Avaliação de detecção de barcos com multiprocessamento em imagens grandes de alta resolução

Guilherme Braga[1], Kelvin Lima[1]

[1]Curso de Sistemas de informação – Centro Universitário Euro-Americano (UniEuro)

AV. Castanheira – CEP 71.910-900 - Brasília – DF – Brasil

guilhermebragariosdacosta@gmail.com, prokelvin65@gmail.com

Resumo. *Este artigo apresenta uma abordagem para detecção de barcos em imagens grandes utilizando técnicas de processamento paralelo. Usando as bibliotecas OpenCV para detecção de objetos, PyVips para manipulação de imagens de grande porte, e a biblioteca concurrent.futures para paralelização, o estudo demonstra a eficiência e a escalabilidade do método proposto. A imagem de entrada é dividida em partes menores, processadas simultaneamente, esperando que resultasse em uma significativa redução no tempo de processamento. Porém, ao terminar de realizar os testes, notamos que o desempenho do processamento tem uma piora significa conforme o aumento do número de threads.*

1. Introdução

A utilização de computação para resolver problemas em diversas áreas, como a marítima, é um dos grandes desafios de pesquisa tanto no país quanto no mundo. Notavelmente, o desenvolvimento dessa área vem contando com avanços tecnológicos provenientes da aplicação de diversas técnicas computacionais. Além disso, com os recentes avanços em diversos tipos de sensores, como satélites de alta resolução, tem aumentado rapidamente a quantidade de dados disponíveis para análise. Como consequência, a demanda computacional na área é cada vez maior, levando à necessidade de analisar a adequação das diversas alternativas computacionais disponíveis para essa classe de aplicações.

Assim, neste trabalho, avaliamos o uso de multiprocessamento em imagens grandes (cerca de 4GB) para a detecção de navios. Esta avaliação inclui a comparação entre diferentes abordagens de paralelização, levando em conta aspectos como tempo de execução e eficiência computacional. A utilização de processamento paralelo tem atraído bastante atenção devido à necessidade de processar grandes volumes de dados de forma eficiente e rápida. Prover recursos computacionais poderosos com baixo custo é um desafio, e o caminho para tal feito é complexo. A eficiência de processamento e a capacidade de escalar a aplicação são questões centrais que precisam ser avaliadas.

Nossas avaliações utilizam métricas como o tempo de execução total e a quantidade de recursos computacionais utilizados. Em relação ao tempo de execução, o multiprocessamento mostra-se promissor, sendo capaz de reduzir significativamente o tempo necessário para processar imagens de grande porte. No entanto, há limitações a

serem consideradas, como a necessidade de gerenciar eficientemente os recursos de memória e a comunicação entre os processos.

Apesar deste artigo focar na detecção de navios em imagens grandes, essa aplicação pertence a uma classe maior de aplicações científicas que analisam dados em alta resolução com o objetivo de identificar e analisar objetos. Esta classe maior de aplicações inclui aquelas que processam dados de satélites para diferentes fins, como monitoramento ambiental e estudos oceânicos. Assim, acreditamos que os resultados obtidos das avaliações experimentais podem ser generalizados para outras classes de aplicações.

O restante deste artigo está organizado da seguinte forma. Na Seção 2, descrevemos a aplicação motivadora em detalhes e as plataformas avaliadas. A implementação da aplicação é descrita na Seção 3, e os resultados experimentais são apresentados na Seção 4. Finalmente, na Seção 5, concluímos o artigo.

2. Aplicação Motivadora e Plataformas Avaliadas

Esta seção apresenta a aplicação real de detecção de navios em imagens de alta resolução, assim como as plataformas nas quais avaliamos experimentalmente a aplicação motivadora.

2.1 Aplicação Motivadora

A detecção de navios em imagens de alta resolução capturadas por satélites é crucial para diversas aplicações, incluindo monitoramento marítimo, segurança e estudos ambientais. As imagens de satélite modernas permitem capturar vastas áreas do oceano com alta resolução, resultando em grandes quantidades de dados para análise. Uma única imagem pode ter dimensões de uma ordem extraordinário de pixels, tornando o processamento destas imagens um desafio significativo devido ao tamanho e complexidade dos dados.

O processamento de uma única imagem, dependendo de seu tamanho, pode levar diversas horas ou dias em um processador sequencial. Um estudo em larga escala pode envolver milhares de imagens de alta resolução, que podem ser processadas múltiplas vezes para mensurar o impacto de parâmetros de estudo. Sendo assim, esse tipo de estudo feito de forma sequencial poderia levar décadas para terminar, o que o tornaria inviável, sendo possível realizar o estudo somente de forma paralelizada.

