Relatório Sistemas Distribuídos

Integrantes: Caio Saud, Carolina Carvalhosa e Gabriela Gonçalves

Professor: Tiago Assumpção

Introdução

Neste trabalho, utilizamos a Sucuri, que é uma biblioteca em Python utilizada para otimizar uma aplicação com técnicas de programação paralela. A ideia do trabalho é implementar uma aplicação distribuída utilizando a Sucuri de maneira a obter ganho de desempenho conforme mais "*Workers*" forem adicionados. A aplicação proposta é um conversor de imagens em suas versões negativas.

Todo o código junto com a lib e um README.md está em um repositório público no github: https://github.com/gabriela-ogoncalves/sucuri-negative-images.

Preparação do ambiente

Foi necessário uma preparação do ambiente para podermos executar nosso programa a qual está descrita de forma mais detalhada no README.md do github:

- Instalar a **Python Imaging Library** através dos comandos;
- Instalar as dependências necessárias para rodar o projeto: **numpy** e **scikit-image**;

Executando a aplicação

Para executar a aplicação deve-se colocar as imagens que se pretende converter na pasta *"**imgs/**"* e, ao executar, passar como parâmetro a quantidade de "**Workers**", exemplo:

$ python3 main.py 1

Após isso, só esperar até que as imagens sejam convertidas e salvas na pasta: *"**/negative_imgs**"*.

Descrição do código implementado

Agora iremos apresentar o código implementado:

```
import os
import sys
import time
import numpy as np
import skimage
from skimage import io
from PIL import Image
import pathlib
from pyDF import *

save_dir = 'negative_imgs'
imgs_folder = str(os.path.dirname(os.path.realpath(__file__))) + "/imgs"

def list_imgs(rootdir):
        """ Lista os nomes dos arquivos de imagens """
        file_names = []
        for current, directories, files in os.walk(rootdir):
        for f in files:
             file_names.append(current + '/' + f)

        file_names.sort()
        return file_names

def get_negative(args):
        """ Converte a imagem no seu negativo """
        fname = args[0]
        splitname = fname.split('/')[-1]

        img = io.imread(fname)

        red = img[:,:,0]
        green = img[:,:,1]
        blue = img[:,:,2]

        img_ch0_R_neg = 255-red
        img_ch1_G_neg = 255-green
        img_ch2_B_neg = 255-blue

        new_image = np.stack((img_ch0_R_neg, img_ch1_G_neg, img_ch2_B_neg), axis=-1)

        splitname = save_dir + '/' + 'negative_' + splitname.split('.')[0]  + '.png'
    Image.fromarray(new_image).convert("RGB").save(splitname)
```

```
        return splitname


def print_name(args):
        """ Formata e imprime os nomes """
        fname = args[0]

        print("Convertido %s" %fname)

def sucuri(n_procs):
        """ Aplica comandos da lib Sucuri """
        nprocs = n_procs
        image_path = list_imgs(imgs_folder)

        graph = DFGraph()
        sched = Scheduler(graph, nprocs, mpi_enabled = False)

        feed_files = Source(image_path)

        convert_file = FilterTagged(get_negative, 1)

        pname = Serializer(print_name, 1)

    graph.add(feed_files)
    graph.add(convert_file)
        graph.add(pname)

    feed_files.add_edge(convert_file, 0)
    convert_file.add_edge(pname, 0)

        t0 = time.time()
        sched.start()
        t1 = time.time()
        print( "Tempo de execucao: %.3f" %(t1-t0))


sucuri(int(sys.argv[1]))
```

Imagem 1: Código implementado

Objetivo

O objetivo do trabalho foi utilizar a biblioteca sucuri que é uma biblioteca em Python para otimizar uma aplicação com técnicas de programação paralela. Com isso, unindo conhecimentos obtidos através das disciplinas de Sistemas Distribuídos, e de Computação Gráfica, criamos uma aplicação que utiliza programação paralela para obter negativos de imagens.

Testes

Instância

Foram incluídas 4 (quatro) imagens, para fins de teste, na pasta ***"imgs"***. Criamos uma pasta ***"negative_imgs"***, que inicia vazia, e é populada pelos negativos das imagens originais após rodar o programa.

Medidas

Realizamos 5 execuções com 1, 3 e 5 *workers* e, em seguida, capturamos a média de seus tempos. A medida do tempo foi gerada através do *"time"* do python.

Resultados

	Execução	Tempo
1 worker	1	2.008
	2	1.898
	3	1.856
	4	1.845
	5	1.855
	Média	1.8924

Tabela 1: Média de 5 execuções, com tempo em segundos, com 1 *worker*.

3 workers	Execução	Tempo
	1	1.608
	2	1.626
	3	1.613
	4	1.634
	5	1.673
	Média	1.6308

Tabela 2: Média de 5 execuções, com tempo em segundos, com 3 *workers*.

5 workers	Execução	Tempo
	1	0.774
	2	0.882
	3	0.807
	4	0.824
	5	0.806
	Média	0.8186

Tabela 3: Média de 5 execuções, com tempo em segundos, com 5 *workers*.

Análise dos Resultados

Conforme visto nas tabelas apresentadas na seção de resultados, podemos concluir que, à medida que aumentamos o número de *workers*, menor fica o tempo de execução. Há um ganho significativo se compararmos os resultados obtidos com apenas 1 *worker* e os com 5 *workers*. Dessa forma, podemos garantir que, na nossa aplicação, obtemos ganho de desempenho conforme mais *workers* são adicionados.

Agradecimentos

Agradecemos ao professor Tiago por nos ensinar a metodologia de sistemas distribuídos, assim como o conceito da biblioteca Sucuri. Além disso, também gostaríamos de agradecer ao professor Gilson Costa, de Computação Gráfica, pois utilizamos o conceito de obter negativo de imagem, conforme foi explicado por ele.