Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programa de Pós-Graduação em Informática

Programação Paralela

Paralelismo em algoritmos de detecção de adulterações em imagens digitais

Vitor Filincowsky Ribeiro

Jefferson Chaves Gomes

Felipe Lopes de Souza

07 de Julho de 2015

Agenda

Introdução

- Popularização das fotografias digitais
- Desenvolvimento de aplicativos de edição de imagens
- Utilização maliciosa

Introdução

Introdução

- Mecanismos anti-adulteração
- Detecção de adulterações
- Complexidade de hardware
- Alto custo computacional

Justificativa

Identificação de adulterações

- Análise espacial
- Extração de vetores, comparação de blocos de bytes, transformações matemáticas, operações matriciais...
- Importante esforço computacional

Justificativa

Motivação

- Determinação de níveis aceitáveis de performance
- Apresentação de resposta em tempo real

Justificativa

Motivação

- Determinação de níveis aceitáveis de performance
- Apresentação de resposta em tempo real

Proposta

- Paralelismo na execução
- Processamento distribuído

Objetivos

Avaliar a performance resultante da paralelização e distribuição de um algoritmo de identificação de adulterações em imagens digitais

Objetivos

Avaliar a performance resultante da paralelização e distribuição de um algoritmo de identificação de adulterações em imagens digitais

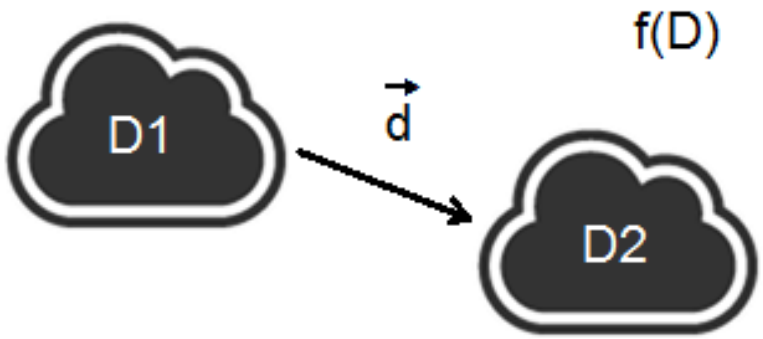
- Detecção de fraudes do tipo *copy-move forgery* em uma imagem digital com a utilização de um algoritmo de *shift vector*
- Técnicas de paralelismo e processamento distribuído
- Avaliação de desempenho

Copy-move forgery

a b

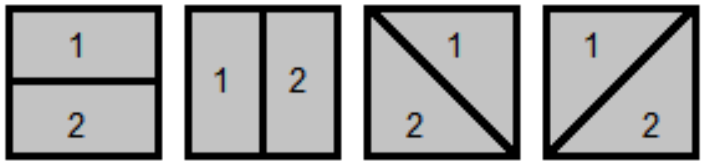
Copy-move forgery

Haverá ao menos duas regiões similares em uma imagem adulterada



Extração dos vetores característicos

- Imagem subdividida em blocos bxb
  - Imagem de dimensões MxN
  - $S = (M - b + 1)(N - b + 1)$ blocos
- Cálculo de vetor de características
  - Informações sobre cores, níveis de cinza e deslocamento dos blocos
  - Média dos valores RGB
  - Conversão para escala de cinza
- Divisão em quatro direções de luminância

