Paralelismo em algoritmos de detecção de adulterações em imagens digitais

Vitor Filincowsky Ribeiro, Jefferson Chaves Gomes, Felipe Lopes de Souza

[1]Instituto de Ciências Exatas
Departamento de Ciência da Computação - CIC
Universidade de Brasília (UnB) - Brasília, DF - Brasil

vribeiro@cic.unb.br, {jefferson.chaves, felipelopess}@gmail.com

Abstract. *Escrever o abstract*

Resumo. *Escrever o resumo*

1. Introdução

A necessidade de se representar objetos, pessoas e experiências em arquivos de imagem tem crescido sobremaneira na última década. A popularização destes arquivos no formato digital impulsionou o surgimento de diversas aplicações para processamento de imagens e vídeos digitais, que são de fundamental importância em áreas como a neurociência, controle de acesso, identificação criminal e até mesmo em redes sociais.

Algumas aplicações permitem que usuários manipulem à revelia qualquer imagem digitalizada à qual tenham acesso, e mesmo leigos podem produzir falsificações imperceptíveis à análise visual. Adulterações em imagens digitais são motivo de preocupação, pois estas extrapolam o mero uso doméstico e são exploradas de maneira maliciosa em várias áreas, como na política, medicina, propaganda ou na execução (ou ocultação) de atividades criminosas [**?**].

Nesta pesquisa, é proposto o estudo de mecanismos para verificação de autenticidade de imagens digitais. Sabendo que esta tarefa possui alto custo computacional, deseja-se explorar o paralelismo de código e o processamento distribuído na implementação da aplicação desenvolvida para detecção da fraude, a fim de que seja possível avaliar o desempenho da mesma quando executada de maneira serial e paralelizada.

O objetivo geral deste trabalho é avaliar a performance resultante da paralelização e distribuição de um algoritmo de identificação de adulterações em imagens que sofreram tratamento de *copy-move forgery*. Esta é uma técnica de adulteração onde um fragmento de uma imagem digital é replicado sobre ela mesma a fim de ocultar ou multiplicar objetos existentes [**?**]. Em particular, é utilizado um algoritmo de *shift vector* [**?**]. Os objetivos específicos são:

- Desenvolver uma aplicação para detecção de fraudes do tipo *copy-move forgery* em uma imagem digital com a utilização de um algoritmo de *shift vector*;
- Utilizar técnicas de paralelismo e processamento distribuído de tarefas no desenvolvimento desta aplicação;
- Avaliar o desempenho na execução em cada cenário de implementação da aplicação desenvolvida.

O trabalho é organizado como se segue: a Seção **??** apresenta as duas técnicas utilizadas para a obtenção do paralelismo na aplicação desenvolvida. A Seção **??** versa sobre o tratamento de imagens digitais. Na Seção **??** são descritos os passos para a implementação da aplicação proposta. As simulações e seus resultados são descritos na Seção **??** e, por fim, a Seção **??** apresenta as considerações finais sobre o trabalho desenvolvido.

2. Processamento paralelo

Com a expectativa de melhora no desempenho, serão utilizadas duas técnicas para execução paralela em implementações incrementais da aplicação [**?**].

O OpenMP (*open multiprocessing*) consiste em uma biblioteca que incrementa o compilador C para implementar o comportamento paralelo em programas com acesso a áreas de memória compartilhada [**?**]. OpenMP foi desenvolvido a fim de proporcionar comportamento *multithreaded* a programas de alto desempenho, porém em um nível mais alto do que ocorre em outras APIs como Pthreads. Deste modo, programas com execução serial podem ser incrementalmente paralelizados [**?**].

O MPI (*Message-Passing Interface*) é um padrão de interfaces portável para escrever programas paralelos utilizando o modelo de memória distribuída [**?**]. Não é um *framework*, mas sim um padrão de interfaces, independentes de linguagem, com foco em transmissão de mensagens entre programas paralelos que executam em ambientes multiprocessados, com computação paralela e memória distribuída. Os dados são movidos de um espaço de endereçamento a outro através de operações cooperativas como *send/receive*; portanto, não existe acesso a áreas de memória compartilhada, mas sim comunicação coletiva entre processos [**?**].

