Comparação e classificação de algoritmos de rotulação de componentes conexos em processamento de imagens

Victor Matheus de Araujo Oliveira , Roberto Alencar lotufo (Orientador)

Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101, 13083-970 – Campinas, SP, Brasil

victormatheus@gmail.com,lotufo@unicamp.br

Abstract – In this article we propose a classification and a comparison of some Connected Components Labeling algorithms, we will consider algorithm complexity, cache memory use and paralelization aspects, specially in GPU architecture. Also, we're going to use the collaborative scientific writing and programming platform Adessowiki [3] to make our benchmarks public.

Keywords – CCL, image processing, GPU, parallel algorithms

1. Introdução

O uso de algoritmos de rotulação de componentes conexos em grafos remonta ao próprio início da ciência da computação [2]. Do mesmo modo, seu uso em imagens remonta ao começo do estudo do processamento digital de imagens [4].

Sendo a rotulação, como também passaremos a chamá-la a partir de agora, uma ferramenta importante não só em processamento de imagens, mas também em engenharia e física, é natural que muitos algoritmos tenham sido propostos até o momento na literatura. Nestes artigos, muitas vezes são mostrados algoritmos, implementações e benchmarks.

Porém, a longo prazo existem mudanças graduais de linguagem de programação, na arquitetura das máquinas, entre outros, que são capazes de fazer com que algoritmos que possuíam baixa performance sejam eficientes hoje em dia. Podemos tomar como exemplo desse fenômeno a recente renovação no ramo da computação paralela, algoritmos destinados à execução em máquinas massivamente paralelas se tornam mais competitivos com o auxílio de placas gráficas.

Assim, para verificar o desempenho de um algoritmo não basta apenas possuirmos o próprio algoritmo, sua implementação e benchmarks, mas também sua *execução*.

Para isso, utilizaremos a plataforma de programação colaborativa Adessowiki [3], em que os códigos dos algoritmos na linguagem Python e em C estarão disponíveis publicamente, assim como as imagens de teste, resultados de benchmarks e a própria execução do algoritmo, que acontecerá no servidor do Adessowiki.

2. Descrição do problema

Para definir o problema da rotulação, precisamos dos conceitos de *imagem*, *vizinhança*, *conectividade*, *caminho conexo*, *componente conexo* e *partição*.

Uma *imagem binária* é um par $\hat{I} = (D_I, I)$ em que $D_I \subset \mathbb{Z}^2$ e I é um mapeamento $I(p) \in \{0,1\} \forall p \in D_I$. Também podemos estabeler dois conjuntos: A *frente*, em que $I(p) = 1$ e o *fundo*, onde $I(p) = 0$. Chamaremos a frente de $\mathbf{F}$ e o fundo de $\mathbf{B}$.

Nesse artigo, trataremos apenas de imagens dimensionais binárias, já que alguns conceitos como o de componente conexo são mais simples de serem descritos em imagens binárias e o funcionamento dos algoritmos de rotulação pode ser facilmente generalizado para dimensões maiores e para imagens multi-banda.

Vizinhança é uma relação binária $D_I \times D_I$ entre os pixels em uma imagem que depende de suas posições relativas e possivelmente de algum outro atributo da imagem. Por exemplo, a *4-Vizinhança* de um pixel p de coordenadas (x, y) é o conjunto $\boldsymbol{V}(p) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$.

A *conectividade* entre pixels se determina se além deles serem vizinhos, também atendem a algum critério com relação à seus atributos. Por exemplo, a relação *4-conexa*, em que dois pixels p e q estão 4-conectados se são 4-vizinhos e ambos estão em $\mathbf{F}$. Se dois pixels atendem a uma relação de

conectividade dizemos que são *adjacentes*.

Um *caminho conexo* entre p e q, respectivamente de coordenadas (x_0, y_0) e (x_n, y_n), é uma sequência de pixels distintos de coordenadas $(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)$ em que (x_i, y_i) é adjacente a (x_{i-1}, y_{i-1}) para $1 \leq i \leq n$.

Um subconjunto S de uma imagem binária é um *componente conexo* se dado um pixel $p \in S$, S é formado por todos os pixels que possuem um caminho conexo até p e é maximal, ou seja, todos os pixels da imagem que possuem algum caminho conexo até p então em S.

