**Disciplina**: Performance em Sistemas Ciberfisicos

**Professor:** Guilherme Schnirmann

**Nome Estudante:**

**Atividade Prática / Relatório**

# **Descrição da Atividade:**

Esta atividade tem como objetivo buscar o entendimento da criação de múltiplas threads e seus benefícios em um problema de processamento de imagens.

Utilize o código fonte disponibilizado no AVA que resolve o problema single-thread.

Deseja-se pintar todas as flores brancas:



Exercício:

Crie agora um método para executar o processamento da imagem multi-threaded:



**PROFESS RO EU UTILIZEI METODO DE RODAR O CODIGO 5X EM CADA QUANTIDADE DE THREAD PARA TIRAR A MEDIA**

Faça um código que permita criar quantas threads forem desejadas.

1. Meça o tempo (utilizando funções do java ou python) para executar o programa sem thread.
2. Agora meça o tempo utilizando somente uma thread. O tempo é maior ou menor? Explique.

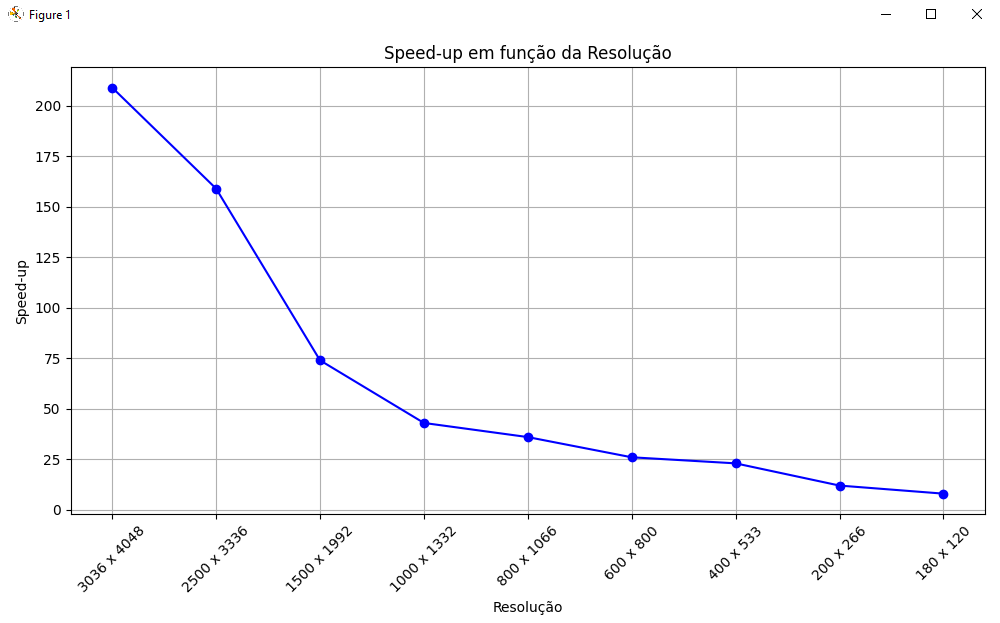
Foi 313 o tempo e isso faz dele um tempo maior para processamento

1. Agora meça o tempo utilizando duas threads e quatro threads. Explique os tempos, foram melhores?

Com duas Threads deu 237 e com quatro 226, Sim os tempos foram melhores por utilizar mais threads já que passou de 2 para 4 mas não foi uma diferença muito grande

1. Faça várias medidas e gere um gráfico relacionando o número de threads com o tempo. Apresente aqui o gráfico e explique a relação com o hardware do seu computador.

No caso as mult threads permite que o código distribui as threads em vários núcleos da CPU no caso se o computador tem bastante núcleos isso diminui o tempo de processamento, O número de threads pode ser ajustado pois deve escolher um número que corresponda ao número de núcleos de sua CPU para obter o melhor desempenho.

1. Agora fixe o número de threads no melhor número encontrado no gráfico anterior e gere um gráfico variando o tamanho da imagem (diminuindo). Para isso calcule o speed-up (em relação a execução single-thread). O gráfico será de speed-up X resolução.

Codigo utilizado para gerar o gráfico:

import matplotlib.pyplot as plt

resolucoes = [

"3036 x 4048",

"2500 x 3336",

"1500 x 1992",

"1000 x 1332",

"800 x 1066",

"600 x 800",

"400 x 533",

"200 x 266",

"180 x 120"

]

speed\_up = [209, 159, 74, 43, 36, 26, 23, 12, 8]

# Gerar o gráfico

plt.figure(figsize=(10, 6))

plt.plot(resolucoes, speed\_up, marker='o', linestyle='-', color='b')

# Adicionar título e rótulos

plt.title('Speed-up em função da Resolução')

plt.xlabel('Resolução')

plt.ylabel('Speed-up')

plt.grid(True)

plt.xticks(rotation=45)

plt.tight\_layout()

# Mostrar o gráfico

plt.show()

1. **Na sua entrega deve ser apresentada a imagem das flores pintadas por uma cor escolhida pela equipe (diferente da do professor).**



1. **Deve-se apresentar o gráfico de tempo x número de threads com o apontamento do melhor N (número de threads)**
2. **Deve-se apresentar o gráfico de speed-up x resolução ( chegue até uma resolução bem baixa – exemplo: 180x120).**

**Exercício teórico:**

Pesquise o que são deadlocks, demonstre sua resposta com um exemplo. Explique também as condições para acontecerem deadlocks e possíveis soluções.

Deadlocks são situações em sistemas computacionais onde dois ou mais processos ficam bloqueados indefinidamente porque cada um aguarda um recurso que está sendo usado por outro processo. Isso cria um impasse onde nenhum processo pode progredir, mesmo que recursos estejam disponíveis, devido à espera circular e à não preempção dos recursos. A prevenção e a resolução de deadlocks são essenciais para garantir a eficiência e a estabilidade de sistemas multitarefa.



**Para que ocorra um deadlock, geralmente são necessárias quatro condições simultâneas, conhecidas como as quatro condições de Coffman, que são:**

**Exclusão Mútua**: Pelo menos um recurso deve ser mantido exclusivamente por um processo e não pode ser acessado por outros enquanto estiver em uso.

**Posse e Espera**: Processos já alocados podem solicitar recursos adicionais e esperar indefinidamente por eles enquanto seguram os recursos já alocados.

**Não Preempção**: Recursos não podem ser retirados à força de um processo; eles devem ser liberados voluntariamente pelo processo que os possui quando não são mais necessários.

**Espera Circular**: Deve haver um conjunto de dois ou mais processos, onde cada processo está esperando por um recurso que é mantido pelo próximo processo no conjunto.

**Para que ocorra um deadlock, geralmente são necessárias quatro condições simultâneas, conhecidas como as quatro condições de Coffman, que são:**

**Exclusão Mútua**: Pelo menos um recurso deve ser mantido exclusivamente por um processo e não pode ser acessado por outros enquanto estiver em uso.

**Posse e Espera:** Processos já alocados podem solicitar recursos adicionais e esperar indefinidamente por eles enquanto seguram os recursos já alocados.

**Não Preempção**: Recursos não podem ser retirados à força de um processo; eles devem ser liberados voluntariamente pelo processo que os possui quando não são mais necessários.

**Espera Circular**: Deve haver um conjunto de dois ou mais processos, onde cada processo está esperando por um recurso que é mantido pelo próximo processo no conjunto.