UMA PROPOSTA DE IMPLEMENTAÇÃO DE ALGORITMO QUICKSORT MULTI-PIVOT PARALELO

CASTRO, W. H. L.[1], FONSECA, I. O.[2], FOLLMANN, J.[3], NEVES, B. S.[4]

[1] Universidade Federal do Pampa (Unipampa) – Bagé – RS – Brasil – welbertcastro.aluno@unipampa.edu.br

[2] Universidade Federal do Pampa (Unipampa) – Bagé – RS – Brasil – igorfonseca.aluno@unipampa.edu.br

[3] Universidade Federal do Pampa (Unipampa) – Bagé – RS – Brasil – jonefollmann.aluno@unipampa.edu.br

[4] Universidade Federal do Pampa (Unipampa) – Bagé – RS – Brasil – brunoneves@unipampa.edu.br

RESUMO

O algoritmo de ordenação Quicksort foi proposto pela primeira vez em 1960. Ele escolhe arbitrariamente um elemento de uma lista para utilizar como pivot e a subdivide em duas, uma com valores menores que o pivot, outra com valores maiores. Então, aplica recursivamente o procedimento às subdivisões, até que a lista esteja ordenada. Em 2009, foi proposta a otimização do algoritmo utilizando dois pivots. Já em 2013, foi apresentado pela primeira vez o Quicksort com três pivots. Soluções mais recentes implementam paralelismo no algoritmo para obter maior eficiência. Identifica-se a oportunidade de integrar as soluções multi-pivot e paralela para otimizar o desempenho. A literatura mostra que a eficiência em implementações multi-pivot deve-se ao comportamento da cache em processadores modernos, e as vantagens do processamento paralelo devem-se à capacidade de execução de múltiplas instruções simultaneamente. O objetivo do trabalho é a proposta de uma implementação multi-pivot paralela do Quicksort. Foram desenvolvidos, em Java, os algoritmos existentes na literatura: Quicksort Mono-Pivot, Dual-Pivot e Tri-Pivot, em suas versões sequencial e paralela. O paralelismo utilizado nestes algoritmos consiste em executar as subdivisões do vetor inicial em threads. Em seguida, foi implementado o ParaQuick, algoritmo Mono Pivot totalmente paralelo. Os resultados dos testes mostram que os algoritmos sequenciais possuem os piores desempenhos, os algoritmos que utilizam threads possuem melhor desempenho, pouco dependendo do número de pivots, e o ParaQuick possui desempenho intermediário. Com base no ParaQuick, a proposta de nova implementação deste trabalho é o algoritmo Quicksort Dual-Pivot totalmente paralelo.

Palavras-chave: Quicksort, paralelismo, otimização, Java.

1 INTRODUÇÃO

A história do desenvolvimento de versões otimizadas do algoritmo de ordenação Quicksort, que surgiu em 1960, é relativamente recente. Iniciou-se quando em Yaroslavskiy (2009) foi desenvolvido o Quicksort Dual Pivot. Com o

avanço dos estudos nessa área, em Kushagra (2014) foi proposto o Three-Pivot Quicksort, trazendo fortes evidências de que os benefícios notados nas implementações Multi-pivot do Quicksort devem-se ao comportamento da cache das arquiteturas de processadores mais modernos.

Mais recentemente, foi apresentado o ParaQuick, um algoritmo totalmente paralelo (MAUS, 2015). A proposta deste trabalho é implementar a solução de melhor desempenho de um algoritmo Quicksort Multi-Pivot Paralelo.

2 METODOLOGIA (MATERIAL E MÉTODOS)

Este projeto está constituído sobre um conjunto de etapas previamente definidas a fim de atender como resposta efetiva às demandas do problema de pesquisa apresentado, o qual aponta para a necessidade de desenvolver uma solução do algoritmo de ordenação Quicksort paralelo com desempenho superior aos já apresentados na literatura.

