APLICAÇÃO DA PROGRAMAÇÃO PARALELA EM UM ALGORITMO DE CLASSIFICAÇÃO DE IMAGENS

Carlos Willian Silva Camargos, João Vitor Veloso Rodrigues

Instituto Federal de Minas Gerais (IFMG) – Campus Bambuí

cwscamargos@gmail.com, joaovitor_new2@live.com

RESUMO

Devido às demandas de processamento cada vez maiores, por parte das aplicações, formas eficazes de melhorar o desempenho dos computadores têm sido procuradas há anos. Existem duas opções principais para resolver problemas de desempenho, a primeira é aumentar o desempenho do processador, e a segunda é segmentar as tarefas em diversas partes do processador, para que possam ser executadas em conjunto. A programação paralela permite expressar o paralelismo e incluir mecanismos de sincronização e comunicação em computadores paralelos. No presente trabalho, os conceitos de *threads* e processos serão utilizados para demonstrar a aplicação da programação paralela, e também serão apresentados os resultados obtidos através de sua aplicação a algoritmos de extração de cores em imagens.

Palavras-chave: Programação paralela; Threads; Processos; Extração de características.

1 INTRODUÇÃO

Devido à demanda crescente por processamento, já há alguns anos tem se buscado maneiras eficientes de aumentar o desempenho dos computadores. Existem duas principais alternativas para solucionar o problema do desempenho, a primeira é aumentar o desempenho dos processadores e a segunda é usar vários processos ou *threads* ao mesmo tempo e dividir entre eles as tarefas do programa a ser executado. Nesse contexto, a programação paralela permite expressar o paralelismo e incluir mecanismos de sincronização e comunicação em computadores paralelos.

No presente trabalho será demonstrada uma aplicação da programação paralela utilizando os conceitos de *threads* e processos. De forma resumida, um processo é um programa que foi carregado para a memória do computador e está sendo gerenciado pelo sistema operacional. Um processo pode criar vários fluxos de execução diferentes, esses fluxos compartilham o mesmo espaço de endereçamento e são chamados de *threads*. As *threads* são gerenciadas pelo processo que as criou, enquanto que, um processo é gerenciado pelo sistema operacional e pode criar outros processos independentes que também serão gerenciados pelo sistema operacional.

O objetivo deste trabalho é demonstrar como a programação paralela pode ser usada para reduzir o tempo de processamento de diversos tipos de programas, bem como apresentar os resultados obtidos através de sua aplicação em um algoritmo de extração de características e cores de imagens.

2 METODOLOGIA OU MATERIAL E MÉTODO

Para implementar o algoritmo apresentado neste trabalho foi utilizada a linguagem Python em conjunto com algumas bibliotecas. Dentre as bibliotecas utilizadas se destacam as bibliotecas *queue*, *threading* e *multiprocessing* que foram utilizadas para implementar o paralelismo no código. A Figura 01 mostra a implementação do algoritmo de extração de cores de imagens sem paralelismo.

Figura 01 - Algoritmo sem paralelismo

```
import argparse
import time
import coffee_analyzer
import pandas as pd


def main():
    itens = []

    parser = argparse.ArgumentParser('PROCESS')
    parser.add_argument('-i', '--images', type=int, default=24)
    args = parser.parse_args()

    START_TIME = time.time()

    df = pd.read_csv('./photos.csv', delimiter=';')

    if(args.images > df.shape[0]):
        args.images = df.shape[0]

    for i in range(args.images):
        itens.append(coffee_analyzer.read_crop_analyze(df.iloc[i]))

    END_TIME = time.time()

    print('Time for sequential:', END_TIME - START_TIME, 'secs')
    print('Total itens:', len(itens))


if __name__ == '__main__':
    main()
```

Fonte: Elaborado pelos autores, 2022.

Inicialmente é criada uma lista de itens, em seguida são definidos os argumentos que serão passados como parâmetros na hora de executar o código pelo terminal, o parâmetro a ser enviado define a quantidade de imagens que serão processadas. Os dados das imagens são então extraídos do arquivo '*.csv*' e armazenados em uma tabela. No método `coffee_analyzer` essa tabela é utilizada; as imagens são cortadas e em seguida são criados dois dicionários com dados da imagem,

um deles contém dados sobre a cor da imagem e o outro contém dados sobre a iluminação. Ao final dessas etapas as imagens preparadas são adicionadas à lista de itens e o tempo de execução é exibido na tela.

O paralelismo neste código foi implementado através de *threads* e de processos, cada uma das implementações foi separada em arquivos diferentes. As figuras 02 e 03 mostram o código implementado com *threads* e processos respectivamente.

