UNIVERSIDADE FEDERAL DO PAMPA– UNIPAMPA

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

HEITOR MAURO CHAVEZ HUARACHE

RELATÓRIO DE SISTEMAS OPERACIONAIS

BAGÉ – RIO GRANDE DO SUL

2022

1. INTRODUÇÃO

O trabalho tem como apresentar alguns conceitos de threads e algumas informações da máquina operada pelo software.

Todo o código desenvolvido ao decorrer deste trabalho pode ser conferido no github[1].

2. HARDWARE EXECUTADO

O trabalho foi executado em um desktop de mesa, onde suas especificações estarão listadas abaixo:

• Processador ryzen 5 3400g(4 cores e 8 threads)

• Placa Mãe b450

• Memória RAM 16gb 3000mhz ddr4

• SSD 500GB

Com isso demos progresso ao código e testando a capacidade e tempo em que executa o programa criado com os desejos do professor.

3. GRÁFICO DE DESEMPENHO

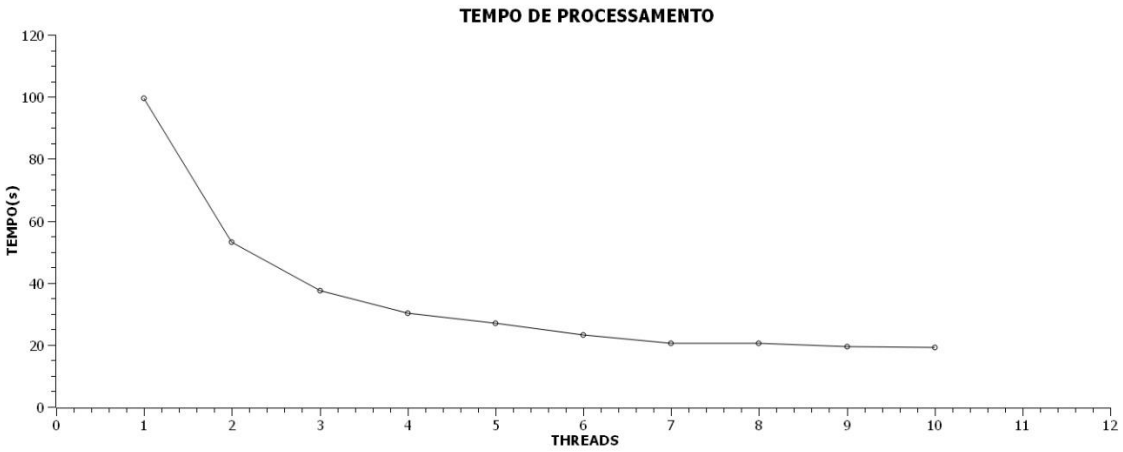
Para fazer o desempenho do c´odigo foi utilizado duas ferramentas para fazer esse processo com conhecimentos de laborat´orio de f´ısica 1, utilizamos um software chamado “Scidavis” para criação do gráfico, junto com o comando ”htop” do linux, onde ´e mostrado todas as informações necessárias para definir o trabalho como as threads criadas, a cache e a quantidade de memória RAM utilizada. Foram desenvolvidos duas versões do código, uma com Interface Gráfica(GUI.py), e a outra sem(euler.py).O intuito era criar primeiramente o código sem interface e deixa-lo funcional para posteriormente integra-lo com interface.

3.1.Sem interface Gráfica

Como dito na seção acima o código sem interface gráfica teve como objetivo deixar o código funcional, isto é, o código deveria aceitar qualquer valor de threads e qualquer valor de entrada. O Objetivo foi alcançado e pode ser conferido na **Figura 1**

[1] https://www.github.com/elcabriton/SO-

Figura 1 - Tempo de processamento



Fonte: Autoria Própria.

De acordo com a figura acima, podemos uma pequena amostra do que foi realizado no trabalhando utilizando até 10 threads. É importante informar que o número N de entradas foi utilizado durante os testes foi constante, um valor de 10000, mas cabe reafirmar que esse valor poderia ser maior conforme a vontade do usuário que o código funcionaria.

Analisando a imagem, podemos ver que utilizando uma única thread, o valor de processamento foi acima de 100s, ao passo que ao utilizar 7 threads, o mesmo ficou pouco acima de 20s. Podemos notar também que a partir de 7 threads o ganho de desempenho é irrisório.

Um outro fato observado ao decorrer dos experimentos deste trabalho, foi o tempo de execução sem utilizar nenhuma thread, que se manteve em uma média de 100s, lembrando que o valor de entrada permaneceu em 10000 para testes.