2.1. Medidas de desempenho

O *speed-up* é a medida de desempenho dada pela razão entre os tempos de execução do programa serializado e do programa paralelizado, conforme equação **??**.

$$S = \frac{t_{serial}}{t_{parallel}} \tag{1}$$

A eficiência de um programa paralelizado é medida pela razão entre o *speed-up* e a quantidade de núcleos do processador, conforme equação **??**.

$$E = \frac{S}{p} = \frac{t_{serial}}{p * t_{parallel}} \tag{2}$$

3. Processamento de imagens digitais

3.1. Detecção de adulterações

Os mais influentes trabalhos na área de detecção de adulterações em imagens exploram distintas técnicas para cada modalidade de falsificação. [**?**] dividem a imagem em blocos de tamanho fixo e os blocos são comparados dois-a-dois para a detecção de blocos idênticos. No entanto, o custo computacional desta abordagem é proibitivo.

[**?**] propõem um algoritmo que simplifica a análise espacial da imagem, dividindo-a em blocos e extraindo um vetor de características para cada bloco. Estes vetores, que carregam informações sobre cores, níveis de cinza e deslocamento dos blocos, são utilizados para comparação e identificação de regiões similares. Este trabalho é a base para a implementação da técnica exposta neste documento.

3.2. Operações morfológicas

Processamento morfológico é a operação na qual a forma espacial ou estrutura de objetos na imagem são modificados [**?**]. As três principais operações morfológicas de imagens são:

- *Dilatação*: um objeto cresce uniformemente em extensão espacial;
- *Erosão*: um objeto encolhe uniformemente em extensão espacial;
- *Esqueletização*: representação do objeto em uma figura linear.

Dilatação e erosão são consideradas operações do tipo *hit-or-miss* [**?**]. Primeiramente, a imagem é transformada em uma imagem binária (apenas pixels pretos e brancos). Em seguida, é utilizada uma máscara matricial (geralmente 3x3) para escanear a imagem. Se o padrão da máscara corresponder aos pixels sob ela, é gerado um pixel de saída espacialmente correspondente ao pixel central da máscara, que por sua vez é configurado para o estado binário desejado. Se o padrão não corresponder, o pixel de saída é transformado para o estado binário oposto. Exemplos destas operações podem ser vistos na Figura **??**.

4. Aplicação desenvolvida

A aplicação provê a detecção de *copy-move forgery*. A detecção de tal adulteração é feita com o algoritmo *robust shift vector*, que consiste na identificação de similaridades entre regiões distintas por meio da análise dos vetores de características. Estes vetores carregam informações sobre cores, níveis de cinza e deslocamento dos blocos [**?**].

4.1. Metodologia de implementaçao

Devido à natureza da duplicação de regiões, haverá ao menos duas regiões similares em uma imagem adulterada [**?**]. Assim, dada uma imagem $D = f(x, y)$, a imagem adulterada $D' = f'(x, y)$ consistirá do seguinte: $f'(x, y) = f(x, y)$ se (x, y) não pertence a D_2 e $f'(x, y) = f(x - dx, y - dy)$ se (x, y) pertence a $D2$, onde $D2 = D1 + d$, conforme Figura **??**-a.

Os passos necessários para o algoritmo de *shift-vector* na classificação de *copy-move forgery* são divididos em três fases: extração dos vetores característicos dos blocos, busca por blocos similares e eliminação de falso-positivos.

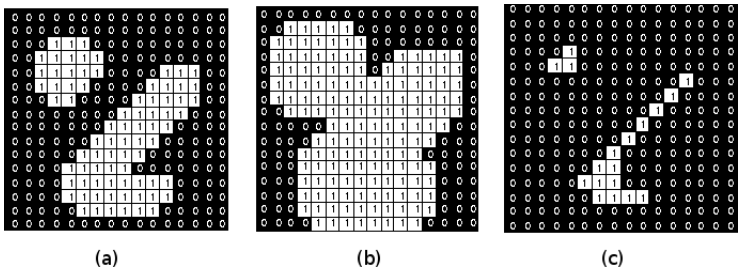

Figure 1. (a) Imagem original; (b) Imagem dilatada; (c) Imagem erodida.

