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

Resumo. *O procedimento de copy-move forgery é o mais utilizado para adulteração de imagens digitais, onde uma região da imagem é replicada de modo a esconder ou multiplicar objetos. Uma interessante técnica para detecção de manipulações em imagens digitais é a utilização das características espaciais da imagem para identificar similaridades entre regiões distintas e, por meio da análise dos vetores de deslocamento entre as regiões identificadas, é possível constatar a autenticidade ou adulteração do documento. São utilizadas duas abordagens para a paralelização do algoritmo. A primeira delas utiliza memória compartilhada (OpenMP) e a segunda utiliza processamento distribuído (MPI). Com o paralelismo de processamento, foi possível aumentar o desempenho da detecção da adulteração em até 73%.*

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 2 apresenta as duas técnicas utilizadas para a obtenção do paralelismo na aplicação desenvolvida. A Seção 3 versa sobre o tratamento de imagens digitais. Na Seção 4 são descritos os passos para a implementação da aplicação proposta. As simulações e seus resultados são descritos na Seção 5 e, por fim, a Seção 6 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 1.

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

A eficiência de um programa paralelizado é medida pela razão entre o *speed-up* e a quantidade de núcleos do processador, conforme equação 2.

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

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 1.

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 [**?**].

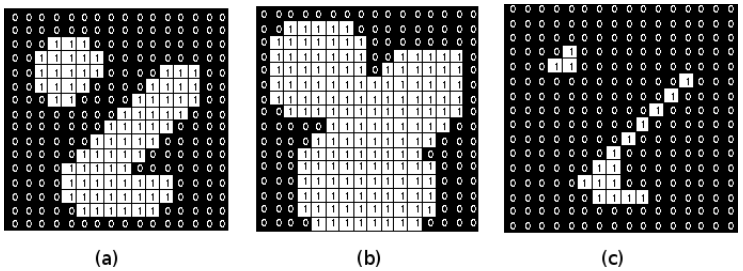

Figure 1. (a) Imagem original; (b) Imagem dilatada; (c) Imagem erodida.

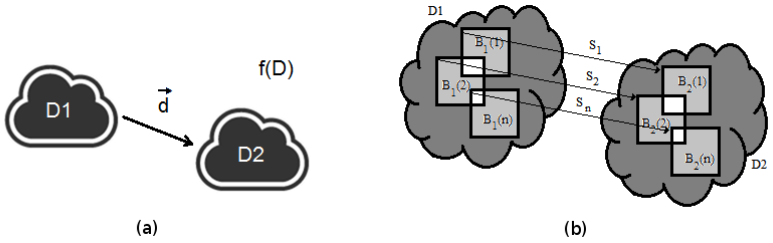

Figure 2. (a) Duplicação de regiões na imagem D; (b) Vetor de deslocamento entre blocos de regiões similares

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 D_2, onde $D_2 = D_1 + d$, conforme Figura 2-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.

A imagem de entrada é subdividida em blocos de dimensões bxb. Ao total, a imagem, de dimensões MxN, será descrita por $S = (M - b + 1)(N - b + 1)$ 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 2-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

Para as simulações foi utilizado um computador com processador Intel Core i7-4500U CPU @ 1.80GHz (com dois núcleos e quatro *threads*) e 8GB de memória RAM. A máquina possui instalado o sistema operacional Ubuntu 14.04 64 bits.

A aplicação é desenvolvida na linguagem C++. 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

Como entrada, foram utilizadas cinco imagens de diferentes tamanhos, conforme apresentado na Tabela 1 com blocos de informação para o *blocksize* igual a 16x16. O algoritmo de *shift vector* divide cada imagem em blocos, calcula o vetor de características, ordena os blocos, compara todos os pares de blocos consecutivos, verifica a similaridade por meio de um histograma e descarta os falso-positivos. Observe que, no caso da imagem *img-05*, são gerados 4.971.033 blocos de informação.

Table 1. Tamanho e dimensões dos arquivos de entrada

Label	Tamanho (MB)	Largura (px)	Altura (px)
img-01	1,1	1250	300
img-02	2,2	1250	575
img-03	4,3	1560	925
img-04	8,5	2070	1368
img-05	15,1	2592	1944

As imagens foram utilizadas como entrada para a aplicação, que detectou corretamente as adulterações em todos os casos. O desempenho da tarefa de descoberta da manipulação na aplicação serializada pode ser visto na Tabela 2. Estes valores servirão de parâmetro para a avaliação dos cenários paralelizados.

O algoritmo de criação do vetor de características possui complexidade igual a $O(n^2)$. Esta é a tarefa mais cara, chegando a consumir 90% do esforço de processamento da aplicação. As tarefas de ordenação de blocos, criação de histograma e eliminação de falso-positivos possuem complexidade linear e apresentam dependência de dados entre iterações sucessivas.