Uma *partição* de uma imagem é um conjunto de componentes conexos disjuntas cuja união é D_I.

A *rotulação de componentes conexos* é o problema de obter uma partição da imagem e atribuir um *rótulo* $l \in \mathbb{N}$ para cada componente conexo em $\mathbf{F}$.

Existem dois modos geralmente utilizados na literatura de atribuir rótulos, a atribuição *sequencial* e a de *menor endereço*.

Na atribuição sequencial atribuímos o rótulo à componente de acordo com a ordem em que o componente é processado, portanto, teremos um conjunto $l \in \{1, 2, \cdots, L\}$ de rótulos, em que L é o número total de componentes conexos na imagem.

Na atribuição de menor endereço, o rótulo do componente fica sendo o menor endereço raster dos pixels naquele componente conexo.

A estratégia que será usada ao longo deste artigo é a de menor endereço, pois ela não escalona melhor em algoritmos paralelos e é mais fácil de implementar no geral. Todos os algoritmos aqui apresentados serão adaptados para seguir essa regra. Assim garantimos que todas as rotulações terão o mesmo resultado independentemente do algoritmo utilizado.

3. Exemplo de aplicação

Usaremos a rotulação para identificar letras, palavras e parágrafos em uma imagem binária em que o texto está em $\mathbf{F}$.

Para separar letras, usamos o critério tradicional de 8-conectividade. Portanto, dado um certo pixel $p \in \mathbf{F}$ e um pixel q ao redor de p, em que

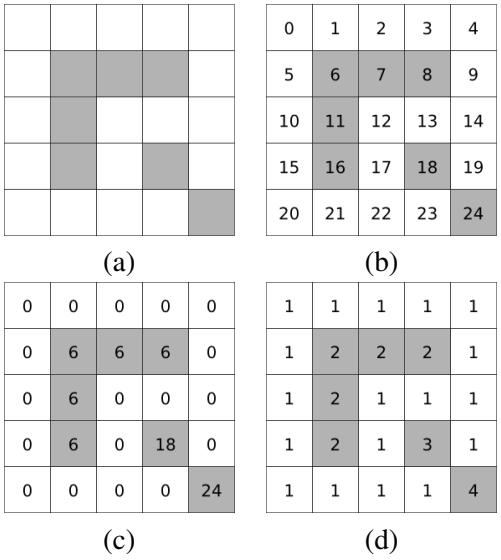

Figura 1. (a) Imagem binária; (b) Endereços na ordem raster; (c) Rotulação sequencial; (d) Rotulação usando menor endereço raster

$q \in \mathbf{F}$, então p e q são adjacentes.

Palavras são letras próximas uma da outra. Para separar as palavras, usamos uma relação de conexidade diferente. Dado um certo pixel $p \in \mathbf{F}$, fazemos um retângulo de 7 pixels de altura e 11 de altura e colocamos p no centro deste retângulo. Todos os pixels que estiverem dentro do retângulo e estão em $\mathbf{F}$ são adjacentes a p. Os valores 7 e 11 foram escolhidos experimentalmente e dependem do tamanho da fonte.

Do mesmo modo, parágrafos são palavras próximas uma da outra. Neste caso, a conectividade vai ser dada por um retângulo de 35 por 20 pixels.

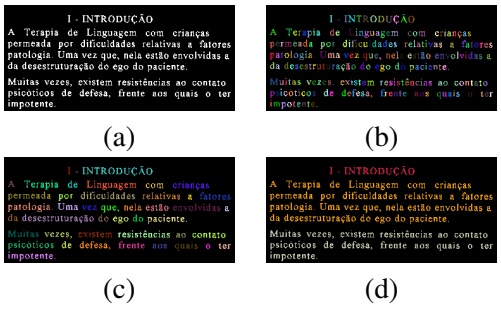

Figura 2. (a) Exemplo de imagem binária com texto; (b) Segmentação das letras; (c) Segmentação de palavras; (d) Segmentação de parágrafos

4. Busca em largura

A busca em largura é uma técnica muito utilizada para percorrer grafos [2]. Uma imagem binária $\hat{I}$ pode ser vista como um grafo G em que os pixels são os vértices e a relação de adjacência estabelece as arestas. Esse procedimento faz uma busca em largura em G para descobrir as componentes conexas.