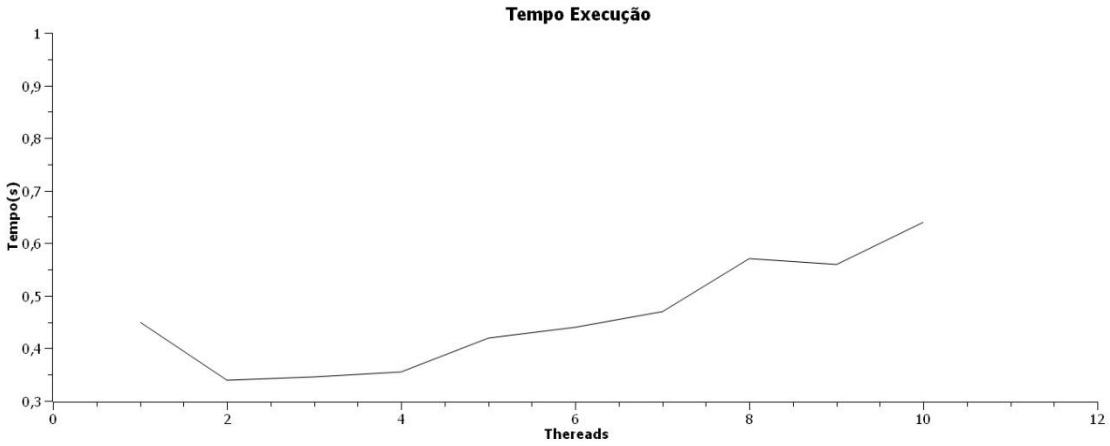
Um breve parágrafo para comentar a respeito do código, foi utilizado a programação funcional, ou seja, a linha de desenvolvimento foi quebrada em diversas funções e separadas da forma que parecesse mais lógica. Cabe relembrar que dessa forma o trabalho atingiu a meta de não se limitar a uma quantidade fixa de entradas N, podendo ser qualquer número sendo determinado pelo usuário através do terminal.

3.2.Com interface Gráfica

Ao terminar o código sem interface gráfica, a ideia era adaptar o código para inserção da GUI e em teoria tudo deveria funcionar perfeitamente, mas problemas foram encontrados ao decorrer dessa abordagem e adaptações foram necessárias para manter o programa rodando.

A mudança mais significa se deu na entrada N, que na subseção anterior poderia ser qualquer número a critério do usuário, mas que ao adicionar a interface gráfica o computador não suportava o processamento requerido e travava. Para realizar os testes e driblar esse problema, foi determinado um número baixo de entradas no valor de 1000. Com esse valor foi possível gerar os dados que podem ser conferidos na tabela a seguir.

Figura 2 - Tempo de processamento após implementação da GUI



Fonte: Autoria Própria.

Analisando o gráfico, podemos conferir que ao utilizar 2 threads, obtivemos o tempo de execução abaixo de 0,4s ao passo que ao utilizar 8 threads, o resultado foi maior, acima de 0,6s. O desempenho de tempo foi piorando conforme o número de threads foi aumentando. Vale ressaltar que o tempo médio de execução sem a utilização de threads foi de 0,208s.

Para essa abordagem com a integração da interface gráfica, foi adotado a programação orientada a objetos utilizando a classe gerada a partir da biblioteca PyQt5 e foi adicionado o método “Threads” que basicamente é toda a sequência lógica anterior, mas adaptada para ser utilizada com a POO.

4. CONCLUSÃO

Podemos concluir com este trabalho que na primeira abordagem, que pode ser conferido na **Figura 1**, obtivemos um maior sucesso de desempenho da máquina ao adicionar mais threads. O ganho de desempenho foi significado utilizando 7 threads. Contudo, ao tentar adaptar o código vigente e integrar com a interface gráfica, o resultado não foi tão positivo, como pode ser conferido na **Figura 2** onde podemos observar que o tempo de execução foi aumentando conforme o número de threads foi sendo ampliado, mesmo esse tendo uma entrada N de elementos igual a 1000, 10x menos do que no primeiro caso.

Uma dificuldade que foi encontrada ao adaptar o código para receber a GUI, foi que conforme o programa fosse criando novas threads, novas interfaces gráficas eram abertas, matando o desempenho do processador e forçando a reinicialização do computador. Isso foi driblado ao adotar a programação orientada a objetos (POO), onde foi observado que o programa não executava várias instâncias de janelas. Cabe ressaltar que não foi identificado o motivo desse comportamento acontecer, apenas foi notado e ao tentar várias abordagens, a de POO pareceu ser a mais correta.