

Conversor de Imagens Coloridas para Escala de Cinza
Riquelme Freitas Gomes
Vitória de Souza Serafim)

Relatório Final
Programação Concorrente (ICP-361) — 2024/1

1

1. Descrição do problema

O presente trabalho aborda o problema de converter imagens coloridas para escala de cinza. Tal conversão é uma das principais operações quando estamos falando de processamento de imagens. A ideia principal se baseia em transformar cada pixel de uma imagem colorida em um valor único de cinza, manipulando os valores dos canais Vermelho, Verde e Azul representados em uma imagem.

Há várias formas de realizar essa conversão, porém neste projeto optamos por utilizar um modo em que o programa percorre cada pixel da imagem, calcula a média dos valores RGB e substitui o valor original do pixel pela média ponderada calculada, resultando finalmente em uma imagem em escala de cinza. Nesse caso, cada pixel depende apenas de seus próprios valores RGB para o cálculo dessa média.

De forma mais específica, estamos utilizando o método da luminosidade em relação aos pixels de uma imagem. Tal método refere-se à maneira como o brilho de cada pixel é calculado com base nos valores de intensidade de luz nas cores vermelha, verde e azul (RGB). Por exemplo, uma fórmula comum, e a qual utilizamos, para a luminosidade ponderada é

$$\text{Luminosidade} = [0,299 \times R + 0.587 \times G + 0.114] \times B \quad (1)$$

atribuindo maior peso ao componente verde devido à maior sensibilidade do olho humano a essa cor.

Dessa forma, o dado de entrada do nosso problema consiste em um arquivo de uma imagem colorida, e a saída é a mesma imagem convertida para escala de cinza, de modo que a imagem é processada como uma matriz de pixels, onde cada pixel contém valores para as componentes R, G e B.

Uma maneira de tentar otimizar essa tarefa é implementando uma solução concorrente para esse problema. Tal implementação permite a paralelização da tarefa de calcular o valor 'cinza' dos pixels da imagem, onde cada pixel pode ser processado de forma independente, permitindo que a tarefa seja dividida entre múltiplas threads de execução. Assim, por meio do paralelismo pretendemos reduzir o tempo desse processamento da imagem, principalmente para imagens de alta maiores.

Concluindo, ao longo do trabalho, discutiremos a implementação detalhada das soluções sequencial e concorrente, analisando a estratégia de divisão de tarefas. Além disso, realizaremos testes para avaliar a corretude e o desempenho das solução proposta, utilizando a linguagem C e a biblioteca pthread para implementar a concorrência.

2. Projeto e implementação da solução concorrente

A ideia de usar uma solução concorrente é dividir a tarefa principal de calcular o valor cinza de cada pixel em várias etapas distintas. Assim, a imagem será dividida em blocos ou seções, com cada bloco sendo associado a uma thread distinta, o que facilitará o cálculo dos valores da imagem entre as threads. A estratégia escolhida foca na divisão da imagem em blocos horizontais (linhas), pois essa abordagem facilita o balanceamento das cargas de trabalho entre as threads e evita problemas que poderiam surgir com a separação em blocos irregulares, tornando o processo mais organizado.

Fazendo com que cada thread processe um conjunto de linhas da matriz imagem, temos duas abordagens possíveis: uma em que as threads processam linhas consecutivas e outra em que as threads se alternam entre as linhas. Nesse trabalho, estamos utilizando a primeira opção com o objetivo de fazer com que o acesso à memória seja mais eficiente pois cada thread trabalha em uma área contígua da imagem. Assim, buscamos tentar melhorar a performance devido à melhor utilização do cache.

Como forma de lidar com os casos em que não ocorre uma divisão exata das linhas da matriz pelas threads, optamos por fazer com que a última thread receba também as linhas excedentes, garantindo que todas as linhas sejam processadas.

Portanto, em nossa implementação, a abordagem consecutiva é utilizada, o que é apropriado para muitas situações, especialmente quando a imagem é relativamente homogênea em termos de complexidade de processamento. A vantagem principal aqui é a simplicidade e eficiência na utilização do cache.

