

PARALELIZAÇÃO DO ALGORITMO QUICK-SORT MULTI-PIVOT COM A UTILIZAÇÃO DE THREADS

Jean da Silva Bragamonte¹

Flávia Morales Antunes²

Bruno Silveira Neves³

Resumo:

Organizar dados é um problema recorrente na Computação, pois muitas aplicações possuem uma grande quantidade de dados e a tendência, se os dados não forem previamente organizados, é a de que seja necessário um grande esforço computacional para encontrar os dados desejados. A fim de agilizar a procura e obter um bom desempenho foram criados os algoritmos de ordenação. Existe uma grande variedade destes algoritmos, tais como Bubble-sort, Merge-sort, Selection-sort e Insertion-Sort. No entanto, neste artigo, iremos tratar sobre o algoritmo Quick-Sort, por este ser um dos mais eficientes (em termos de tempo de ordenamento) em meio ao rol de algoritmos já propostos. Existem muitas variações do algoritmo Quick-Sort, a mais simples é o single-pivot, onde a ordenação ocorre com a utilização de apenas um pivô (elemento usado para auxiliar no processo de ordenamento). Com o tempo, outras implementações baseadas nesta solução mais simples de Quick-sort também se tornaram disponíveis, tais como o Quick-sort dual-pivot e multi-pivot. O algoritmo Quick-sort é conhecido como um dos mais rápidos, porém, estudos demonstraram que um dual-pivot tende a ordenar mais rapidamente um conjunto de dados, sugerindo que o aumento na de pivôs contribui para um melhor desempenho do algoritmo clássico Quick-sort. Tendo em vista a velocidade da ordenação, a proposta deste trabalho é melhorar o tempo de execução de um Quick-sort multi-pivot, por meio de uma implementação utilizando múltiplas threads. Para isso, a estratégia central consiste em dividir o conjunto de dados a ser ordenado (vetor) em vários blocos com menos elementos para que sejam ordenados simultaneamente. No método utilizado foram feitos ordenamentos paralelos utilizando threads, na medida que as threads ordenavam suas partes, o vetor era novamente dividido em menor número de threads que na fase anterior, até chegar na thread final que irá deixar o vetor completamente ordenado. Como resultado obtivemos que ao separar o vetor em quatro partes, depois duas e após isso, a thread final, percebeu-se que foi o mais eficiente quick-sort dual pivot com paralelismo.

Palavras-chave: java;paralelismo;quicksort;multithreading

Modalidade de Participação: Iniciação Científica

PARALELIZAÇÃO DO ALGORITMO QUICK-SORT MULTI-PIVOT COM A UTILIZAÇÃO DE THREADS

¹ Aluno de graduação. jeanbragamonte@gmail.com. Autor principal

² Aluno de Graduação. flaviamoralesantunes@gmail.com. Co-autor

³ Docente. brunoneves@unipampa.edu.br. Orientador

PARALELIZAÇÃO DO ALGORITMO *QUICK-SORT MULTI-PIVOT* COM A UTILIZAÇÃO DE *THREADS*

1 INTRODUÇÃO

Organizar dados é um problema recorrente na Computação, pois muitas aplicações possuem uma grande quantidade de dados e a tendência, se os dados não forem previamente organizados, é a de que seja necessário um grande esforço computacional para encontrar os dados desejados. A fim de agilizar a procura e obter um bom desempenho foram criados os algoritmos de ordenação.

Existe uma grande variedade destes algoritmos, tais como Bubble-sort, Merge-sort, Selection-sort e Insertion-Sort. No entanto, neste artigo, iremos tratar sobre o algoritmo *Quick-Sort*, por este ser um dos mais eficientes (em termos de tempo de ordenamento) em meio ao rol de algoritmos já propostos. Existem muitas variações do algoritmo *Quick-Sort*, a mais simples é o *single-pivot*, onde a ordenação ocorre com a utilização de apenas um pivô (elemento usado para auxiliar no processo de ordenamento). Com o tempo, outras implementações baseadas nesta solução mais simples de Quick-sort também se tornaram disponíveis, tais como o Quick-sort *dual-pivot* e *multi-pivot* [AUMÜLLER *et al.*, 2016]. O algoritmo *Quick-sort* é conhecido como um dos mais rápidos, porém, estudos demonstraram que um *dual-pivot* tende a ordenar mais rapidamente um conjunto de dados [FEOFILOFF, 2015], sugerindo que o aumento na de pivôs contribui para um melhor desempenho do algoritmo clássico Quick-sort.

