Trabalho Prático: Acelerando o Processamento de Imagens com Programação Multithread

1. Introdução e Objetivos

Este trabalho prático tem como objetivo proporcionar aos estudantes uma experiência com programação concorrente, especificamente utilizando threads para otimizar o desempenho de uma aplicação. Em um cenário cada vez mais comum de processadores multi-core, a habilidade de desenvolver software que explore o paralelismo de hardware é uma competência essencial para engenheiros e engenheiras de software.

A atividade envolverá a alteração e a avaliação de um programa de processamento de imagens. Uma solução sequencial (single-thread) está disponível, em anexo. Esta solução deve ser refatorada para uma versão concorrente (multithread). A avaliação de desempenho comparativa entre as duas versões será o ponto central do trabalho, permitindo uma análise concreta dos ganhos (ou perdas) obtidos com multithreading.

Ao final deste trabalho, o estudante deverá ser capaz de:

- **Compreender e aplicar os conceitos fundamentais de threads:** criação, gerenciamento, sincronização e finalização.
- **Desenvolver um programa concorrente:** particionando uma tarefa computacionalmente intensiva para ser executada em paralelo por múltiplas threads.
- **Avaliar e analisar o desempenho de uma aplicação multithread:** medindo o tempo de execução e calculando métricas como o *speedup*.
- **Identificar os desafios da programação concorrente:** como condições de corrida e a necessidade de mecanismos de sincronização (embora o foco deste trabalho seja no paralelismo de dados, a conscientização desses desafios é importante).
- **Elaborar um relatório técnico:** apresentando e justificando as decisões de projeto, os resultados de desempenho e as conclusões obtidas.

2. O Problema: Filtro de Imagem em Escala de Cinza

O problema a ser resolvido é a aplicação de um filtro de conversão de uma imagem colorida para uma escala de cinza. A conversão de um pixel colorido para um pixel em escala de cinza pode ser realizada através da média ponderada dos seus componentes de cor (Vermelho, Verde e Azul - RGB). Uma fórmula comumente utilizada é:

$$\text{Cinza} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Para cada pixel da imagem original, este cálculo deve ser realizado, e o valor resultante de

"Cinza" será o novo valor para os três componentes (R, G e B) do pixel na imagem processada.

3. Especificações e Requisitos

3.1. Ambiente de Desenvolvimento

A solução inicial disponibilizada está implementada na linguagem Python, com uso da biblioteca PIL (Python Imaging Library).

Entretanto, esta é apenas uma implementação de referência. Os estudantes podem fazer uma nova implementação da solução em qualquer linguagem que preferirem. O essencial é utilizar programação multithread para tentar melhorar o desempenho.

3.2. Requisitos Funcionais

O programa single-thread deve ser alterado para:

1. Implementar uma versão **multithread** que realiza a mesma tarefa.
2. Na versão multithread, o número de threads a ser utilizado deve ser configurável (ex: via argumento de linha de comando).
    - Para a versão multithread, uma alternativa é dividir a imagem em porções horizontais (ou verticais) e atribuir cada porção a uma thread diferente. Por exemplo, se a imagem tem 1200 pixels de altura e o programa for executado com 4 threads, cada thread processará uma faixa de 300 pixels de altura.
3. Medir e exibir o tempo de execução (em milissegundos) tanto da versão sequencial quanto da multithread.

4. Etapas do Trabalho

Etapa 1: Implementação Concorrente (Multithread)

- Altere o código (`filtro_python.py`) para uma versão multithread.
- A lógica principal deverá criar e iniciar um número N de threads.
- Cada thread será responsável por processar uma fatia da imagem. É crucial calcular corretamente os índices de início e fim para cada thread, garantindo que a imagem inteira seja processada exatamente uma vez.
- A thread principal deve aguardar a finalização de todas as threads trabalhadoras antes de salvar a imagem resultante e finalizar a medição do tempo.

Etapa 3: Avaliação de Desempenho

- Execute o programa com diferentes números de threads (ex: 1, 2, 4, 8, 16 - dependendo do número de núcleos do processador da máquina de teste).
- Para cada configuração de threads, execute o processamento múltiplas vezes (ex: 5 vezes) e calcule o tempo médio de execução para obter resultados mais estáveis.
- Execute os testes com pelo menos 3 imagens de resoluções distintas (alta, média e baixa). Seguem alguns exemplos de imagens no formato JPG.

5. Análise e Relatório Técnico

O resultado final do trabalho será a entrega do código-fonte e de um relatório técnico em formato PDF. O relatório deve conter as seguintes seções:

1. **Introdução:** Breve descrição do problema e dos objetivos do trabalho.
2. **Arquitetura da Solução:** Explicação de como versão inicial foi modificada para para uso de múltiplas threads. Inclua um diagrama ou pseudocódigo que ilustre a estratégia de divisão do trabalho entre as threads.
3. **Ambiente de Teste:** Descrição do hardware e software utilizados para os testes (modelo do processador, número de núcleos, sistema operacional, versão da linguagem de programação/compilador).
4. **Resultados:** Apresentação dos tempos de execução coletados em tabelas. Calcule o **Speedup** para cada configuração multithread em relação à versão sequencial. O speedup é calculado como:

   $$Speedup(N) = \frac{Tempo\ de\ Execução\ Sequencial}{Tempo\ de\ Execução\ com\ N\ threads}$$
5. **Análise dos Resultados:**
   - Plote gráficos de "Tempo de Execução vs. Número de Threads" e "Speedup vs. Número de Threads".
   - Discuta os resultados obtidos. O speedup aumentou linearmente com o número de threads? Houve algum ponto em que adicionar mais threads não trouxe melhoria de desempenho ou até mesmo piorou?
   - Relacione os resultados com o hardware utilizado. O speedup foi limitado pelo número de núcleos físicos do processador?
   - Discuta o conceito de *overhead* (sobrecarga) na criação e gerenciamento de threads e como ele pode ter impactado os resultados, especialmente com um número muito grande de threads.
6. **Conclusão:** Apresentar um breve resumo dos aprendizados obtidos com o trabalho prático e as principais conclusões sobre a utilização de multithreading para otimização de desempenho.

6. Critérios de Avaliação

- **Funcionalidade (40%):** O código compila e executa sem erros. A versão multithread produz a imagem em escala de cinza e calcula o tempo corretamente. O gerenciamento das threads foi implementado de forma correta.
- **Qualidade do Código (20%):** Código bem estruturado, comentado e legível. Boas práticas de programação foram seguidas.
- **Relatório Técnico (40%):** Clareza na apresentação das ideias, profundidade da análise dos resultados, formatação adequada e ausência de plágio. A capacidade de analisar criticamente os dados de desempenho é um fator chave nesta avaliação.