Figura 02 - Implementação do código com *threads*

```
import argparse
import time
from queue import Queue
from threading import Thread
import coffee_analyzer
import pandas as pd
import time

def function(QUEUE_IMAGES, DATA):
    while True:
        image = QUEUE_IMAGES.get()
        DATA.append(coffee_analyzer.read_crop_analyze(image))
        QUEUE_IMAGES.task_done()
        if QUEUE_IMAGES.empty():
            break


def main():
    QUEUE_IMAGES = Queue()
    DATA = []

    parser = argparse.ArgumentParser('THREADED')
    parser.add_argument('-k', '--threads', type=int, default=4)
    parser.add_argument('-i', '--images', type=int, default=24)
    args = parser.parse_args()

    TOTAL_THREADS = args.threads

    START_TIME = time.time()
    threads = [Thread(target=function, args=[QUEUE_IMAGES, DATA]) for _ in range(TOTAL_THREADS)]

    df = pd.read_csv('./photos.csv', delimiter=';')

    if(args.images > df.shape[0]):
        args.images = df.shape[0]

    for i in range(args.images):
        QUEUE_IMAGES.put(df.iloc[i])

    for thread in threads:
        thread.start()

    QUEUE_IMAGES.join()
    END_TIME = time.time()

    print('Time for Threaded:', END_TIME - START_TIME, 'secs')
    print(len(DATA))

if __name__ == "__main__":
    main()
```

Fonte: Elaborado pelos autores, 2022.

A diferença nessa implementação está no uso das *threads* através do método `Thread()`, que cria um fluxo de execução diferente para cada imagem inserida na fila `QUEUE_IMAGES`. Ao chamar o método `start()` do objeto `thread`, cada imagem é enviada a uma *thread* e então é tratada pelo algoritmo independentemente, fazendo com que todas elas sejam processadas paralelamente. A Figura 03 mostra o mesmo algoritmo, porém implementado com processos em vez de *threads*. A lógica da implementação nesse caso é a mesma das *threads* (Figura 02), com a diferença de que os fluxos de execução criados pelo processo, também são processos gerenciados pelo sistema operacional e não *threads* gerenciadas pelo processo que as criou.

Figura 03 - Implementação do código com processos

```
import argparse
import time
from multiprocessing import Process, JoinableQueue
import coffee_analyzer
import pandas as pd
import time


def function(QUEUE_IMAGES, QUEUE_DATA):
    while True:
        image = QUEUE_IMAGES.get()
        QUEUE_DATA.put(coffee_analyzer.read_crop_analyze(image))
        print(QUEUE_DATA.qsize())
        QUEUE_IMAGES.task_done()
        if QUEUE_IMAGES.empty():
            break


def main():
    QUEUE_IMAGES = JoinableQueue()
    QUEUE_DATA = JoinableQueue()

    parser = argparse.ArgumentParser('PROCESS')
    parser.add_argument('-k', '--process', type=int, default=4)
    parser.add_argument('-i', '--images', type=int, default=24)
    args = parser.parse_args()

    TOTAL_PROCESS = args.process

    START_TIME = time.time()
    list_process = [Process(target=function, args=[QUEUE_IMAGES, QUEUE_DATA]) for _ in range(TOTAL_PROCESS)]

    df = pd.read_csv('./photos.csv', delimiter=';')

    if(args.images > df.shape[0]):
        args.images = df.shape[0]

    for i in range(args.images):
        QUEUE_IMAGES.put(df.iloc[i])

    for process in list_process:
        process.start()

    QUEUE_IMAGES.join()
    END_TIME = time.time()

    print('Time for Processed:', END_TIME - START_TIME, 'secs')


if __name__ == "__main__":
    main()
```

Fonte: Elaborado pelos autores, 2022.

3 RESULTADOS E DISCUSSÃO

Para comparar os resultados das abordagens, o número de imagens foi alterado entre 25, 80, 120, 200, 500 e 700. Os resultados obtidos em segundos, podem ser observados na Tabela 01 e Figura 04. Número de *threads* e processos foi fixado como quatro nos testes, porém, podem ser alterados através do parâmetro *"-k"*. O computador utilizado nos testes possui um processador Intel® Core™ i5-7300HQ @ 2.50GHz, com quatro núcleos e quatro *threads* e 16 GB de memória RAM, 2400MHz, DDR4.

Tabela 01 - Tempo de execução em cada implementação

Quantidade de imagens	Implementação sequencial (s)	Implementação com *threads* (s)	Implementação com processos (s)
25	20,89	6,7	7,93
80	63,53	17,29	23,11
120	87,03	24,54	37,44
200	131,93	38,27	40,42
500	286,77	89,76	98,93
700	351,49	109,41	112,81