Tendo em vista a velocidade da ordenação, a proposta deste trabalho é melhorar o tempo de execução de um *Quick-sort multi-pivot*, por meio de uma implementação utilizando múltiplas *threads*. Para isso, a estratégia central consiste em dividir o conjunto de dados a ser ordenado (vetor) em vários blocos com menos elementos para que sejam ordenados simultaneamente.

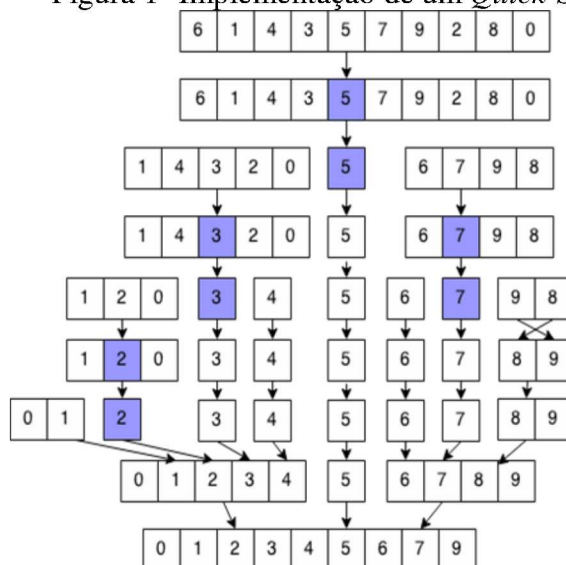
2 *QUICK-SORT*

Quick-Sort é um eficiente método de ordenação, que visa realizar uma ordenação de um grande conjunto de dados usando partições menores para facilitar a ordenação do mesmo [SOUZA *et al.*, 2017]. Para isso, conforme ilustra a Figura 1, primeiramente, é escolhido um elemento aleatório denominado de pivô e, em seguida, são feitas comparações de elementos onde os elementos maiores que o pivô, ficarão à direita dele, e os menores a esquerda. Quando a primeira volta estiver concluída, será selecionado um novo pivot para cada uma das duas partições previamente formadas (elementos maiores e elementos menores que o pivô) e o algoritmo segue assim recursivamente até o final da execução, onde estará todo o vetor ordenado [CAR, 1962].

Se o pivô for o maior ou o menor número do vetor, o algoritmo perderá desempenho, pois recursivamente o método que ordena será chamado muitas vezes, fazendo com que o algoritmo que deveria ser um dos mais rápidos se torne um dos mais lentos.

Utilizando apenas um pivô este algoritmo já possui um bom desempenho, porém, conforme demonstram os nossos resultados, uma solução que explora paralelamente suas trocas possibilita que o algoritmo apresente um desempenho melhorado em comparação com com as soluções sequenciais do mesmo algoritmo nas versões Quick-Sort Clássico e Quick-Sort Dual-Pivot.

Figura 1- Implementação de um *Quick-Sort*.

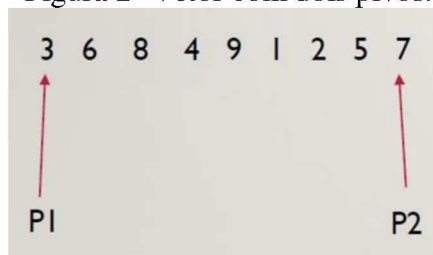


Fonte: do autor, 2017.