3. Casos de teste

Para realizar os testes utilizamos 4 imagens coloridas de disponibilizadas pelo site "Free-pik". Tais imagens possuem diferentes dimensões para que possamos construir testes que contenham desde pequenas até imagens maiores. São elas:



Figura 1. Dimensões: 500 x 750



Figura 2. Dimensões: 2000 x 3000



Figura 3. Dimensões: 3500 x 3500



Figura 4. Dimensões: 5000 x 3333

A fim de testar a corretude de nossa implementação concorrente, comparamos diversos resultados obtidos nessa com os resultados da versão sequencial, que, nesse contexto, assumimos como o resultado correto. Isto é, comparamos a saída do programa concorrente com a saída da versão sequencial com o uso do comando "diff", o qual compara o conteúdo de dois arquivos linha por linha. Para todas as imagens utilizadas, inclusive as 4 consideradas nesse relatório, atestamos que não houveram diferenças entre os valores dos pixels das imagens convertidas por cada abordagem. Assim, garantimos que ambas garantindo que geraram a mesma imagem em escala de cinza.

4. Avaliação de desempenho

Para os testes de desempenho, rodamos nossos algoritmos sequencial e concorrente para as quatro imagens. Nos casos concorrentes, cada imagem foi convertida utilizando quatro número de threads distintos (1, 2, 4 e 8). Cada configuração foi executada 4 vezes, estimando o tempo gasto no processamento em cada execução e calculando o tempo médio entre esses.

A partir das medidas dos tempos médios, calculamos a aceleração (A) e a eficiência (E) alcançadas por meio da concorrência. Tais valores estão sendo calculados da seguinte forma:

$$A_p = \frac{\text{tempo de processamento sequencial}}{\text{tempo de processamento concorrente com } p \text{ threads}} \quad (2)$$

Para o cálculo da aceleração, optamos por utilizar apenas o tempo de processamento e não o tempo total da execução do programa. Acreditamos que dessa forma é possível observar mais precisamente o impacto da concorrência na resolução do problema, uma vez que as partes de “inicialização” e “finalização” do código ocorrem da mesma maneira independente da abordagem.

$$E_p = \frac{\text{Aceleração}}{\text{Número de threads } p} \quad (3)$$

Abaixo estão os resultados obtidos em forma de tabela para cada imagem abordada.

Tabela 1. Dados gerados a partir da Figura 1

Threads	Tempo médio	Aceleração	Eficiência
Sequencial	0,01068775		
1	0,01302225	0,8207299046	0,8207299046
2	0,006889	1,551422558	0,7757112789
4	0,00369525	2,892294161	0,7230735404
8	0,00369575	2,891902861	0,3614878577

Tabela 2. Dados gerados a partir da Figura 2

Threads	Tempo médio	Aceleração	Eficiência
Sequencial	0,181515		
1	0,20017	0,9068042164	0,9068042164
2	0,1020375	1,778904814	0,8894524072
4	0,0539125	3,366844424	0,841711106
8	0,0539265	3,365970349	0,4207462936

Tabela 3. Dados gerados a partir da Figura 3

Threads	Tempo médio	Aceleração	Eficiência
Sequencial	0,35376025		
1	0,41207825	0,8584783351	0,8584783351
2	0,209045	1,692268411	0,8461342056
4	0,110763	3,193848578	0,7984621444
8	0,1015075	3,485065143	0,4356331429

Tabela 4. Dados gerados a partir da Figura 4

Threads	Tempo médio	Aceleração	Eficiência
Sequencial	0,47719425		
1	0,55381425	0,8616503638	0,8616503638
2	0,2805765	1,700763428	0,8503817141
4	0,1577788	3,02427466	0,7560686649
8	0,130785	3,648692511	0,4560865638