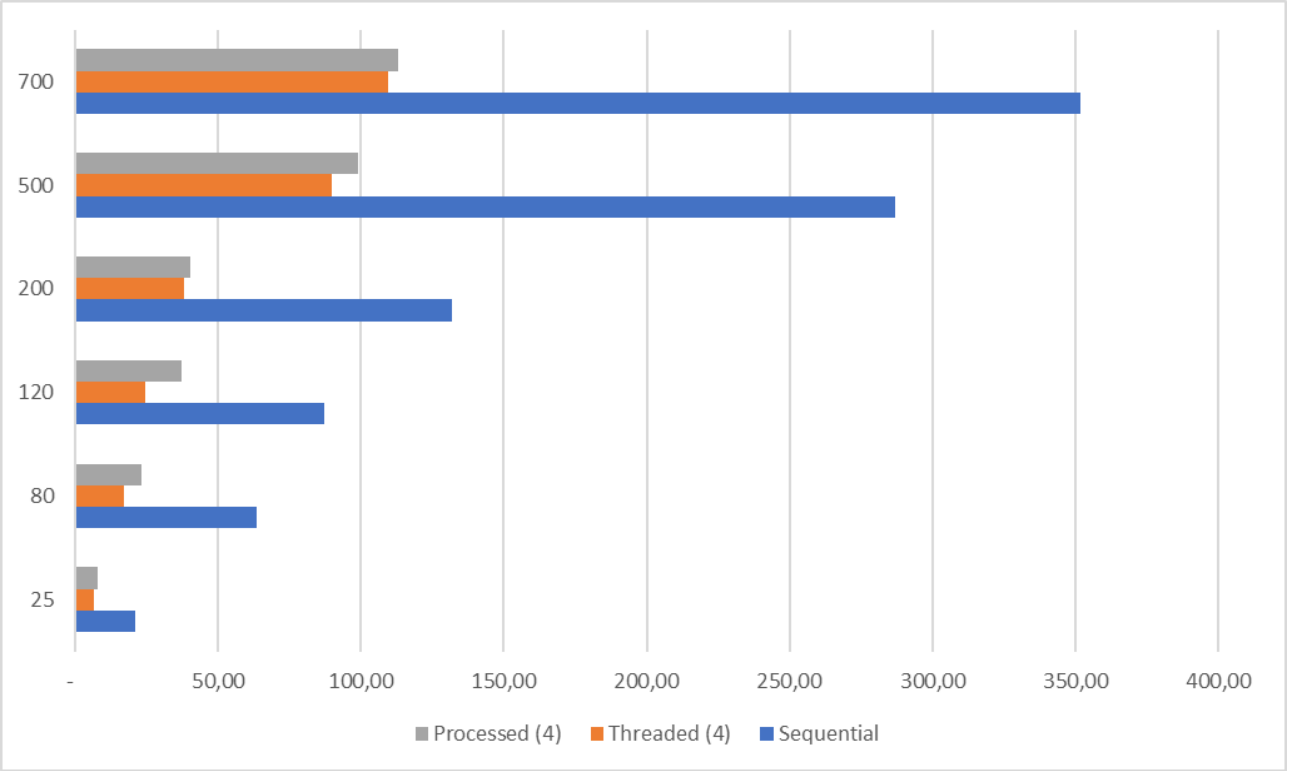
Fonte: Elaborado pelos autores, 2022.

Pode-se observar que claramente que os resultados utilizando as abordagens de paralelismo foram muito superiores à abordagem sequencial tradicional. Em grande parte dos resultados o tempo de execução para *threads* e processos foi em média, três vezes mais rápido que a sequencial, demonstrando claramente como algumas tarefas podem ser otimizadas.

Processadores com mais *threads*, podem se beneficiar ainda mais da implementação utilizando a biblioteca *threading*, pois mais imagens poderão ser processadas em paralelo, para isso deve-se alterar o parâmetro *"-k"*. Como os processos e seus recursos são gerenciados pelo sistema, e não pelo processo pai, a implementação em *threads* obteve um resultado ligeiramente melhor, mas atinge um limite físico do processador.

Outros testes foram realizados, como a alteração dos parâmetros que definem a quantidade de *threads* a serem utilizadas e processos a serem criados, porém, os resultados não foram superiores ao padrão definido de quatro processos e *threads*.

Figura 04 - Comparação das abordagens em relação à quantidade de imagens e tempo de execução.



Fonte: Elaborado pelos autores, 2022.

4 CONCLUSÃO

Comparando as 3 implementações, pode se constatar através dos resultados apresentados que o algoritmo implementado com *threads* obteve um desempenho superior à implementação com processos, devido à duas grandes diferenças: as *threads* são gerenciadas pelo processo que as criou (e não pelo sistema operacional), e compartilham o mesmo espaço de endereçamento, podendo existir, assim, várias *threads* que acessam as mesmas variáveis com facilidade. Mas ainda, a implementação como processos foi bem superior à sequencial, ou seja, qualquer uma das abordagens deverão ter resultados superiores, tendo uma leve vantagem para as *threads*.

REFERÊNCIAS BIBLIOGRÁFICAS

SATO, Liria Matsumoto; MIDORIKAWA, Edson Toshimi; SENGER, Hermes. **Introdução a programação paralela e distribuída**. Anais do XV Jornada de Atualização em Informática, Recife, PE, p. 1-56, 1996.