3 QUICK-SORT MULTI-PIVOT

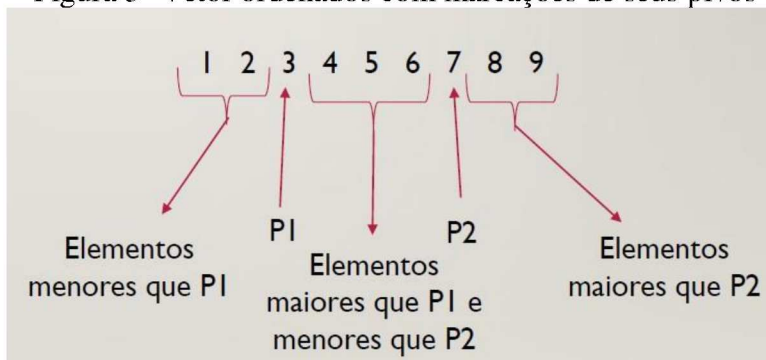
Multi-pivot é uma variação do *Quick-Sort* clássico (implementado com apenas um pivô), que pode utilizar tantos pivôs quantos se desejar[MASHARANI]. Se o desejado for uma execução *dual-pivot*, ou seja, com dois pivôs como mostra a Figura 2, serão escolhidos dois elementos aleatórios para o particionamento do vetor, onde ambos serão comparados com todos os demais elementos. Após esta comparação ocorrer, os valores à esquerda do primeiro pivô serão todos menores que ele. Os elementos à direita do primeiro pivô e à esquerda do segundo pivô terão valores maiores que o pivô um e menores que o pivô dois e, por fim, os valores que ficarem à direita do segundo pivô terão valores maiores que ele, conforme a Figura 3.

Figura 2- Vetor com dois pivôs.



Fonte: do autor, 2017.

Figura 3- Vetor ordenados com marcações de seus pivôs

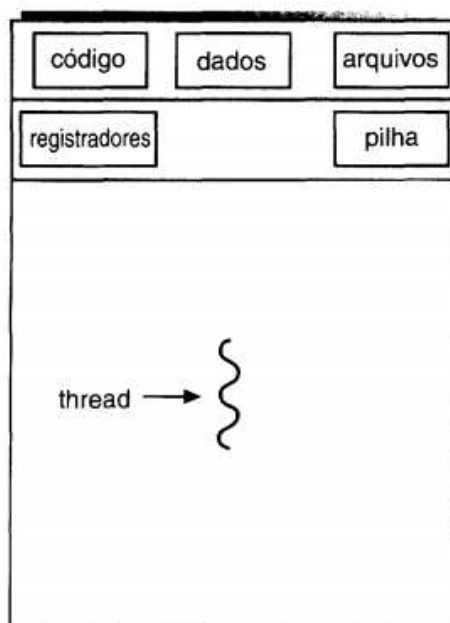


Fonte: do autor, 2017.

4 THREAD

Quando um programa que tem uma grande demanda de atividades, ele tende a ser mais rapidamente executado se forem utilizados mecanismos para realizar suas tarefas simultaneamente. A execução simultânea de vários fluxos de software pertencentes a um processo é denominada de *Multithread*, conforme ilustra a Figura 4. As *threads* podem ser facilmente criadas e destruídas [SILBERSCHATZ *et al.*, 2004]

Figura 4- Representação de uma *thread* dentro de um processo



Fonte: SILBERSCHATZ *et al.*, 2004.

5 METODOLOGIA

A linguagem de programação utilizada para a implementação do *Quick-Sort Multi-Pivot*, foi Java, na plataforma Netbeans. Nesta implementação foram criadas várias opções nas separações das *threads* e em cada uma das separações foram criados dois pivôs como padrão, resultando em um *Quick-Sort Dual-Pivot*. O método para a criação das opções consistiu em iniciar a execução com um número maior de *threads*, onde cada thread executa as comparações e trocas em um pedaço do vetor em paralelo com as demais.