Tempo médio para diferentes imagens e número de threads

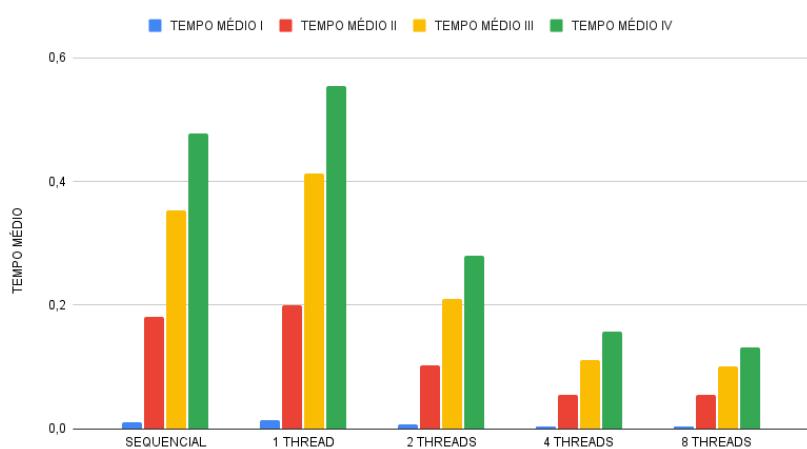


Figura 5. Tempo médio obtido através dos dados gerados

Aceleração para diferentes imagens e número de threads

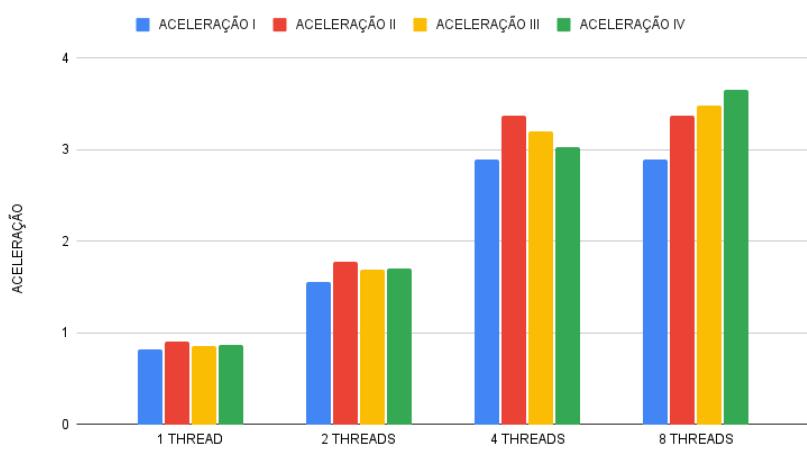


Figura 6. Aceleração obtida através dos dados gerados

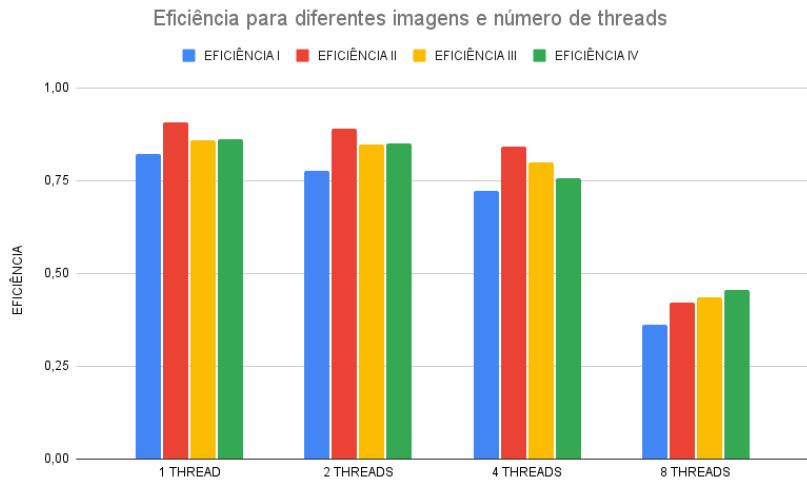


Figura 7. Eficiência obtida através dos dados gerados

Nesse projeto, utilizamos uma máquina com capacidade de processamento 2,3 GHz, Quad-Core e Chipset Intel Core i5. Além disso, a máquina é equipada com a memória gráfica Intel Iris Plus Graphics 655 1536 MB e Memória RAM de 8 GB 2133 MHz LPDDR3, 8 Núcleos.

