Aplicação do Filtro de Convolução em Imagens Gigantescas

BAYARD DA ROCHA

PROJETO DE CONCLUSÃO

CCM-206 -Inteligencia na WEB e Big Data

Abstract

Na aplicação do Filtro de Convolução em Imagens Gigantescas nos deparamos com o desafio da obtenção de boa performance na execução de algoritmos que irão processar milhares de operações. A adoção dos recursos de paralelismo das linguagens de programação funcional permite ganho significativo no tempo de processamento quando falamos de grande massa de dados. Utilizando Python e Sparks, o projeto realiza um estudo comparativo entre o tempo de processamento de algoritmos desenvolvidos para execução no modelo sequencial versus algoritmo utilizando técnicas de processamento em paralelo.

Keywords: Convolução, Imagens, Python, Sparks

1. Introdução

Convolução é um efeito de filtro de propósito geral para imagens (desfocagem, nitidez, mapa de relevo, dentre outros). A técnica consiste em aplicar uma matriz que chamamos de Kernel a uma matriz geradora da Imagem através de operação matemática composta por inteiros; funciona determinando o valor de um pixel central adicionando valores ponderados de todos os seus vizinhos (posições adjacentes) tendo como saída uma nova imagem filtrada modificada.

A formula abaixo corresponde à formula de Convolução para a obtenção do novo valor do Pixel alvo:

$$V = \left| \frac{\sum_{i=1} \left(\sum_{j=1} f_{ij} d_{ij} \right)}{F} \right|$$

Figura 1: Filtro de Convolução

Para a obtenção do Pixel alvo, realizamos a multiplicação dos pixels vizinhos pelo valor na Kernel na mesma posição ; realiza-se a somatória de todas as multiplicações e o resultado divide-se pela somatória dos valores que compões a Kernel.

Graficamente podemos representar conforme abaixo.

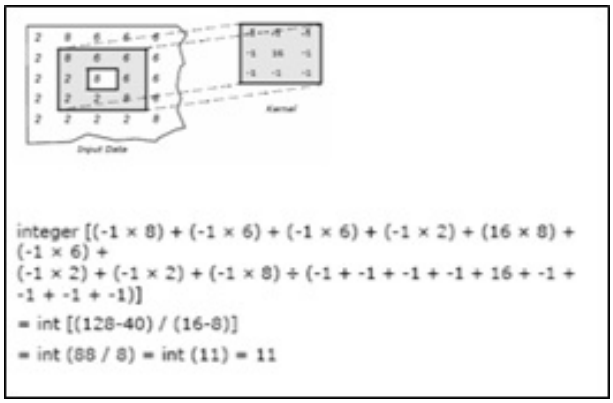

Figura 2: Kernel Convolução - Visualização Gráfica

A opção por imagens gigantes (acima de 1.000 pixels) permitirá traçarmos o comparativo entre o tempo de processamento para a aplicação do Filtro de Convolução utilizando algoritmo sequencial e o mesmo filtro utilizando técnicas de paralelismo.

2. O Projeto

Selecionei uma imagem com dimensão 2041x1080 pixels (losangeles.jpg) que resultou em 3 matrizes com dimensões 2041X1080 identificadas como rgb; desmembrar a imagem em 3 matrizes distintas foi um desafio significativo na etapa inicial do processo.

Desenvolvi o algoritmo sequencial para a criação de submatrizes 3x3 a partir da posição (0,0) de cada uma das 3 matrizes e nestas submatrizes foi aplicado o Kernel 3x3; este algoritmo sofreu diversas revisões até chegarmos em um bom nível de otimização pois entendo que só assim terei um parâmetro de comparação adequado.Obtive significativo ganho em tempo de processamento eliminando a utilização de variáveis temporárias chamadas de "variáveis de passagem".

Identifiquei após a finalização do primeiro algorítmo, que a aplicação do Kernel de forma sequencial nas três matrizes seria o candidato a ser transformado em processamento paralelo pois esta aplicação em uma matriz em nada depende do processamento nas demais matrizes.