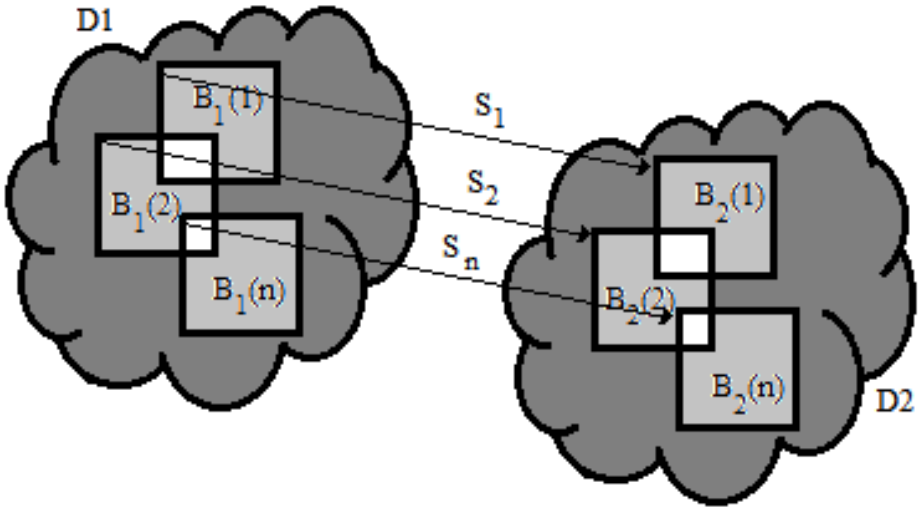


Busca por blocos similares

- Ordenação lexicográfica dos blocos
  - Matriz de dimensões Sxb^2
  - Comparação de blocos consecutivos V_i e V_j
  - Cálculo de similaridade

Eliminação dos falso-positivos

- Blocos similares não implicam duplicação de região
- Regiões duplicadas: blocos possuem o mesmo *shift*

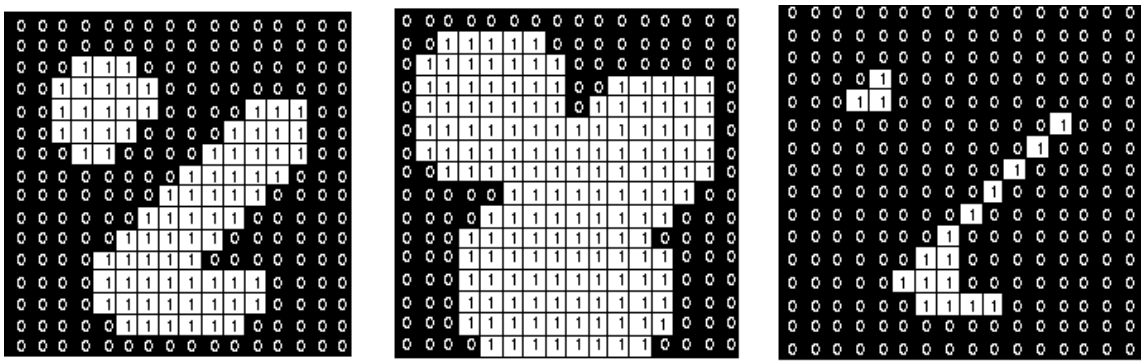


Eliminação dos falso-positivos

- Histograma criado para armazenar *shifts*
- $H(d') = H(dx', dy')$: frequência de ocorrências
- $d = max(freq(H(d')))$
- Remoção de blocos similares incorretos

Determinação de adulteração

- Imagem binária
- Operação de abertura (erosão + dilatação)
  - Eliminação de elementos espúrios
  - Se há regiões conectadas, a imagem é adulterada
- *Merge* da imagem binária com a imagem de entrada



(a) (b) (c)

Ambiente de execução e testes

- Processador Intel Core i7-4500U CPU @ 1.80GHz
  - dois núcleos e suporte *hyper-threading*
- 8GB de memória RAM
- SO Ubuntu 14.04 64 bits.
- Linguagem de programação: C++

Implementação serial

Desenvolvimento primordial assegura os resultados previstos

- Execução sequencial

Expectativa de melhora no desempenho

- Implementações incrementais da aplicação
- Execução paralela com OpenMP e MPI

Implementação serial

O algoritmo de *shift vector*

- Imagens divididas em blocos
- Cálculo do vetor de características
- Ordenação dos blocos
- Comparação de pares de blocos consecutivos
- Histograma de similaridade
- Descarte de falso-positivos

Implementação serial

O algoritmo de *shift vector*

- Imagens divididas em blocos
- Cálculo do vetor de características
- Ordenação dos blocos
- Comparação de pares de blocos consecutivos
- Histograma de similaridade
- Descarte de falso-positivos

- Geração do vetor de características: $O(n^4)$
- Ordenação de blocos: $O(nlg_2n)$
- Demais tarefas: $O(n)$

Implementação serial

- Cinco imagens de diferentes tamanhos
- Imagem *img-05*: 4.971.033 blocos de 16x16 pixels

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

Implementação serial

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

(a)

(b)

