Sistemas Multinível e Distribuição
Mestrado em Engenharia Inform tica
Instituto Superior de Engenharia do Porto

Processamento e análise paralelos de imagens digitais

Introdução

O objetivo deste trabalho é implementar ferramentas de processamento de imagem usando abordagens sequenciais, multi-thread e baseadas em threadpool, para estudar como maximizar eventuais ganhos de desempenho em cenários onde a programação paralela e concorrente pode alavancar o potencial do hardware, e explicar como e porquê cada uma destas abordagens tem impacto no desempenho da solução.

Filtros de imagem

Existem vários f i l t r o s de imagem disponíveis no software de edição de fotografias. Estes lters transformam uma imagem de entrada numa imagem de saída em que pelo menos alguns pixéis são convertidos através da aplicação de algum cálculo. Exemplos são: converter imagens de cor para escala de cinzentos, aumentar o brilho ou aplicar algum tipo de distorção. Para este projeto vamos usar os lters descritos nas secções seguintes e note-se que vamos considerar as imagens como matrizes de pixéis (objectos de cor) e será fornecida uma API com funcionalidades simples de processamento de imagem.

Brigthness

Os valores de vermelho, verde e azul de cada pixel estão no intervalo de 0 a 255. O aumento do brilho é conseguido através do aumento deste valor. Assim, para a imagem de entrada 1, aumentar o valor dos pixéis em 64 resulta na imagem 2.

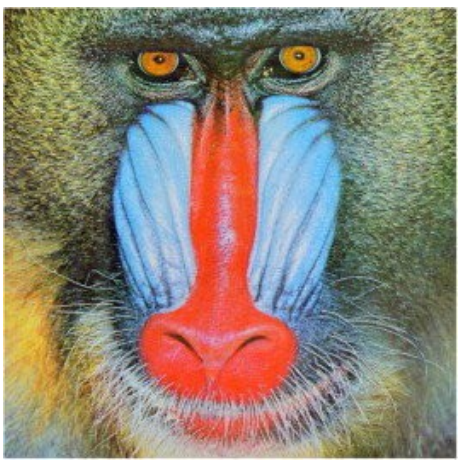

Figura 1: Fotografia original

Figura 2: Carta mais brilhante

Escala de cinzentos

A conversão para escala de cinzentos pode ser efectuada por diferentes métodos. O método aqui sugerido consiste em calcular a média dos valores de vermelho, verde e azul de cada pixel e substituir cada um deles por esse valor. Mais formalmente, para cada pixel com valores de vermelho (r), verde (g) e azul (b), calcule:

$$\mathsf{A} = \frac{(r + g + b)}{3}$$

Substitua cada um dos r, g e b por A. Assim, para a entrada gure 1, a aplicação deste método resulta na gure 3.

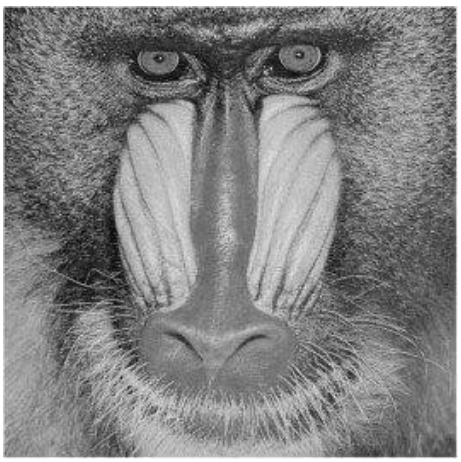

Figura 3: Filtro em escala de cinzentos

Redemoinho

O swirl lter ou warp lter consiste em rodar a imagem como se apresenta na figura 4. Para o efeito, pode-se utilizar a fórmula explicada√a seguir. Dado o centro da imagem $(x_0 , y)_0$ e dado qualquer ponto (x, y) na imagem, $d = (x - x_0)^2 + (y - y_0)^2$ e $\theta = \frac{\pi}{256} * d$ substituir os valores de x e y pelos valores de x' e y' de modo que:

$$x' = (x - x_0) \cos\theta - (y - y_0) \sin\theta + x_0$$
$$y' = (x - x_0) \sin\theta + (y - y_0) \cos\theta + y_0$$

Assim, para a entrada gure 1, a aplicação deste método resulta na gure 4.