Table 2. Desempenho da execução serializada da aplicação

Imagem	Tempo (μs)
img-01	4582935
img-02	8133682
img-03	15820772
img-04	37126480
img-05	58514158

A Figura 3 apresenta um exemplo de detecção de uma adulteração do tipo *copy-move forgery* em uma imagem.

Figure 3. (a) Imagem adulterada; (b) Detecção da adulteração.

5.1. Paralelismo em OpenMP

Durante o processo de implementação e testes, foi constatado que o escalonamento dinâmico foi o mais eficiente na quase totalidade das aferições. Por este motivo, somente serão apresentados os resultados para este escalonador.

Na implementação OpenMP, cada processo recebeu a responsabilidade de calcular as características de um bloco por vez. As cinco imagens foram utilizadas como entrada da aplicação e o número de *threads* utilizadas na detecção foi incrementado em duas unidades a cada iteração. Os resultados obtidos podem ser visualizados na Tabela 3.

5.2. Paralelismo em MPI

Na implementação MPI, uma seção linear da imagem é transmitida a cada processo em cada iteração. Deste modo, ao invés de efetuar a extração das características por blocos alternados, os processos MPI dividem as seções sob sua responsabilidade em janelas de igual tamanho no qual os blocos são configurados (no caso, 16x16 pixels).

Semelhantemente, as cinco imagens foram utilizadas como entrada da aplicação e o número de *threads* utilizadas na detecção foi incrementado em duas unidades a cada iteração. Os resultados obtidos podem ser visualizados na Tabela 3.

6. Considerações finais

Analisando o *speed-up* obtido com OpenMP e MPI, é possível observar que o tempo de execução em paralelo possui ganhos significativos com relação ao tempo de execução

Table 3. Desempenho da execução com paralelismo em OpenMP

Imagem	2	4	6	8	10	Threads
img-01	3127274	2762568	2809358	2790926	2873154	
img-02	5344887	4694369	4732559	4782824	4743334	
img-03	10231756	9282441	9305634	9290024	9363854	Tempo (μs)
img-04	26321177	24317259	24641359	24603312	24527624	
img-05	39826680	36273938	37118727	37220529	36385214	
img-01	1,45	1,65	1,62	1,63	1,58	
img-02	1,52	1,73	1,72	1,70	1,71	
img-03	1,54	1,70	1,69	1,70	1,68	*speed-up*
img-04	1,40	1,51	1,49	1,50	1,50	
img-05	1,47	1,62	1,58	1,58	1,61	
img-01	72,99%	41,31%	27,08%	20,44%	15,88%	
img-02	76,23%	43,39%	28,69%	21,29%	17,18%	
img-03	77,20%	42,55%	28,29%	21,25%	16,87%	Eficiência
img-04	70,12%	37,95%	24,96%	18,75%	15,05%	
img-05	73,96%	40,60%	26,45%	19,78%	16,19%	

Table 4. Desempenho da execução com paralelismo em MPI

Imagem	2	4	6	8	10	Threads
img-01	4186326	4030228	3250678	3951783	4066305	
img-02	5471995	4986739	5625084	5173484	5582370	
img-03	11821220	9650020	10726416	10325904	11040981	Tempo (μs)
img-04	25856842	25744934	27673709	26095908	28700199	
img-05	42036315	38646075	42103407	39740096	43673177	
img-01	1,10	1,14	1,42	1,17	1,13	
img-02	1,50	1,65	1,46	1,59	1,47	
img-03	1,36	1,67	1,50	1,56	1,46	*speed-up*
img-04	1,46	1,47	1,37	1,45	1,32	
img-05	1,42	1,54	1,42	1,50	1,37	
img-01	55,28%	28,71%	23,73%	14,64%	11,38%	
img-02	75,38%	41,35%	24,44%	19,93%	14,77%	
img-03	68,49%	41,95%	25,16%	19,60%	14,66%	Eficiência
img-04	73,42%	36,87%	22,86%	18,18%	13,23%	
img-05	71,16%	38,70%	23,68%	18,82%	13,70%	

sequencial. A execução paralela atinge o *speed-up* ótimo de 1,73 na implementação OpenMP e 1,67 na implementação MPI, ambas quando são utilizadas 4 *threads*/processos. Isto ocorre devido ao fato de haver quatro *cores* existente na máquina de testes quando é explorada a capacidade de *hyperthreading* do processador.

O algoritmo paralelizado com OpenMP se mostrou mais eficiente, devido ao fato de não haver a necessidades de tráfego de dados via rede, como é necessário ao MPI. Procedimentos que possuem interdependência de dados e complexidade inferior à distribuição/centralização dos dados com MPI se mostraram ineficientes, ou seja, o custo

de tráfego dos dados é maior do que o custo do processamento dos dados em si.

Tanto no OpenMP quanto no MPI foi possível perceber que a eficiência manteve níveis constantes à medida em que o número de *threads* e o tamanho da entrada aumentavam. Por esse motivo, conclui-se que a paralelização do algoritmo de detecção de *copy-move forgery* possui escalabilidade fraca.