OpenMP

Comportamento paralelo em programas com acesso a áreas de memória compartilhada

OpenMP

Comportamento paralelo em programas com acesso a áreas de memória compartilhada

- Escalonamento dinâmico foi o mais eficiente na quase totalidade das aferições
- Cálculo das características de um bloco por vez
- Número de *threads* incrementado em duas unidades a cada iteração

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%	

MPI

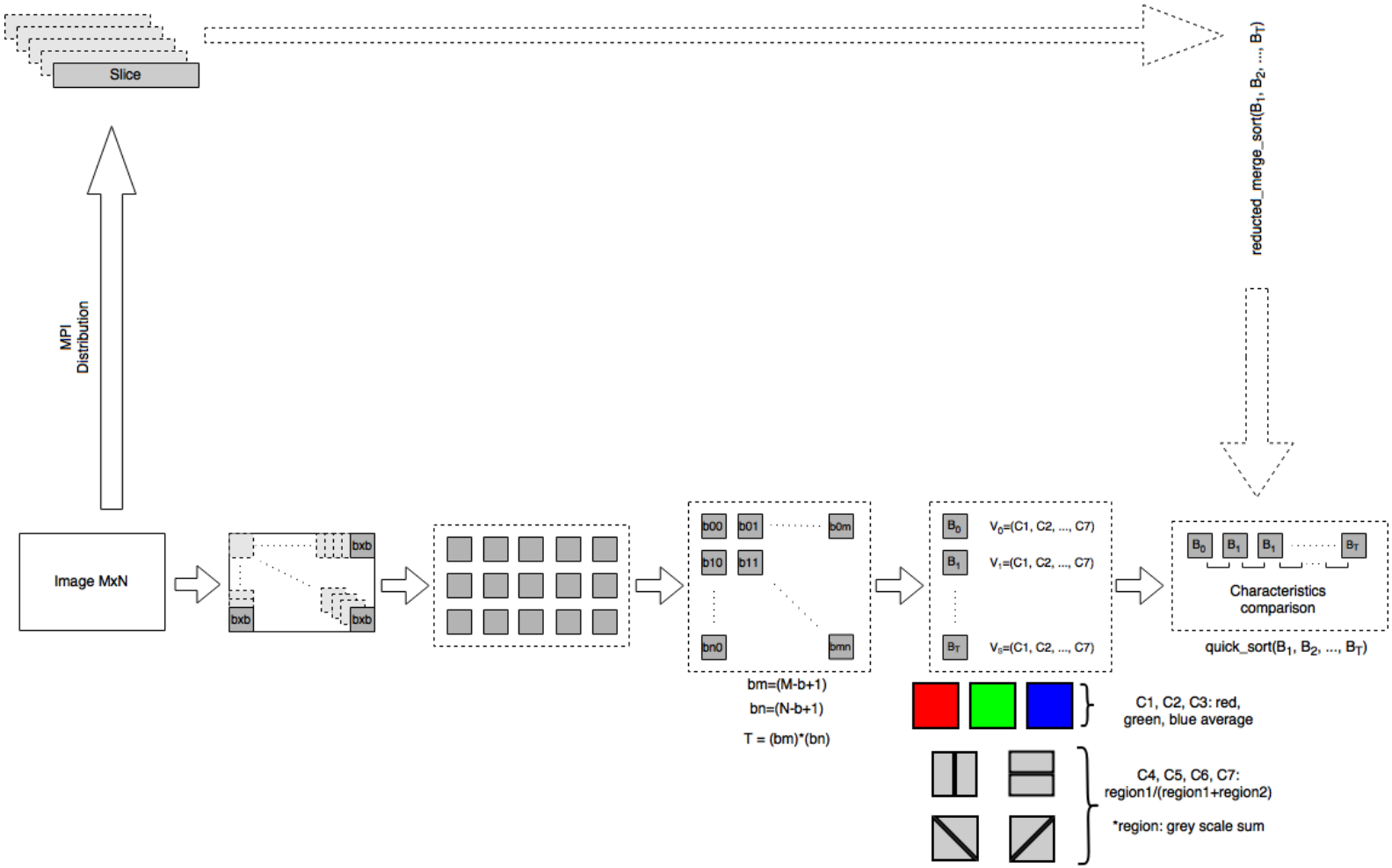
- Programas paralelos utilizando o modelo de memória distribuída
- Transmissão de mensagens entre programas paralelos que executam em ambientes multiprocessados

