julio cesar sanches de souza
joao victor pimenta lopes

Contador de Estrelas

objetivo

Este projeto tem como objetivo desenvolver uma solução eficiente para a contagem de estrelas em imagens astronômicas de alta resolução, utilizando técnicas de computação paralela. O problema surge da necessidade de processar imagens extremamente grandes, que demandariam muito tempo em processamento sequencial. A solução proposta faz uso de paralelismo com Python e multiprocessing, simulando conceitos de MPI (Message Passing Interface) para acelerar o processamento por meio da divisão de tarefas.

Versão Sequencial

- **A imagem é dividida em vários tiles (blocos menores).**
- **Cada tile é processado de forma linear para detectar as estrelas.**
- **A detecção é feita por meio de pré-processamento com filtros, binarização e identificação dos centróides das estrelas.**
- **Inclui um atraso artificial por tile para simular processamento custoso.**

Descrição do Problema e Justificativa

O processamento de imagens astronômicas de grandes dimensões apresenta gargalos computacionais quando realizado de forma sequencial, gerando tempos de espera elevados e alto consumo de recursos. Portanto, é necessário aplicar estratégias de paralelização para:

- **Reduzir o tempo de execução**
- **Melhorar o uso dos recursos computacionais**
- **Tornar o processamento de dados escalável e eficiente**

Descrição da Solução

A solução foi criar um código que funcione em paralelo para dividir a tarefa e assim conseguir resultados mais eficientes e rápidos.

e assim surge a versão Paralela utilizando MPI

Versão Paralela

A lista de tiles é distribuída entre múltiplos processos.

Cada processo realiza a detecção de estrelas em seu subconjunto de tiles de forma independente.

Ao final, os resultados são reunidos e combinados em uma imagem de saída com as detecções sobrepostas.

Estrutura dos Códigos

- **imagem.jpg → Imagem original usada no conversor de imagem.**
- **imagem.tiff → é gerada e puxada pelo contador**
- **contadordeestrela.py → Processamento linear.**
- **contadorparalelo.py → Processamento paralelo.**
- **conversordeimagem.py → Gera imagem grande em formato TIFF.**
- **requirements.txt → Apresenta todas as bibliotecas que necessitam ser usadas.**
- **performances_analysis.html → Página HTML com os dados.**

Imagem original: 7001x4001px, canais: 3, ~80.14 MB por cópia
Objetivo: ~20 GB → 255.55 cópias → grade 16 x 16 = 256 cópias
Imagem final: (64016, 112016, 3), tamanho na memória: 20.04 GB