Foram feitas dez divisões para a execução, são elas:

1. 8 threads -> 4 threads -> 2 threads -> 1 thread
2. 8 threads -> 4 threads -> 1 thread
3. 8 threads -> 2 threads -> 1 thread
4. 8 threads -> 1 thread
5. 6 threads -> 2 threads -> 1 thread
6. 6 threads -> 1 thread
7. 4 threads -> 2 threads -> 1 thread
8. 4 threads -> 1 thread
9. 2 threads -> 1 thread
10. 16 threads -> 1 thread

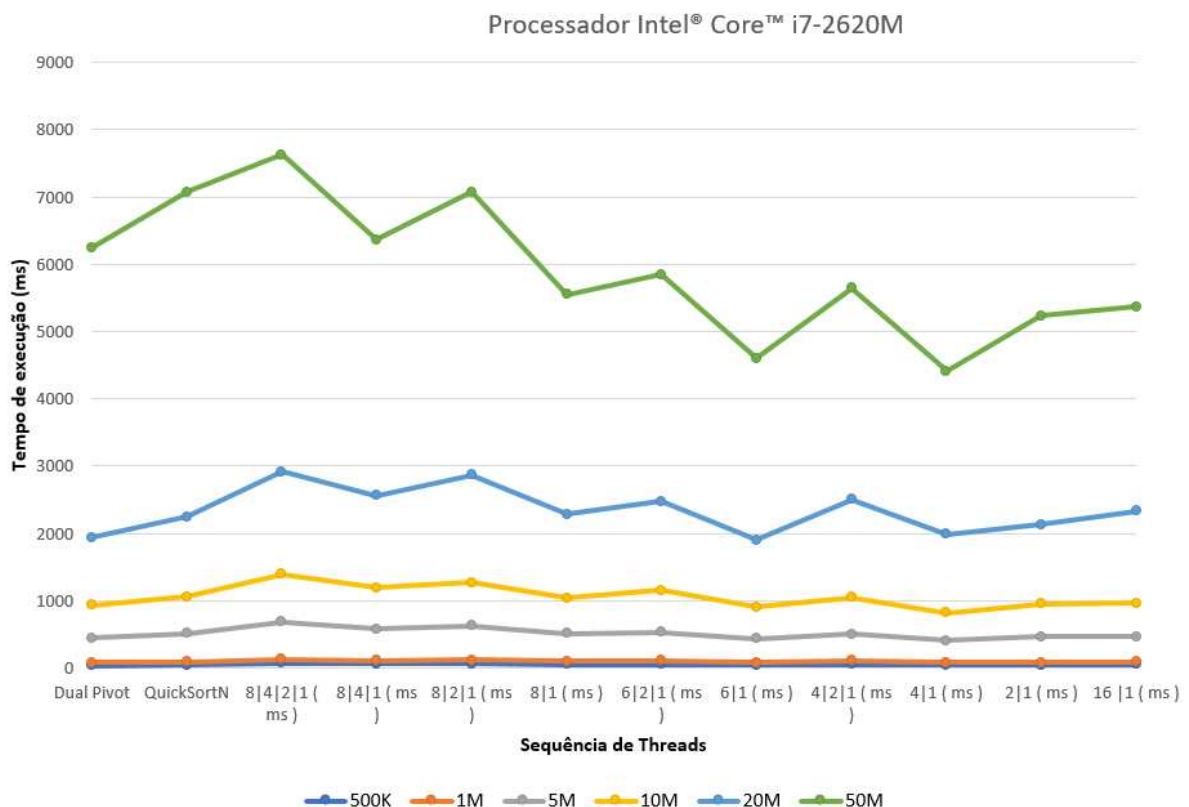
A execução ocorre da seguinte forma: primeiramente é solicitado qual a divisão de threads desejada, ou seja, o usuário deve escolher uma destas dez divisões. Supondo que a opção escolhida tenha sido a opção 1, o vetor será particionado inicialmente em oito threads, cada uma delas possuindo dois pivôs e todas sendo ordenadas ao mesmo tempo. Quando a ordenação de cada uma das oito threads estiver concluída, elas irão se unir de duas em duas e virar quatro threads. O processo de ordenação irá se repetir e quando estiverem todas completamente ordenadas irão virar duas threads. Estando estas duas ordenadas, por fim, elas irão se unir em apenas uma thread, gerando um vetor totalmente ordenado.

