UNIVERSIDADE DE FORTALEZA
CENTRO DE CIÊNCIAS TECNOLÓGICAS
CURSO: CIÊNCIA DA COMPUTAÇÃO

Análise de Desempenho de Algoritmos de Busca em Ambientes Concorrentes e Paralelos: Um Estudo Comparativo em Java

Autor 1: Francisco Arthur Nunes Montenegro
Autor 2: Daniel de Carvalho Moreira

Ordenação Paralela. Java. Fork/Join. Merge Sort. Quick Sort.

Resumo

O projeto implementa e compara o desempenho de quatro algoritmos de ordenação clássicos (Merge Sort, Selection Sort, Quick Sort e Bubble Sort) em suas versões sequenciais e paralelas utilizando o framework Fork/Join em Java. A execução comparativa é realizada com diferentes números de *threads* para avaliar o ganho de desempenho obtido pelo paralelismo, sendo que a proporção Tempo Paralelo/Tempo Serial é o principal indicador. O Merge Sort demonstrou ser o mais beneficiado pelo paralelismo, com uma razão tempo Paralelo/Serial de 0,282 (com 5 *threads*), enquanto os demais algoritmos apresentaram ganhos marginais ou degradação de desempenho devido ao *overhead* de sincronização e comunicação.

Introdução

O problema da ordenação de dados é fundamental na Ciência da Computação. Com o avanço do hardware e a proliferação de processadores *multi-core*, a paralelização de algoritmos tradicionais surge como uma estratégia chave para otimizar o tempo de execução. Este trabalho foca na implementação de algoritmos de ordenação clássicos, como Merge Sort, Selection Sort, Quick Sort e Bubble Sort, utilizando o paradigma "dividir e conquistar" para introduzir o paralelismo. O objetivo principal é analisar o ganho de desempenho das versões paralelas em Java, utilizando o framework Fork/Join, em comparação com suas respectivas versões sequenciais, variando o número de *threads* utilizadas.

Metodologia

A metodologia adotada consistiu na implementação em Java de versões sequenciais e paralelas dos algoritmos Merge Sort, Selection Sort, Quick Sort e Bubble Sort.

Paralelismo: Foi empregado o framework Fork/Join (java.util.concurrent) do Java, ideal para algoritmos recursivos baseados em "dividir e conquistar".

Merge Sort Paralelo: Utiliza RecursiveTask para retornar um resultado (a lista ordenada).

Selection Sort Paralelo: Utiliza RecursiveTask para encontrar o valor mínimo na sublista.

Quick Sort Paralelo: Utiliza RecursiveAction para realizar as divisões e ordenações in-place.

Bubble Sort Paralelo: Implementa uma variação paralela de "Odd-Even Sort", utilizando Thread e CyclicBarrier para sincronização.

Conjunto de Dados: Uma lista de 800 (tamanho = 800) números inteiros aleatórios foi gerada para os testes.

Medição: O tempo de execução (em nanossegundos) foi registrado usando System.nanoTime().

Comparativo: Os resultados foram consolidados em um arquivo CSV, onde a métrica principal de comparação é a razão Tempo Paralelo / Tempo Serial.

Resultados e Discussão

Os testes foram realizados com uma lista de 800 elementos, comparando o tempo de execução serial com as execuções paralelas de 2, 3, 4, 5 e 6 *threads*. A Tabela 1 apresenta os resultados obtidos:

Algoritmo	Tempo Serial (ns)	Nº Threads	Tempo Paralelo (ns)	Comparativo (Paralelo/Serial)
Merge Sort	7.898.900	4	6.987.600	0,8846295
	7.898.900	3	2.921.000	0,36979833
	7.898.900	2	2.709.700	0,34304777
	7.898.900	5	2.230.400	**0,28236842**
	7.898.900	6	2.851.000	0,36093634
Selection Sort	31.480.100	4	72.273.500	2,2958474
	31.480.100	3	43.356.900	1,3772796
	31.480.100	2	30.515.400	**0,9693552**
	31.480.100	5	32.363.800	1,0280718
	31.480.100	6	30.060.800	0,9549144
Quick Sort	1.521.900	4	3.608.300	2,370918
	1.521.900	3	1.918.600	1,260661

Algoritmo	Tempo Serial (ns)	Nº Threads	Tempo Paralelo (ns)	Comparativo (Paralelo/Serial)
	1.521.900	2	2.473.400	1,6252053
	1.521.900	5	1.948.100	1,2800447
	1.521.900	6	1.356.300	**0,8911886**
Bubble Sort	7.154.900	4	135.558.400	18,946232
	7.154.900	3	135.729.100	18,97009
	7.154.900	2	120.046.100	16,778166
	7.154.900	5	59.180.700	8,271353
	7.154.900	6	57.204.300	7,9951224

Os resultados demonstram que o **Merge Sort** obteve o melhor desempenho no ambiente paralelo, com uma razão Paralelo/Serial menor que 1 em todas as configurações, indicando aceleração. O melhor resultado, com a razão de **0,282**, foi alcançado com 5 *threads*. O **Selection Sort** obteve ganhos marginais (razão próxima de 1), enquanto o **Quick Sort** apresentou degradação na maioria dos testes, com exceção da execução de 6 *threads* (0,891), sugerindo que o *overhead* de particionamento superou o ganho de paralelismo. O **Bubble Sort** paralelo (Odd-Even Sort) demonstrou ineficiência extrema, com a razão chegando a **18,97**. Isso é atribuído à alta interdependência de dados e ao custo de sincronização das *threads*.

Conclusão

O estudo demonstrou que a eficiência da paralelização de algoritmos de ordenação no ambiente Java Fork/Join está fortemente ligada à natureza do algoritmo. O **Merge Sort** provou ser o mais eficiente no paradigma "dividir e conquistar", resultando em uma aceleração significativa (melhor razão de 0,282), confirmando sua adequação à paralelização. Os demais algoritmos, como Selection Sort, Quick Sort e, especialmente, Bubble Sort, sofreram com o *overhead* de paralelização ou com sua inerente dificuldade de serem divididos de forma eficiente, resultando em tempos de execução paralelos semelhantes ou, em muitos casos, piores que suas contrapartes sequenciais.

Referências

CHATGPT. **Plataforma de IA Generativa**. [S.l.]: OpenAI, 2023. Disponível em: https://chatgpt.com. Acesso em: 30 out. 2025.

SHARMA, Keshav. **Parallel Bubble Sort**. *Medium*, [S.l.], 2019. Disponível em: https://medium.com/@keshav519sharma/parallel-bubble-sort-7ec75891afff. Acesso em: 30 out. 2025.

STACK OVERFLOW. **Parallelise bubble sort using CUDA**. [S.l.], 2017. Disponível em: https://stackoverflow.com/questions/42688288/parallelise-bubble-sort-using-cuda. Acesso em: 30 out. 2025.

STACK OVERFLOW. **Parallel Quicksort in C**. [S.l.], 2012. Disponível em: https://stackoverflow.com/questions/8686865/parallel-quicksort-in-c. Acesso em: 30 out. 2025.

HACKERNOON. ***Parallel Merge Sort with Fork/Join Framework in Java***. [S.l.], 2021.

Disponível em: https://hackernoon.com/parallel-merge-sort-with-forkjoin-framework.

Acesso em: 30 out. 2025.

Anexos

Link do gitHub:https://github.com/DanielMoreiraCarv/ProjetoSortsComparative