MPI

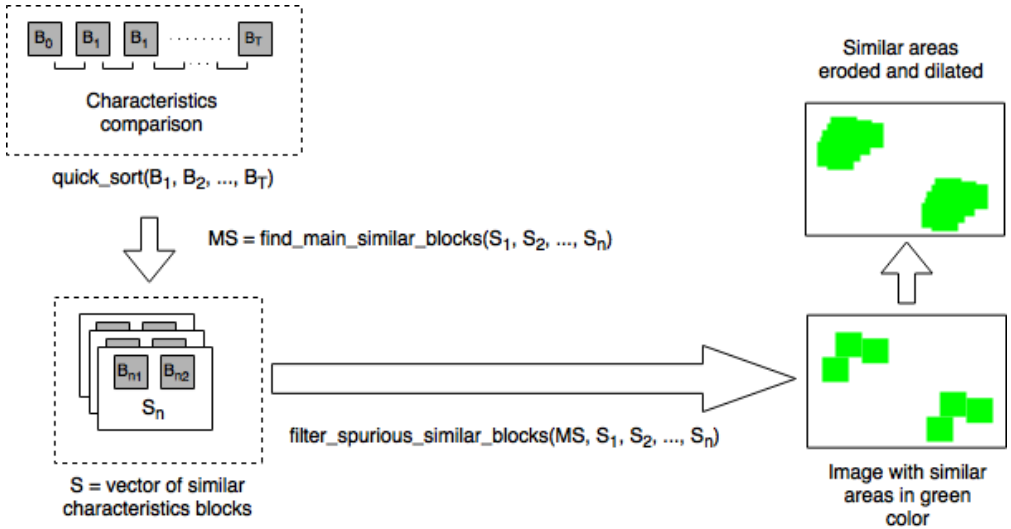
- Programas paralelos utilizando o modelo de memória distribuída
- Transmissão de mensagens entre programas paralelos que executam em ambientes multiprocessados

- Seções lineares da imagem transmitidas a cada processo
- Divisão das seções em janelas de 16x16 pixels

MPI


Slice
MPI Distribution
Image MxN
bxb
bxb
bxb
b00
b01
b0m
b10
b11
bn0
bmn
bm=(M-b+1)
bn=(N-b+1)
T = (bm)*(bn)
B0
V0=(C1, C2, ..., C7)
B1
V1=(C1, C2, ..., C7)
BT
VS=(C1, C2, ..., C7)
C1, C2, C3: red, green, blue average
C4, C5, C6, C7: region1/(region1+region2)
*region: grey scale sum
reducted_merge_sort(B1, B2, ..., BT)
B0
B1
B1
BT
Characteristics comparison
quick_sort(B1, B2, ..., BT)

MPI


B0 B1 B1 ..... BT
Characteristics comparison
quick_sort(B1, B2, ..., BT)
MS = find_main_similar_blocks(S1, S2, ..., Sn)
Bn1 Bn2
Sn
S = vector of similar characteristics blocks
filter_spurious_similar_blocks(MS, S1, S2, ..., Sn)
Similar areas eroded and dilated
Image with similar areas in green color

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%	

Conclusões

- Execução paralela: ganhos significativos com relação à execução sequencial
- *Speed-up* máximo
  - OpenMP: 1,73
  - MPI: 1,67
  - dois *cores*
  - capacidade de *hyper-threading*

Conclusões

- OpenMP se mostrou mais eficiente
  - Não há tráfego de dados via rede
- MPI é menos eficiente
  - Interdependência de dados e complexidade inferior à distribuição/centralização
  - Custo de tráfego maior do que custo de processamento

Conclusões

- OpenMP se mostrou mais eficiente
  - Não há tráfego de dados via rede
- MPI é menos eficiente
  - Interdependência de dados e complexidade inferior à distribuição/centralização
  - Custo de tráfego maior do que custo de processamento
- Eficiência diminui
  - Aumento no número de *threads*
  - Aumento do tamanho da entrada
- Detecção de *copy-move forgery* não é escalável

Obrigado!