No seguinte código Python podemos ver o funcionamento do algoritmo. Há como entrada uma imagem I e como saída, a imagem rotulada L, Q é uma fila FIFO que guarda os pixels que ainda serão rotulados.

```
def labeling_BFS(I):
  L = Array(I.N, NOLABEL)
  Q = Queue()

  for p in I.domain:
    if L[p] == NOLABEL and I[p]:
      L[p] = p
      Q.enqueue(p)
      while Q != 0:
        q = Q.dequeue()
        for n in I.adjacency(q):
          if L[n] == NOLABEL:
            L[n] = p
            Q.enqueue(n)
```

5. Conjuntos disjuntos

Esse algoritmo usa uma estrutura de conjuntos disjuntos [1]. Percorre-se a imagem fazer a união de um pixel com seus adjacentes, Quando um pixel p se une a um pixel q, buscamos os representantes dos componentes a que p e q pertencem e fazemos com que um dos representantes se ligue a outro, juntando os componentes um um só. No fim desse processo teremos os componentes conexos da imagem.

Abaixo está o código Python da implementação do algoritmo. Osamos a estratégia do encurtamento de caminho no procedimento *find*. Temos como entrada a imagem I, como saída a imagem rotulada L e como estrutura auxiliar o vetor P, que guarda a estrutura de conjuntos disjuntos, o vetor P é inicializado com a sequencia $\{0, 1, \cdots, \left\|I\right\| - 1\}$, pois no início, todo pixel é sua própria componente conexa.

```
def merge(P, x, y):
  a = self.find(P, x)
  b = self.find(P, y)
  if a < b:
    P[b] = a
  elif a > b:
    P[a] = b

def find(P, x):
  k = x
  while P[k] != k:
    k = P[k]
    equiv = k
    k = j = x
    while P[k] != k:
      j = k
      k = [k]
      P[j] = equiv
  return equiv

def labeling_UnionFind(I):
  P = Sequence(I.N)

  for i in I.domain:
    if I[i]:
      for j in I.adjacency(p):
        merge(P,i,j)
    else:
      P[i] = NOLABEL

  for i in I.domain:
    find(P,i)

  return P
```

6. Propagação

O algoritmo de propagação difere dos mencionados anteriormente porque é *iterativo*, portanto, o número de passos do algoritmo depende da entrada fornecida.

O funcionamento é simples, em uma iteração do algoritmo cada pixel verifica se seus adjacentes possuem um rótulo menor que o seu, se for o caso, ele troca seu rótulo pelo menor. Essa etapa é repetida até a convergência.

No código Python a seguir temos a implementação do algoritmo de propagação, além da etapa de verificação dos rótulos das adjacências (*scan*), é feita uma etapa em que se resolve as cadeias de rótulos (*analysis*). Essa etapa serve para acelerar a convergência do algoritmo.

```
def scan(I, L):
  changed = False
  for p in I.domain:
    tmp_label = L[p]
    for q in I.adjacency(p):
      tmp_label = \
        min(tmp_label, L[q])
    if tmp_label < L[p]:
      L[p] = tmp_label
      changed = True
  return changed

def analysis(I,L):
  for p in I.domain:
    q = p
    while q != L[q]:
      q = L[q]

    equiv = q
    r = q = p
    while L[q] != L[L[q]]:
      r = q
      q = L[q]
      L[r] = equiv

def labelling_Propagation(I):
  L = Sequence(I.N)

  changed = True
  while changed:
    changed = scan(I,L)
    analysis(I,L)

  return L
```

Por ter um funcionamento simples e acesso de memória bem organizado, o algoritmo de propapagação é um bom candidato à paralelização.

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Chapter 21: Data structures for disjoint sets. In *Introduction to Algorithms*, pages 498–524. MIT Press, 2001.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Chapter 22: Elementary graph algorithms. In *Introduction to Algorithms*, pages 552–556. MIT Press, 2001.

[3] Lotufo et. al. Adessowiki - on-line collaborative scientific programming platform. *Wikisym*, 2009.

[4] Azriel Rosenfeld. Connectivity in digital pictures. *J. ACM*, 17(1):146–160, 1970.