Figura 4: Filtro de turbilhão

Vidro

O filtro de vidro troca um pixel por um vizinho próximo aleatório. O resultado é que a imagem parece ser exibida através de um vidro. Mais formalmente, podemos conseguir isto substituindo cada pixel (*x, y*) por um vizinho próximo aleatório com uma distância máxima de, por exemplo, 5 pixéis. Assim, para a entrada gure 1, a aplicação deste método resulta na gure 5.

Figura 5: Filtro de vidro

Borrão

Aqui, a ideia é implementar um filtro para desfocar uma imagem. A desfocagem de uma imagem pode ser feita calculando a média dos pixéis com os valores dos seus vizinhos.

Assim, se, por exemplo, tivermos o pixel

$(x, y) = (113, 91, 94)$ em que 113 é o valor para o vermelho, 91 o valor para o verde e 94 o valor para o azul, e os seus vizinhos são os descritos no quadro 1.

(142,113,115)	(150,120,120)	(157,127,127)
(113,91,94)	(113,91,94)	(123,101,104)
(129,108,113)	(128,107,112)	(133,112,117)

Tabela 1: Matriz 3x3 do conteúdo dos píxeis com centro em (113,91,94)

Agora, os valores para o vermelho, o verde e o azul são os seguintes:

$$r = \frac{142 + 150 + 157 + 113 + \mathbf{113} + 123 + 129 + 128 + 133}{9} = 132$$

$$g = \frac{113 + 120 + 127 + 91 + \mathbf{91} + 101 + 108 + 107 + 112}{9} = 107$$

$$b = \frac{115 + 120 + 127 + 94 + \mathbf{94} + 104 + 113 + 112 + 117}{9} = 110$$

Como resultado da operação, o pixel com $(x, y) = (113, 91, 94)$, deve ser substituído por $(x, y) = (132, 107, 110)$. Quanto maior for a matriz de pixéis utilizada, mais desfocada fica a imagem. Deve ser possível escolher um valor m e criar o ect correspondente para cada pixel usando uma submatriz $m \times m$. Para as arestas, onde não se dispõe de uma matriz completa, devem usar-se apenas os valores disponíveis. Para a entrada gure 1, a aplicação deste método resulta na gure 6.

Figura 6: Filtro de desfocagem

Desfoque condicional

A desfocagem condicional consiste em aplicar a desfocagem apenas quando alguma condição é satisfeita. Por exemplo, se um pixel tiver uma certa quantidade de vermelho, será desfocado; caso contrário, ignoramos esse pixel. Se definirmos o limiar de vermelho como, por exemplo, 200, um pixel $(x, y) = (r, g, b)$ em que $r > 200$ será aplicado o filtro, caso contrário será ignorado. Um exemplo em que apenas os pixéis vermelhos são desfocados é apresentado na figura 7.

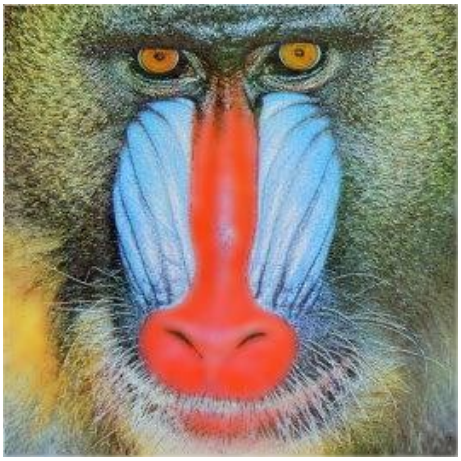

Figura 7: Filtro de desfoque condicional

Implementação sequencial

A primeira tarefa é criar uma implementação sequencial dos lters. É fornecida uma API simples para traduzir imagens para matrizes de objectos de cor e vice-versa, bem como para ler e escrever imagens de e para o sistema. Existem também referências que podem ajudar a compreender estas implementações, por exemplo, o livro Digital Image Processing 4th edition [1] e o livro Computer Science: An Interdisciplinary Approach [2] e o sítio Web do antigo livro[1] .

