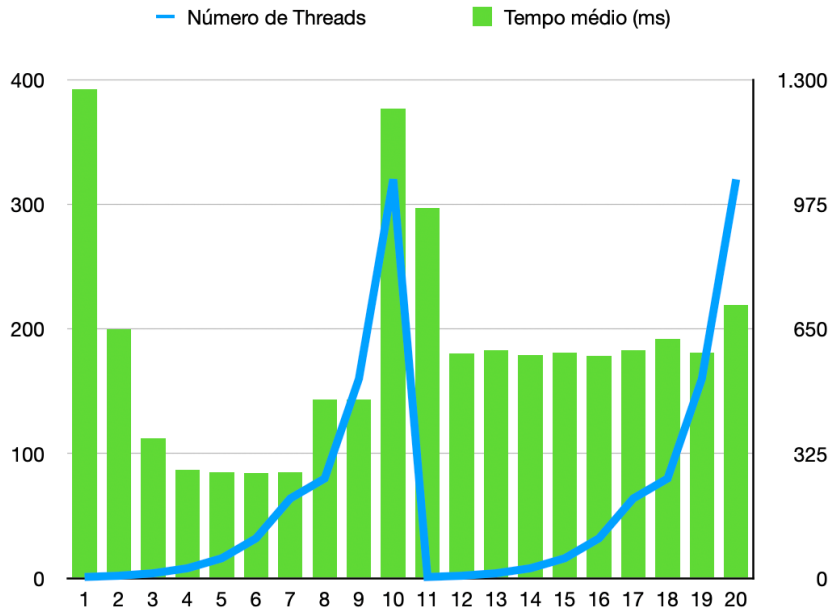
O experimento demonstrou que o uso de threads é uma solução eficiente para reduzir o tempo de execução, especialmente nas primeiras versões, onde o número de threads foi cuidadosamente escalonado. No entanto, ao aumentar excessivamente o número de threads, a sobrecarga do sistema começa a impactar negativamente o desempenho. A partir de um certo ponto, adicionar mais threads não apenas deixa de melhorar o tempo de execução, como também o piora. O desempenho ótimo foi observado nas versões com 4 a 16 threads, onde houve uma combinação eficiente entre concorrência e tempo de execução. Além disso, a introdução de threads por cidade e por ano nas últimas versões adicionou complexidade, mas também evidenciou que há um limite na capacidade de processamento paralelo que o sistema pode suportar antes de saturar. Esses resultados indicam que a escolha do número de threads deve ser cuidadosamente ajustada para equilibrar os benefícios do paralelismo com os custos de gerenciamento de threads e a sobrecarga de processamento. O experimento com as 20 versões destacou a importância de encontrar um equilíbrio adequado no uso de threads para maximizar a eficiência e minimizar a sobrecarga do sistema. Os ganhos mais significativos foram obtidos nas versões com um número moderado de threads, especialmente entre 4 e 16 threads. Quando o número de threads foi excessivamente aumentado, os benefícios diminuíram e até mesmo se inverteram, levando a tempos de execução mais altos.

A análise sugere que o aumento da concorrência melhora o desempenho até um certo ponto, mas há um limite em que a sobrecarga associada ao gerenciamento de muitas threads começa a

interferir na eficiência. Em suma, para maximizar a performance de um sistema de processamento paralelo, é essencial calibrar o número de threads conforme o volume de dados e a capacidade de hardware, evitando tanto o processamento sequencial quanto o uso excessivo de threads.

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.