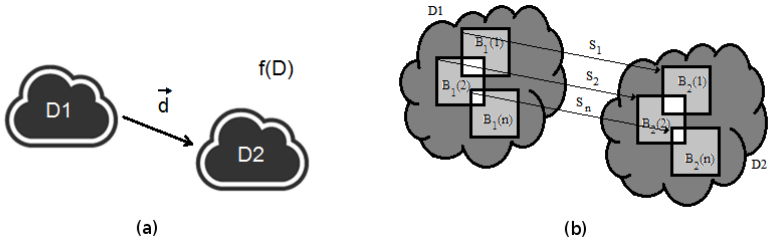

Figure 2. (a) Duplicação de regiões na imagem D; (b) Vetor de deslocamento entre blocos de regiões similares

A imagem de entrada é subdividida em blocos de dimensões bxb. Ao total, a imagem, de dimensões MxN, será descrita por $S = (M - b + 1)x(N - b + 1)$ blocos. Para cada bloco é calculado um vetor de características $V = (c_1, ..., c_7)$, onde $c_1...c_3$ são as médias dos valores RGB e $c_4...c_7$ são uma normalização dos pixels da imagem para um tom de cinza.

O bloco é então dividido em duas partes, sendo considerado em quatro direções de luminância (vertical, horizontal e duas diagonais). Cada bloco é calculado e adicionado a uma matriz com as linhas lexicograficamente ordenadas. A adição já garante a ordenação, o que evita que a matriz seja pós-processada para este propósito. Os blocos consecutivos são comparados dois-a-dois e é calculada a similaridade entre eles. A similaridade entre os blocos B_i e B_j é calculada pela diferença entre seus vetores V_i e V_j, sendo que $Diff(k) = |c_k(i) - c_k(j)|$. Dois blocos são considerados similares se quatro condições básicas são satisfeitas, onde os valores $P(k)$, t_1, t_2 e L são limiares pré-definidos:

- $Diff(k) < P(k)$
- $Diff(1) + Diff(2) + Diff(3) < t1$
- $Diff(4) + Diff(5) + Diff(6) + Diff(7) < t2$
- Vetor deslocamento entre B_i e B_j é menor do que L.

Este vetor deslocamento (*shift*) é dado por $d' = (dx, dy)$, onde $dx = x_i - x_j$, $dy = y_i - y_j$, (x_i, y_i) e (x_j, y_j) são os pixels do canto superior de B_i e B_j, respectivamente.

Em uma imagem adulterada, presume-se que um determinado conjunto de blocos tenha sido copiado para outra região da imagem. Deste modo, os seus vetores de deslocamento serão idênticos (ver Figura **??**-b). No entanto, o fato de dois blocos serem similares não implica que os mesmos pertencem a regiões duplicadas. É então gerado um histograma para hierarquizar todos os vetores de deslocamento entre os blocos similares e armazenar a frequência de ocorrência de cada vetor de deslocamento. Se os vetores são similares mas estão abaixo de um limiar, eles são classificados um falso positivo e são eliminados da matriz.

A lista definitiva de blocos similares é obtida e a imagem é convertida para preto e branco (binária), onde a cor preta é atribuída a pixels originais e a cor branca é atribuída aos pixels pertencentes aos blocos similares. A operação de abertura da imagem (erosão + dilatação) é aplicada para a eliminação de elementos espúrios da imagem. A imagem de entrada é mesclada com o resultado da abertura e as regiões duplicadas são obtidas.

4.2. Ambiente de desenvolvimento

A aplicação é desenvolvida na linguagem C++, com a utilização do sistema operacional Ubuntu 14.04. O desenvolvimento primordial assegura os resultados previstos, porém a execução é feita de maneira sequencial. Com a expectativa de melhora no desempenho, são utilizadas as técnicas OpenMP e MPI para execução paralela em duas implementações incrementais da aplicação.

5. Análise dos resultados

6. Considerações finais