Implementação multithread

A segunda parte deste projeto é a criação de uma implementação multithread dos lters onde as imagens são divididas e repartidas por diferentes threads que deverão otimizar a aplicação dos lters em arquitecturas multicore. A definição do número de threads e da quantidade de trabalho para cada uma delas faz parte dos objectivos do projeto, pelo que a estratégia de decomposição de dados e as estruturas de dados envolvidas devem ser cuidadosamente escolhidas, compreendidas e analisadas para obter o melhor desempenho possível para cada cenário.

Implementação baseada em Threadpool

A terceira parte deste projeto é usar threadpools para resolver o problema. O que significa que a criação e desmontagem de threads deve ser da responsabilidade de uma threadpool. A quantidade de trabalho a ser enviada para o threadpool deve ser cuidadosamente analisada. Nesta parte do projeto, devem ser abordadas e analisadas diferentes utilizações de threadpool, nomeadamente, baseadas em Executor, baseadas em ForkJoinPool e, ao mais alto nível, utilizando CompletableFutures. Note-se que a utilização de estratégias como o work-stealing só é possível em alguns tipos de threadpools e, portanto, dependendo do cenário das aplicações, diferentes threadpools podem ter resultados de desempenho diferentes e é um objetivo primordial compreender estas diferenças.

[1] https://introcs.cs.princeton.edu/java/31datatype/

Afinação do coletor de lixo

Java fornece diferentes colectores de lixo que funcionam em simultâneo com as nossas aplicações e que limpam a memória utilizando diferentes estratégias. Alguns dividem a área de memória em partes, utilizam uma ou mais threads, etc. É possível escolher e configurar colectores de lixo em Java, devendo ser fornecidas provas desse trabalho e justificada a escolha de um coletor de lixo adequado.

Geração de resultados

A última parte deste projeto consiste em medir e analisar os resultados das diferentes abordagens. A solução desenvolvida deverá medir o tempo de execução da aplicação dos lters para várias imagens com tamanhos diferentes em cada uma das abordagens diferentes. Todos os dados devem ser reunidos em tabelas geradas automaticamente e os resultados devem ser usados no relatório final para explicação dos resultados.

Código de iniciação

É fornecido um código de iniciação. Inclui uma API simples para as imagens e um exemplo da aplicação de um dos filtros. A API é descrita de seguida:

Color[][] loadImage(String lename) : dado o caminho para uma imagem le em lename devolve uma matriz Color[][] de pixels dessa imagem.

void writeImage(Color[][] image,String lename) : dada uma matriz de objectos de cor e um caminho para um ficheiro em lename, escreve a matriz nesse ficheiro criando uma imagem no sistema de ficheiros.

Color[][] copyImage(Color[][] image) : cria e devolve uma cópia de uma matriz de objectos Color. Isto é útil quando é necessário trabalhar sobre uma cópia da imagem.

O ficheiro ApplyFilters.java utiliza a API simples descrita anteriormente para criar um objeto Filter que lê uma imagem do lesystem, aplica o filtro Brighter a essa imagem e escreve o resultado de volta no lesystem.

Classificação

O projeto deve ser resolvido individualmente ou em grupos de, no máximo, quatro (4) elementos.

Descrição	Percentagem (%)
Implementação de filtros de imagem	15
Processamento baseado em multithread	20
Processamento baseado em Threadpool	30
Análise dos resultados e discussão (inclui relatório final)	35

Programação

O prazo final para a apresentação deste projeto é 28th de abril de 2024 e as apresentações terão lugar nos dias seguintes. Haverá uma apresentação opcional e parcial na semana de

8th de abril para feedback sobre o progresso do projeto.

Código de honra

No "C digo de Boas Pr ticas de Conduta", de 27 de outubro de 2020, é indicado que os estudantes devem apresentar uma declaração, tal como descrito no Artigo 8, para cada projeto apresentado. Esta declaração é um compromisso assinado de originalidade do trabalho apresentado. A não apresentação desta declaração retirará o trabalho da avaliação. A apresentação de um trabalho que não cumpra a declaração assinada terá consequências legais.

Referências

[1] R.C. Gonzalez e R.E. Woods. Processamento digital de imagens. Pearson, 2018.

[2] Robert Sedgewick e Kevin Wayne. Ciência da Computação: Uma Abordagem Interdisciplinar. Addison-Wesley Professional, 1ª edição, 2016.