Aplicar o paralelismo na aplicação do Kernel passa por diversas etapas de transformação da estrutura a ser utilizada.

A primeira etapa consiste da criação de um RDD no formato ((x,y), (r,g,b)) onde (x,y) é a coordenadas da matriz e (r,g,b) corresponde ao conteúdo nas matrizes r,g,b, nesta mesma coordenada. Se tivermos na posição (0,0) os valores 526 em r, 512 em g e 876 em b, teremos 0 elemento ((0,0),(526,512,876)) no RDD e assim sucessivamente até a ultima posição no nosso exemplo, (2041,1080).

Para aplicarmos o Kernel 3x3, precisamos criar submatrizes 3x3 e o caminho a ser adotado é a criação de lista com a seguinte estrutura : (x,y), (coordenada-submatriz, posição na submatriz) ,rgb). Esta estrutura irá referenciar em quais submatrizes a coordenada (x,y) do RDD estará presente e em que posição ela está. Por exemplo, para a coordenada (x,y) do RDD teremos nesta estrutura, o elemento (x,y) e ((x1,y1) (a,b)) , ((x2,y2), (a,c)) e as coordenadas (xn,yn) correspondem aos coordnada de inicio das submatrizes onde (0,0) é encontrada e (a,b),(a,c) as posições dentro de cada submatriz).

Feito isso aplicamos Map para unir o RDD e a nova estrutura para em seguida aplicar Reduce; ao final teremos uma estrutura onde poderemos aplicar o Kernel. Finalmente, construímos a nova imagem que corresponderá à aplicação do Filtro na imagem original.

3. Resultados Obtidos

A aplicação do Kernel no processamento sequencial otimizado, teve duração média de 65,5segs (com pequenas variações em cada execução).

```
for a in range (np.array(img_foto).shape [2]):
    matrizConvolu = np.copy(matrizBGR[a])
    for x in range (matrizBGR[a].shape [0]-2):
            for y in range (matrizBGR[a].shape [1]-2):
                        subMatriz = matrizBGR[a][slice(x,x+3),slice (y,y+3)]
                        matrizConvolu [x+1][y+1] = int(sum(np.reshape(subMatriz*kernel,-1)) / somaKernel)
                        if  matrizConvolu [x+1][y+1] < 0:
                                print (matrizConvolu [x+1][y+1])
                                matrizConvolu [x+1][y+1] = 0
                        qtdeOperac = qtdeOperac + 1
    matrizBGR[a] = matrizConvolu
```

Figura 3: Algoritmo Sequencial - Aplicação do Kernel

Como estratégia para facilitar a concepção do algoritmo com técnicas de paralelismo, criei as matrizes (r,g,b) de forma randômica

Consegui criar o RDD com as coordenadas e valores r,g,b e segui com a estratégia de criar uma função com as coordenadas das submatrizes para na sequência aplicar o map neste RDD. Estou ainda neste ponto que corresponde a execução do Map; não tive sucesso até o momento por desconhecimento adequado da linguagem utilizada. A intenção é dar sequência porém não mais em tempo de projeto da disciplina. Por esta razão , não elementos para apresentar o comparativo final de meu projeto.

No github disponilizei o algoritmo com processamento sequencial (Projeto Imagem - Sequencial - Foto.ipynb) , a imagem utilizada (losangeles.jpg) e também o algoritmo incompleto ainda, utilizando paralelismo (Projeto Imagem - Tabelas Randomicas - Paralelismo.ipynb).

4. Fontes

- http://graphics.stanford.edu/courses/cs178/applets/convolution.html
- www.inatel.br/docentes/ynoguti/downloads-arquivos/.../18-convolucao-s104246-1
- https://spark.apache.org/docs/latest/rdd-programming-guide.html
- https://panda.ime.usp.br/pensepy/static/pensepy/
- https://panda.ime.usp.br/aulasPython/static/aulasPython/index.html
- https://github.com/BayarddaRocha/BIGDATA2018/tree/master/Projeto