5. Discussão

Análise dos Gráficos

Os gráficos mostram como o uso de threads afeta a aceleração, o tempo médio de execução e a eficiência do programa que converte imagens para escala de cinza. Podemos observar como cada gráfico ajuda a entender melhor como se deu o desenvolvimento do trabalho, com isso vamos analisar:

- **Tempo Médio**

O tempo médio de execução é mais baixo para a primeira imagem, que tem um tempo quase insignificante. Isso é previsto, pois a Figura I utilizada nos testes possui tamanho muito inferior comparado às outras. Com uma thread, o tempo médio aumenta um pouco, especialmente para a imagem IV, chegando a 0,5 segundos. Com duas threads, o tempo médio começa a diminuir, mas ainda é maior para a imagem IV. A utilização de quatro threads continua a diminuir o tempo médio, com as imagens I, II e III apresentando tempos bastante baixos, enquanto a imagem IV ainda também reduz bastante seu tempo médio. Com oito threads, o tempo médio se mostra ser o menor entre todos. Nesse caso, quanto mais threads utilizadas menores foram ficando os tempos médios para todas as imagens, em proporções diferentes.

- **Aceleração**

A taxa de aceleração aumenta com a adição de threads para todas as imagens. Com uma thread, a aceleração é menor que 1, o que é esperado, pois não há paralelismo real. Com duas threads, a aceleração começa a mostrar melhorias, atingindo cerca de 1,5. A adição de quatro threads resulta em uma aceleração significativa, aproximando-se de 2,5 para a imagem I e ultrapassando este valor para as outras imagens. Quando usamos

oito threads, a aceleração atinge seu pico, especialmente para a imagem IV, que chega a quase 4. As outras imagens mostram acelerações entre 2,5 e 3,5. Isso indica que o paralelismo melhora o desempenho, mas a aceleração não é linear devido à sobrecarga de sincronização.

- **Eficiência**

A eficiência é alta com uma thread, próxima de 1 para todas as imagens. Com duas threads, a eficiência diminui, mas permanece relativamente alta (entre 0,75 e 0,8), indicando que o paralelismo está mostrando seus benefícios, mas a sobrecarga é evidente. A adição de quatro threads continua a diminuir a eficiência (entre 0,6 e 0,8), mostrando que, embora o ganho de paralelismo esteja presente, a sobrecarga adicional começa a impactar mais significativamente. Com oito threads, a eficiência é a mais baixa (entre 0,4 e 0,6), indicando que a sobrecarga de gerenciamento e sincronização de threads está afetando significativamente os ganhos de paralelismo do código.

Conclusão

O ganho de desempenho alcançado com o uso de múltiplas threads é significativo, especialmente em comparação com o programa sequencial. A maior aceleração é observada com oito threads, particularmente para a imagem IV, o que sugere que o uso de múltiplas threads é benéfico. No entanto, a eficiência diminui com o aumento do número de threads devido à sobrecarga de gerenciamento e conflito entre threads, indicando um ponto de rendimento decrescente.

O comportamento observado é consistente com o esperado: o paralelismo melhora o desempenho, mas com eficiência decrescente à medida que mais threads são adicionadas. A complexidade da imagem IV destaca que algumas tarefas podem não se beneficiar tanto do paralelismo quanto outras devido a características específicas.

Melhorias

Como possíveis melhorias para o programa, poderíamos considerar ainda a forma alternada de distribuir as threads entre as linhas da matriz, a fim de comparar com os resultados que obtivemos. Assim, seria possível perceber se alguma das duas abordagens, nesse contexto, tem maior impacto no desempenho. De forma análoga, podemos também considerar outras formas de realizar a conversão de imagens para escala de cinza, ranqueando-as e destacando aquelas que têm melhores resultados quando com concorrência. Por fim, poderíamos também utilizar uma maior quantidade de imagens para os testes, e até mesmo utilizar maiores valores para o número de threads de modo a deixar os dados gerados cada vez mais específicos e precisos.

6. Referências bibliográficas

1. Convertendo Imagem Colorida para Tons de Cinza, Marcelo Teixeira Silveira, 2008
2. Color to Grayscale Image Conversion Based on Singular Value Decomposition
3. Bibliotecas stb de domínio público de arquivo único para C/C++ - Tratamento de imagens
4. Fotos Animais Coloridos- FreePik