Aluno: Lucas de Queiroz Silva e Silva
16 de fevereiro de 2021

# Modulo 2 – Trabalho de Implementação (2020/1 REMOTO)

**Computação Concorrente (MAB-117)**

**Prof. Silvana Rossetto**

## 1 – Introdução

O trabalho implementa um quicksort multithreading com ordenação crescente onde a linguagem escolhida de implementação foi "C". Todos os resultados foram obtidos ao rodar o código numa maquina com 16Gb de memória principal, uma cpu intel i3-9100f (quad core) e uma máquina virtual Linux rodando no Windows 10.

## 2 – Metodologia

Para modificar a execução do quicksort sequencial para que o mesmo permita a execução multithreading a abordagem escolhida foi a criação das threads selecionando o bloco esquerdo após a quebra do vetor ao definir o pivot (e ordenar os elementos em função do mesmo). Desta forma garantidamente temos um bloco de elementos menores que o pivot que podem ser executados concorrentemente pois seus dados são independentes do resto (pelo algoritmo implementado de forma crescente, ao particionar não pode haver do lado esquerdo um elemento maior que o pivot nem menor do lado direito).

Ao atingir o número máximo de threads a serem executadas, o programa executa o resto das quebras sequencialmente. Desta forma, ao selecionar um número de threads que não seja uma potência de 2, a execução será desbalanceada e alguma(s) threads podem realizar mais operações que outras. Outro fator de desbalanceamento se dá pelo fato de a quantidade de elementos diminuir com a quantidade de partições, o que faz com que as threads iniciais ordenem mais elementos que as seguintes.

Por outro lado, dada abordagem não nos remete a problemas de deadlock (cada thread recebe localmente o conjunto de índices de elementos que vai utilizar e há a condição de parada quando o maior índice é maior ou igual ao menor) e nos garante a corretude da solução.

Para avaliar se o programa executa corretamente, é criado um vetor pseudorandomico de elementos cujo tamanho é passado como parâmetro ao rodar o programa juntamente com o número de threads, ficando a compilação e execução no formato:

```
Gcc -o quicksort QuicksortConcorrente.c -pthread -lm
```

```
./quicksort <número de threads> <número de elementos do vetor>
```

Aluno: Lucas de Queiroz Silva e Silva
16 de fevereiro de 2021

## 3 – Resultados

A seguinte tabela lista os tempos de execução (escolhi os menores dentro de 10 rodados para cada caso) e de ganho de desempenho quando comparado com o tempo sequencial (Ganho de desempenho = Tempo Sequencial / Tempo Concorrente). O tempo de 1 thread foi considerado como sendo o sequencial.

Para avaliar como o programa varia com o número de elementos e o de threads, a execução varia o tamanho do vetor a ser ordenado entre $10^4$, $10^5$ e $10^6$ elementos e o número de threads sendo 1, 2, 4, 8.

| | Nº de Elementos | Tempo | Ganho desempenho |
|---|---|---|---|
| 1 Thread | 10000 | 0.002826 | |
| | 100000 | 0.217440 | |
| | 1000000 | 21.338339 | |
| 2 Threads | 10000 | 0.002121 | 1.3323903819 |
| | 100000 | 0.125944 | 1.72648161087 |
| | 1000000 | 13.513935 | 1.57898783737 |
| 4 Threads | 10000 | 0.001313 | 2.1523229246 |
| | 100000 | 0.078608 | 2.76613067372 |
| | 1000000 | 12.012947 | 1.77627846023 |
| 8 Threads | 10000 | 0.001380 | 2.04782608696 |
| | 100000 | 0.078859 | 2.75732636731 |
| | 1000000 | 12.294124 | 1.73565347153 |

## 4 - Conclusão

Com os resultados obtidos é fácil ver que a abordagem escolhida não escala bem para o tamanho do vetor, provavelmente devido a falhas de cache uma vez que uma thread que tem posse de um bloco suficientemente grande de elementos ao ceder o controle da cpu precisa recarregar os elementos. Devido a forma como é realizada a criação de threads também era esperado que o ganho de desempenho não crescesse na mesma proporção da quantidade de threads (por mais que a lei de Amdahl demonstre que não teríamos de fato um crescimento linear é possível que a aplicação cresça “aproximadamente” em função de 1/nthreads, desde que a parte concorrente seja suficientemente grande). Para 8 threads quando comparado com 4 já era esperado um resultado muito próximo uma vez que o processador possui apenas 4 núcleos e uma quantidade maior de threads só agravaria o problema da falha de cache.