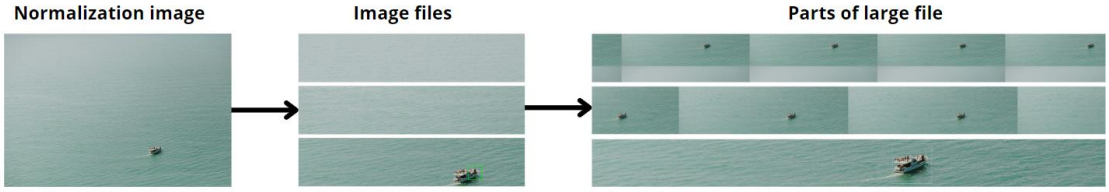

Figure 1. Fluxo de paralelização de arquivos da imagem.

As fases da aplicação são apresentadas na Figura 1. As imagens são tipicamente divididas em partes menores para processamento paralelo. Em seguida, cada parte é processada individualmente, detectando-se e delineando-se os navios presentes na imagem. As técnicas utilizadas incluem detecção de objetos utilizando algoritmos como Cascata de Haar, que são aplicadas em cada parte da imagem.

2.2 Plataforma Avaliada

Nos últimos anos, a computação científica tem evoluído significativamente com o crescimento do número de arquiteturas alternativas para execução de aplicações com alta demanda computacional. Como exemplo, podemos citar os coprocessadores como GPUs, além de CPUs multicores com um número cada vez maior de núcleos de processamento. Esses processadores conseguem aumentar significativamente a capacidade de processamento de máquinas modernas, mas as vezes, até mesmo os mais simples podem ser encontrados a preços que são elevados para grande quantidade da população. Neste trabalho, utilizamos um modelo de CPUs multicores que possui 8 núcleos e 16 processadores lógicos.

Figure 2. **Placa mãe; Lenovo 3i** ***[2021]***

A plataforma utilizada foi a apresentada na figura 2, a qual será apresentada suas configurações na tabela 1. Devemos observar com mais atenção o processador utilizado e também a quantidade de memória RAM disponível no dispositivo. Um destaque desse equipamento é que mesmo em grandes processamentos ele não chega a ter um grande aquecimento, por conta dos coolers e resfriadores, mas possui limitações por conta da quantidade de memória RAM.

Table 1. Especificacão do sistema utilizado

Wi-fi	802.11n/144.4 Wi-fi
USB	2x USB 3.0 Connector
CPU	Octo-Core AMD Ryzen 7 5700U - 1.8GHz
Memory	8GB 3200MHz
GPU	AMD Radeon(TM) Graphics

3. Implementação

A implementação foi feita utilizando Python, e sua paralelização foi feita utilizando a biblioteca concurrent.futures, com ThreadPoolExecutor. A implementação foi feita de um modelo que cada thread se responsabiliza por uma parte da imagem que foi fornecida como caminho, a imagem precisa ter no máximo 4gb de tamanho e o caminho dela precisa ser colocado no código. Cada Thread processa a parte da imagem e cria um novo arquivo. Múltiplos núcleos de processamento em uma máquina ou em um ambiente distribuído podem ser utilizados na criação de cada arquivo.