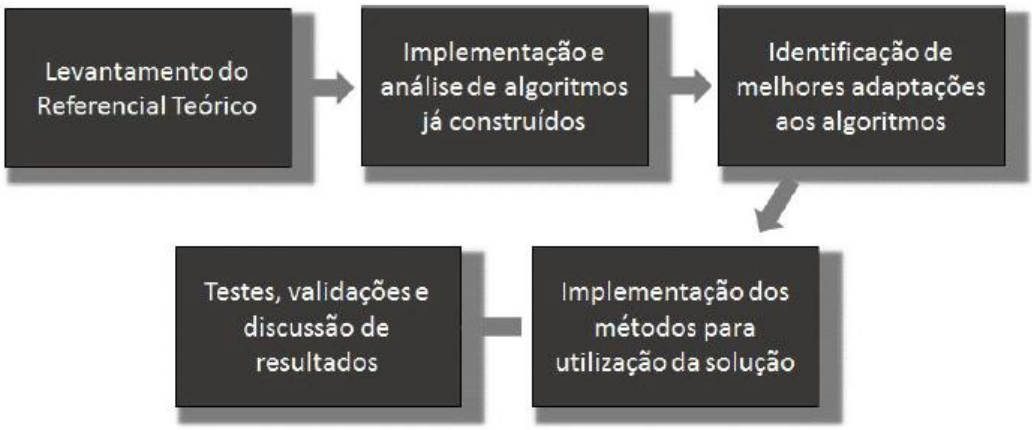

Figura 1. Organograma da metodologia

Conforme apresentado na Figura 1, o estudo baseou-se em uma primeira etapa no levantamento teórico sobre os algoritmos Quicksort, tanto sequenciais como paralelos. A segunda etapa contempla a implementação e análise dos algoritmos já construídos anteriormente em linguagem de programação Java, para em seguida identificar as melhores adaptações a estes algoritmos. Nesta etapa foi identificado que a solução mais pertinente seria a construção de um algoritmo Quicksort paralelizado desde a primeira parte, partindo então, para a implementação dos métodos propostos, os quais podem ser testados, validados e discutidos na quinta etapa.

Primeiramente foi desenvolvido, na linguagem de programação Java, os algoritmos já existentes na literatura, que são: Quicksort Mono Pivot (Sequencial e

Paralelo), Quicksort Dual Pivot (Sequencial e Paralelo) e o Quicksort Tri Pivot (Sequencial e Paralelo). Durante a implementação dos algoritmos paralelos, foi necessário adaptar as soluções existentes na literatura para utilizar funções e variáveis de acordo com as ferramentas de sincronização da linguagem Java e tendo em vista a solução que seria proposta.

A técnica de paralelização utilizada nesses algoritmos consiste em, após uma primeira execução sobre o vetor inteiro, dividir os sub-vetores para serem executados em diferentes threads. Utiliza-se um contador sincronizado entre todas as threads. Sempre que o contador for maior ou igual a n*2, onde n é o número de processadores lógicos do sistema, o programa recorre a chamadas recursivas para ordenar os sub-vetores, ao invés de criar novas threads. Quando o contador chegar a zero, e vetor está ordenado e o programa termina sua execução.

Após essa etapa do desenvolvimento, foi implementado o algoritmo proposto por Arne Maus. É o ParaQuick, um Quicksort Mono Pivot totalmente paralelo, ou seja, que é paralelizado desde sua primeira parte, diferente dos outros, que implementam o paralelismo somente após executarem uma vez sobre o vetor inteiro.

A técnica de paralelização do ParaQuick divide o vetor em k partes de tamanho igual (ou quase). Cada parte é entregue a uma thread individual e todas elas determinam um mesmo valor pivot. Nessa etapa ocorre a primeira sincronização, para garantir que todas as threads já tenham obtido o valor pivot. Após isso, cada thread divide sua porção do vetor entre os elementos maiores e os menores que o pivot. Então ocorre mais uma sincronização para que todas as threads terminem essa ordenação. Em uma terceira etapa, as k/2 porções maiores que o pivot mais à esquerda são trocadas com as k/2 porções menores que o pivot mais à direita. Ocorre uma terceira sincronização e, finalmente, metade das threads é alocada recursivamente para o segmento da esquerda e metade para o segmento da direita. Após uma quarta e última barreira de sincronização, se algum dos segmentos tiver apenas uma thread, é feito um ordenamento Quicksort sequencial. Caso contrário, os passos são repetidos.

3 RESULTADOS E DISCUSSÃO

Abaixo, são apresentados os resultados dos testes dos algoritmos implementados. Para cada algoritmo, foram realizados testes com os tamanhos de vetores 500k, 1M, 5M, 10M, 20M e 50M, preenchidos com números inteiros aleatórios num intervalo de 0 a t*2, onde t é o tamanho do vetor. Os algoritmos

foram executados cinco vezes para cada tamanho de vetor e tiveram seu tempo de execução medido, de modo que o tempo apresentado nos gráficos abaixo é a média desses cinco valores.

Os testes foram realizados em duas arquiteturas. A primeira delas é um notebook Dell Vostro 5480 com processador Intel Core i5 4210U com frequência 1.7 GHz, 2 núcleos e 4 processadores lógicos, 4GB de memória RAM, memória cache 3MB SmartCache e sistema operacional Windows 10. A segunda arquitetura é um servidor HP ProLiant ML350 G6 com processador Intel Xeon E5620 de 2.4 GHz, 4 núcleos e 8 processadores lógicos, 6GB de memória RAM, memória cache de 12MB SmartCache e sistema operacional Linux Mint 18.1 Serena.

