**ORDENAÇÃO NUMÉRICA UTILIZANDO THREADS COM ALGORITMO MERGE SORT**

João Vitor Pogiolli do Lago - 158049

Conteúdo

[1. Introdução 3](#__RefHeading___Toc2191_3010923653)

[2. Objetivo 4](#__RefHeading___Toc2193_3010923653)

[3. Software 4](#__RefHeading___Toc2195_3010923653)

[3.1. Cálculo do Número de Elementos para cada Thread 5](#__RefHeading___Toc2197_3010923653)

[3.2. Definições TIME\_T e CLOCK\_T 6](#__RefHeading___Toc2199_3010923653)

[4. Testes e Validação Multi-Threads 7](#__RefHeading___Toc2201_3010923653)

[5. Vídeo 9](#__RefHeading___Toc2207_3010923653)

[6. Conclusão 9](#__RefHeading___Toc2205_3010923653)

# Introdução

Atualmente a necessidade de desenvolver softwares com capacidade de processamento multi-thread torna-se inevitável. Devido ao avanço dos recursos computacionais atuais, saber desenvolver e projetar corretamente um software para aproveitar tais recursos é de fundamental.

# Objetivo

Este trabalho tem o objetivo de criar um algoritmo para ordenação de um vetor inteiro de *N* elementos, obtidos através da leitura de *M* arquivos. A ordenação, por sua vez, ocorrerá de forma paralela utilizando threads, com isso, o algoritmo deverá ser capaz de distribuir os *N* elementos a serem ordenados preferencialmente igualmente entre as threads. Para a ordenação proposta, realizaremos os testes para 2, 4, 8 e 16 threads de ordenação.Todo o código fonte, arquivos de entrada e saída, arquivo de teste  
podem ser encontrados no repositório GIT do grupo:

<https://github.com/Poggioli/neuron>

# Software

O software proposto consiste na criação de um número *T* de threads de execução para a ordenação, sendo o valor *T* determinado pelo usuário. Com base no número de threads, nos arquivos informados pelo usuário e no total de elementos lidos dos arquivos, o software irá distribuir para cada thread um número *X* de elementos, tentando sempre distribuir um número igualitário de elementos para todas as threads do sistema.

Para a criação de threads foi utilizada a biblioteca ‘pthread’, a qual implementa e controla todas as threads do software. Para controle das informações passadas para as threads, utiliza-se uma estrutura pré definida no inicio do código, contendo as informações necessárias para a ordenação por cada thread.

## Cálculo do Número de Elementos para cada Thread

Para o software proposto, o cálculo do número de elementos para cada thread para cada uma das *T* threads é feito através de duas expressões matemáticas para determinar o intervalo inferior e superior no vetor de números a serem ordenados.





onde:

|  |  |
| --- | --- |
| **ID** | ID da thread. Valores entre [0, T]. |
| **T** | Número total de threads |
| **N** | Número de elementos para ordenação |

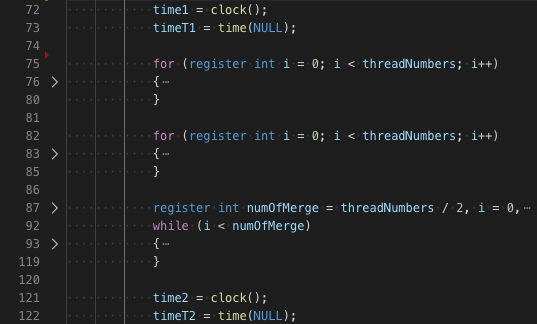
Em uma rodada de ordenação onde o usuário tenha entrado com um número de threads igual a 4 e um número total de 1000 elementos para ordenação, temos os seguintes valores para as posições inferiores e superiores de cada thread.

Tabela 1 - Valores das posições inferiores e superiores para cada thread

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | ID | posicaoInferior | posicaoSupeior |
| 1 | 0 | 0 | 249 |
| 2 | 1 | 250 | 499 |
| 3 | 2 | 500 | 749 |
| 4 | 3 | 750 | 999 |

## Definições TIME\_T e CLOCK\_T

Para o cálculo do tempo de uso do processador pelo algoritmo proposto e o tempo total de execução, foram utilizadas as definições clock\_t e time\_t respectivamente. Ambas são instanciadas no começo do algoritmo (linhas 72 e 73) e tem sua definição de tempo inicial um comando antes das threads serem criadas, e tem sua finalização (linhas 121 e 122) no primeiro comando após todas as threads terem executado suas operações, como vemos na Figura 1 a seguir.

  
Figura 1

# Testes e Validação Multi-Threads

Para validação do código desenvolvido, utilizamos como plataforma de teste um Macbook Pro 2012 com as seguintes configurações da Figura 2:

  
Figura 2

Para uma validação correta de desempenho foi rodado o algoritmo para todos os casos de 2, 4, 8 e 16 threads, utilizando 5 arquivos de entrada com 10.000 (dez mil) números aleatórios cada de 1 até 100.000 (cem mil). A seguir temos exemplos da linha de execução:

1. ./mergesortMultiThread 2 file1.txt file2.txt file3.txt file4.txt file5.txt output.txt
2. ./mergesortMultiThread 4 file1.txt file2.txt file3.txt file4.txt file5.txt output.txt
3. ./mergesortMultiThread 8 file1.txt file2.txt file3.txt file4.txt file5.txt output.txt
4. ./mergesortMultiThread 16 file1.txt file2.txt file3.txt file4.txt file5.txt output.txt

Para a validação final dos testes, rodamos uma bateria de 3 testes do algoritmo, variando o número de threads entre 2, 4, 8 ou 16. Lembramos que o tempo medido é o tempo de uso do processador. Para 2 threads devemos dividir o tempo por 2 para encontrar o tempo real de execução, e para 4, 8 e 16 threads devemos dividir o tempo por 4, visto que esse é o numero máximo de threads de execução do nosso processador, os tempos estão normalizados em milissegundos. O gráfico a seguir diz respeito à primeira bateria de testes.



# Vídeo

<https://www.youtube.com/watch?v=HdiIFFd058Y>

# Conclusão

Ao analizarmos os gráficos de duração dos testes, vemos claramente que o algoritmo se comporta mais rápido com 4 threads para execução, o que é exatamente igual ao número de threads disponíveis no processador. Quando rodamos o teste para 8 ou 16 threads, elas se comportam semelhante ao teste de 4 threads, porém com um pouco mais de tempo devido ao gerenciamento de todas as threads.

Ao analizarmos a duração do algoritmo para 2 threads de execução, vemos que o tempo é aproximadamente 100% maior do que o teste com 4 threads quando colocamos um grande volume de dados. Isso confronta exatamente com o resultado esperado, pois ao utilizarmos todos os núcleos de processamento do processador a aplicação se torna mais eficiente.

Portanto, quando desenvolvemos um sistema, seja ele específico ou não, devemos levar em conta uma análise do hardware no qual o sistema será executado. Não é de interesse desenvolver sistemas multi-thread complexos para hardware onde existem poucos núcleos de execução, como processadores single core ou dual core. Entretanto, é extremamente válido a criação de mecanismos multi-thread quando lidamos com máquinas de alto poder computacional, como servidores.