3.1 Código de multithreads

```
import cv2 as cv
import numpy as np
import os
import concurrent.futures
import pyvips
import time
from datetime import timedelta

add_dll_dir = getattr(os, "add_dll_directory", None)
vipsbin = r"C:\temp\vips-dev-w64-all-8.15.2\vips-dev-8.15\bin"
if callable(add_dll_dir):
    add_dll_dir(vipsbin)
else:
    os.environ["PATH"] = os.pathsep.join((vipsbin, os.environ["PATH"]))

imagem = pyvips.Image.new_from_file('main-project/assets/replicated_image.tiff')

largura, altura = imagem.width, imagem.height
num_partes = 16
parte_largura = largura
parte_altura = altura // num_partes

carregaAlgoritmo = cv.CascadeClassifier('main-project/haarcascades/first_cascade.xml')

def processar_parte(i):
```

```
    global num_boats  # Variável global para contar o número de barcos detectados
    # Definindo as coordenadas de recorte para esta parte
    x1 = 0
    y1 = i * parte_altura
    x2 = largura
    y2 = y1 + parte_altura

    parte_imagem = imagem.crop(x1, y1, parte_largura, parte_altura)

    parte_np = np.ndarray(buffer=parte_imagem.write_to_memory(),
                          dtype=np.uint8,
                          shape=[parte_altura, parte_largura, 3])

    parte_np = cv.cvtColor(parte_np, cv.COLOR_RGB2BGR)

    imagemCinza = cv.cvtColor(parte_np, cv.COLOR_BGR2GRAY)

    boats = carregaAlgoritmo.detectMultiScale(imagemCinza,
                                              scaleFactor=1.1,
                                              minNeighbors=3,
                                              minSize=(250, 250),
                                              maxSize=(800, 800))
    for (x, y, l, a) in boats:
        parte_imagem = parte_imagem.draw_rect(255, x, y, l, a, fill=False)
        num_boats += 1  # Incrementando o número de barcos detectados

    parte_imagem.write_to_file(f'parte_{i}.tiff')

start_time = time.time()
num_boats = 0

with concurrent.futures.ThreadPoolExecutor(max_workers=1) as executor:
    executor.map(processar_parte, range(num_partes))

print('\n\nnúmero de threads:', executor._max_workers)
print('Processamento finalizado!!\n')
```

```
end_time = time.time()

total_time = end_time - start_time

time_formatted = str(timedelta(seconds=total_time))


print(f'Número total de barcos detectados: {num_boats}')

print(f'Tempo total de processamento: {time_formatted} segundos')
```

4. Resultados

A avaliação experimental foi realizada utilizando a aplicação motivadora, implementada em Python com concurrent.futures, utilizando uma imagem de 56k X 37k pixels como entrada. A imagem possui um total de 3.81GB de dados.

4.1 Avaliação de escalabilidade e melhor desempenho em cada plataforma

Nessa seção, apresentamos os resultados onde o equipamento e o funcionamento do código é avaliado instanciado um número de threads diferente para cada execução, sendo possível processar diferentes partes da imagem para criar novos arquivos em paralelo. A Figura 3 apresenta o speedup obtido na execução da aplicação no sistema. A aplicação atingiu um speedup de cerca de 2.5x mais lento em relação a sua execução serial, rodando no mesmo processador.