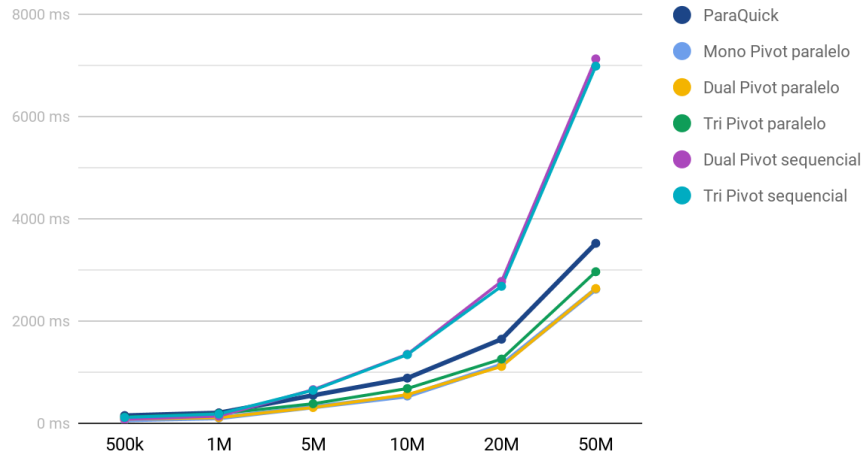

Figura 2. Gráfico dos testes na arquitetura Intel Core i5.

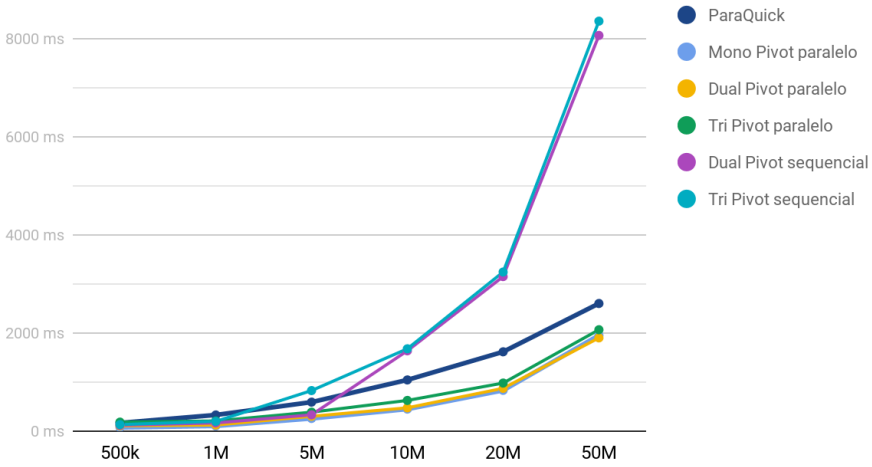

Figura 3. Gráfico dos testes na arquitetura Intel Xeon E5620.

Como pode ser observado nos gráficos da Figura 2 e Figura 3, as soluções sequenciais possuem desempenho muito inferior se comparadas às paralelas. Da mesma forma, em ambas as arquiteturas, os algoritmos paralelos que utilizam threads para ordenar os sub-vetores a partir da segunda etapa são os que possuem

melhor desempenho. Interessante notar que o número de pivots adotado é pouco relevante nesse caso, visto que as três soluções testadas (com um, dois e três pivots) apresentam desempenho bastante semelhante. Já o ParaQuick, solução que implementa o paralelismo total com um único pivot, possui desempenho intermediário. Assim, a proposta de implementação deste trabalho é a utilização do conceito central desenvolvido por Arne Maus no ParaQuick, aliado à utilização de palalelismo com threads, comprovadamente eficaz segundo os testes realizados.

4 CONCLUSÃO

Neste trabalho é proposta uma solução de melhor desempenho de um algoritmo Quicksort Multi-pivot paralelo. Com base na observação e testes realizados em algoritmos já apresentados na literatura, foi possível chegar à proposta de implementação de um algoritmo com essas características. Assim, pretende-se fazer em trabalhos futuros a implementação do Quicksort Dual-pivot totalmente paralelo, tendo em vista a possibilidade de melhor desempenho em relação ao que foi proposto por Arne Maus.

REFERÊNCIAS

YAROSLAVSKIY, Vladimir. Dual-pivot quicksort algorithm. **Dosegljivo: http://codeblab. com/wp-content/uploads/2009/09/DualPivotQuicksort. pdf**, 2009.

WILD, Sebastian. Java 7's Dual Pivot Quicksort. 2012.

KUSHAGRA, Shrinu et al. Multi-pivot quicksort: Theory and experiments. In: **Proceedings of the Meeting on Algorithm Engineering & Experiments**. Society for Industrial and Applied Mathematics, 2014. p. 47-60.

MAUS, Arne. A full parallel Quicksort algorithm for multicore processors. In: **NIK**. 2015.