6 RESULTADOS e DISCUSSÃO

Para realizar uma comparação de tempo de execução com a solução multithreading proposta, a fim de mensurar os benefícios, foram utilizados os algoritmos clássicos *Quick-Sort* e *Quick-Sort Dual-Pivot*, tendo sido o código fonte destes algoritmos descritos pelos presentes autores deste trabalho, com base em suas respectivas descrições disponíveis em [CAR 1962] e [YAROSLAVSKIY. 2009].

O algoritmo foi testado em três máquinas, entretanto este artigo apresentará o melhor resultado obtido, o qual foi de um Processador *Intel Core i7-2620M*, que possui 2 núcleos de processamento. Para a obtenção dos resultados, cada ponto do gráfico corresponde a uma média dos resultados de cinco execuções. A figura 5 apresenta os dados obtidos:

Figura 5 - Dados da execução do algoritmo de *Quick-Sort*, *Quick-Sort Dual-Pivot* Convencional e *Quick-Sort Dual-Pivot* com *Threads* em um Processador *Intel Core i7-2620M*.



Fonte: do autor, 2018.

Como observado no gráfico da figura 5, para as versões que possuem uma sequência menor de divisões de *threads*, como por exemplo 6|1 ou 4|1, o tempo de execução é menor, porém quando o tamanho do vetor é menor que ou igual a 1M o *Quick-Sort Dual-Pivot* convencional é mais rápido que com *threads*. Estima-se que os ganhos obtidos sejam decorrentes do paralelismo quando o vetor é particionado, pois, quando chegar nas últimas ordenações ele já está parcialmente ordenado, tornando o trabalho da última thread mais fácil e rápido.

Complementarmente, considerando uma média geral dos diversos tamanhos de vetor, a sequência de utilização de *threads* que inicia com quatro threads e depois reduz para uma thread, é a execução mais rápida.

7 CONSIDERAÇÕES FINAIS

Concluimos que o *Quick-Sort Dual-Pivot* é um algoritmo rápido e difícil de ser batido para ordenamento de pequenos conjuntos de dados. Entretanto, para grandes conjuntos de dados, uma solução *multithread* pode ser mais rápida que a execução convencional (mono-thread). Considerando a tendência de grande parte das aplicações para manipulação de grandes conjuntos de dados, a estratégia proposta tende a compor uma solução relevante para ordenamento de dados nestas aplicações.

REFERÊNCIAS

AUMÜLLER M.; DIETZFELBINGER M. Optimal Partitioning for Dual-Pivot Quicksort. **Technische Universität Ilmenau**. Alemanha. 2016.

FEOFILOFF Paulo. Análise do algoritmo QuickSort, Disponível em: https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/quick.html Acessado em 12/09/2018.2015.

MASHARANI Nisha. Discussion of Multi-Pivot Quicksort: Theory and Experiments Disponível em: <http://cs.stanford.edu/~rishig/courses/ref/111a.pdf> Acessado em 12/09/2018.

SOUZA, J.É.G.; RICARTE, J.V.G.; LIMA, N.C.A. Algoritmos de Ordenação: Um Estudo Comparativo. **Anais do Encontro de Computação do Oeste Potiguar ECOP/UFRSA (Universidade Federal Rural Semi-Árido)**. Pau dos Ferros/RN, v. 1, p. 166-173, jun. 2017.

CAR Hoare; **Quicksort, The Computer Journal**. Volume 5, Edição 1, 1º de janeiro de 1962, páginas 10–16, <https://doi.org/10.1093/comjnl/5.1.10>

SILBERSCHATZ, A.; GALVIN, P. B.; GAGME G. **Operating System Concepts with Java** Tradução autorizada do idioma inglês da edição publicada por John Wiley & Sons Copyright© 2004 by John Wiley & Sons, Inc. Elsevier. 8. Ed. 2010.

YAROSLAVSKIY, V. Dual-Pivot Quicksort algorithm. Feb./Sep. 2009.