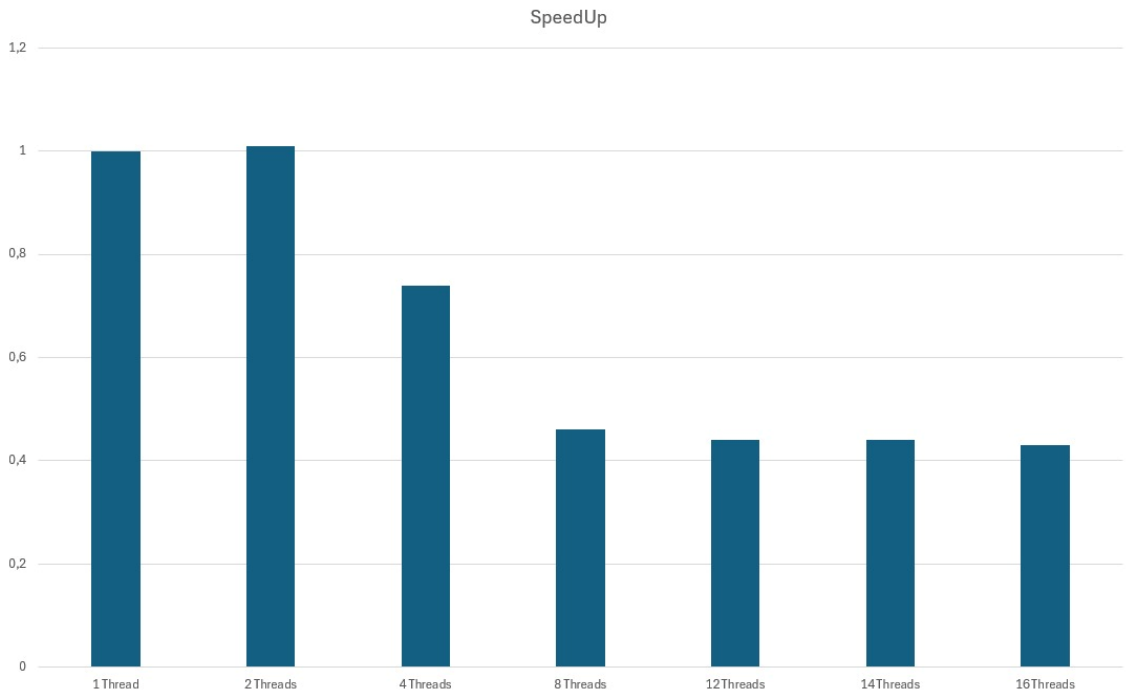

***Figure 3*. Speedup pela quantidade de threads utilizadas**

A tabela 2 apresenta resultados comparativos do multiprocessamento avaliado no mesmo processador. Como visto, o funcionamento com 16 threads atingiu o pior eficiência entre todas. Esse resultado é consequência de um efeito de gargalo que acontece ao utilizar muitos núcleos, que são acoplados ao mesmo subsistema de memória no sistema, devido ao fato das operações serem intensivas em dados. Caso conseguíssemos utilizar o ProcessPoolExecutor ao invés do ThreadPoolExecutor da biblioteca “concurrent.futures” do python, provavelmente conseguiríamos resultados melhores de desempenhos, já que são criados ambientes que funcionam de forma independente para cada núcleo, não havendo limitação também do GIL (Global Interpreter Locker) do Python, que acaba impedindo de ter uma experiência com 100% de aproveitamento de paralelização com algumas bibliotecas. Não foi possível utilizar ProcessPoolExecutor por conta da limitação de memória, já que criar um ambiente diferente e individual para o funcionamento do código requer maiores recursos de memória RAM.

Equipamento	Threads	Tempo (min:ss:ms)	Speedup	Eficiência
AMD Ryzen 7 5700U	1	1:06:66	1	100%
AMD Ryzen 7 5700U	2	1:5:30	1.016	50,8%
AMD Ryzen 7 5700U	4	1:29:29	0.743	18.57%
AMD Ryzen 7 5700U	8	2:22:26	0.465	5.81%
AMD Ryzen 7 5700U	12	2:30:97	0.440	3.66%
AMD Ryzen 7 5700U	14	2:30:99	0.440	3.14%
AMD Ryzen 7 5700U	16	2:32:92	0.434	2.71%

5. Conclusão

Baseado nos resultados obtidos, podemos concluir que a aplicação de detecção de navios em imagens grandes utilizando multiprocessamento apresenta desafios significativos e resultados variáveis. A partir da avaliação experimental realizada no sistema com um processador AMD Ryzen 7 5700U, observamos que o speedup alcançado, em comparação com a execução sequencial, não foi tão significativo quanto o esperado inicialmente. Na verdade, com o aumento do número de threads, houve uma queda no desempenho relativo, indicando a presença de gargalos de memória e limitações de escalabilidade dentro do próprio sistema.

Os resultados mostram que, enquanto o uso de múltiplos threads pode reduzir o tempo de processamento em alguns casos, a eficiência do processamento diminuiu à medida que mais threads foram utilizados. Isso se deve principalmente à natureza intensiva em dados da aplicação e às limitações do subsistema de memória compartilhada entre os núcleos do processador.

Além disso, a comparação entre diferentes quantidades de threads revelou que, para este cenário específico, o uso de um número moderado de threads (entre 2 e 4) pode

proporcionar um melhor equilíbrio entre tempo de execução reduzido e eficiência de processamento.

5.1 Trabalhos futuros

Para melhorar os resultados e explorar mais profundamente o potencial de processamento paralelo nesta aplicação, sugerimos os seguintes trabalhos futuros:

1. **Otimização de Algoritmos**: Investigar algoritmos mais eficientes para detecção de objetos que possam aproveitar melhor o paralelismo oferecido pelos múltiplos núcleos.
2. **Uso de GPU**: Explorar o uso de GPU para processamento paralelo, o que pode oferecer vantagens significativas em termos de desempenho para aplicações intensivas em computação.
3. **Gerenciamento de Memória**: Desenvolver estratégias mais eficientes de gerenciamento de memória para lidar com o aumento do volume de dados processados em paralelo.
4. **Escalonamento em Cluster**: Avaliar a implementação da aplicação em um ambiente distribuído, como um cluster de computadores, para melhorar a escalabilidade e o desempenho geral.

Em suma, embora o multiprocessamento tenha mostrado potencial para acelerar o processamento de imagens grandes, sua implementação eficaz requer considerações cuidadosas sobre as características específicas da aplicação e das plataformas de hardware utilizadas. A continuidade desses estudos é crucial para avançar no campo da análise de dados de alta resolução, beneficiando áreas como monitoramento marítimo e ambiental, entre outras aplicações científicas e industriais.

Agradecimentos: Queremos agradecer a universidade UniEuro por nos proporcionar um ambiente de estudos adequado. Agradecemos também ao nosso professor Rafael Ramos, que apesar dos projetos desafiadores, evoluímos muito em conhecimento e experiência no geral.