imagens

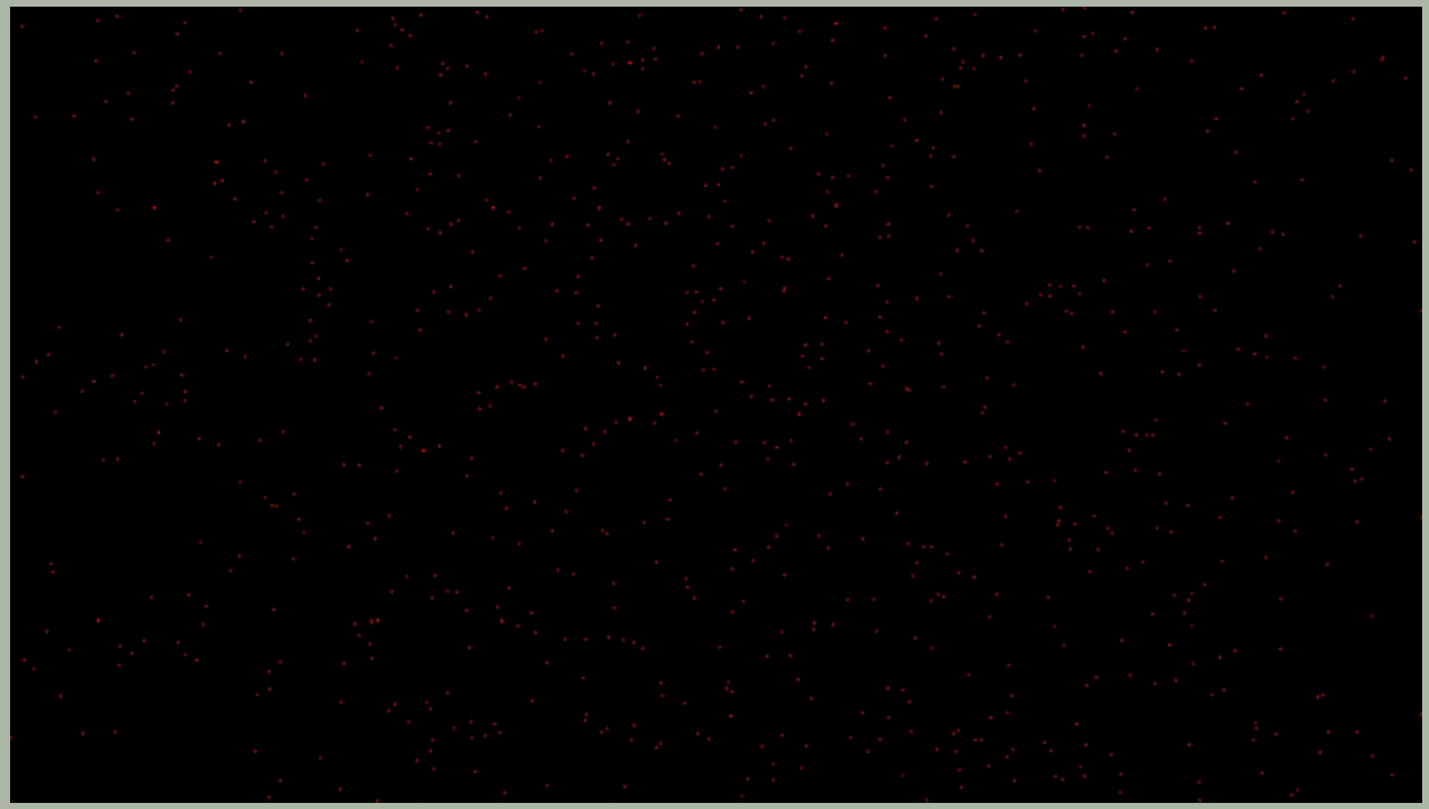


imagem segmentada com as estrelas capturadas

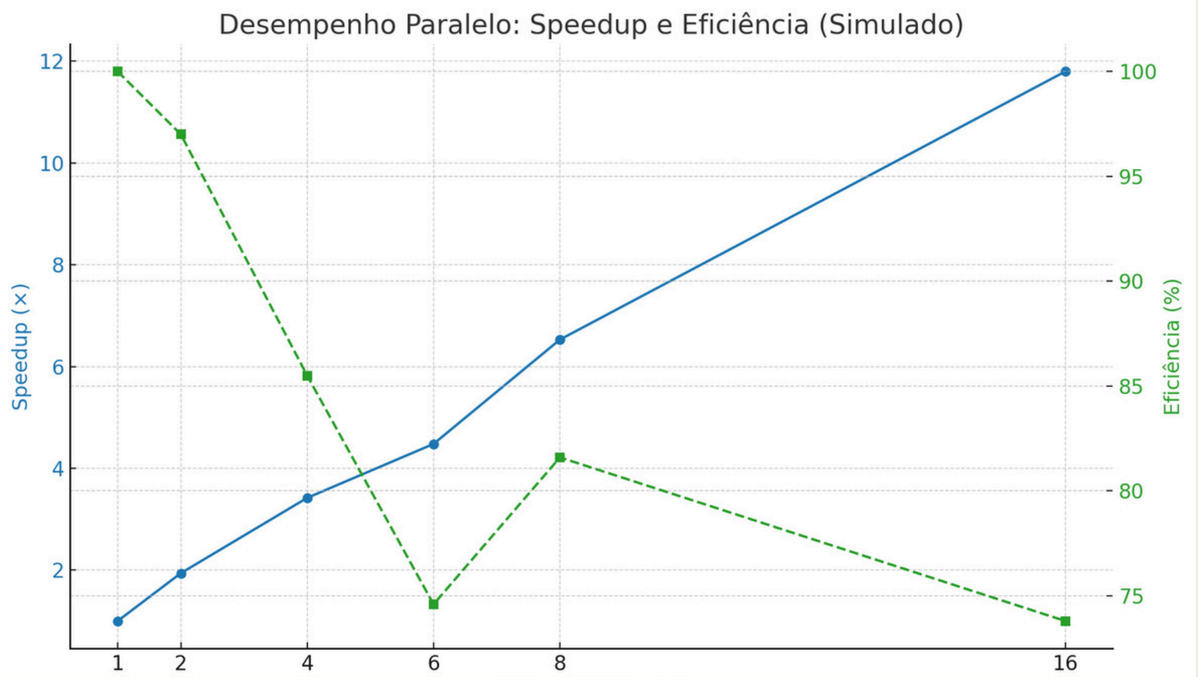
SPEED-UP e EFICIENCIA

threads	Tempo (s)	Tempo (min:seg)	Speedup (×)	eficiencia(%)
1	149.973	24:59	100	100
2	77.198	12:51	194	96
4	43.776	07:17	342	86
6	33.500	05:35	448	74
8	22.960	03:49	653	85
16	12.710	02:07	1.180	68

Faixa de threads	O que aconteceu?
1 → 4	Ganhos lineares, alta eficiência: até 85%. Ideal.
6 → 8	Boa escalabilidade, eficiência ainda forte.
16 threads	Alto desempenho, speedup excelente (11.8×), mesmo com eficiência abaixo de 100%. Comum e esperado devido à sobrecarga da paralelização.

✅ Processamento Concorrente Concluído: 238739 estrelas detectadas. Imagem de saída salva em: resultado_estrelas_concurrent.tiff

✅ Processamento Concorrente Concluído: 438739 estrelas detectadas em 771.98 segundos. Imagem de saída salva em: resultado_estrelas_concurrent.tiff



Conclusão

O paralelismo proporcionou ganhos expressivos de desempenho, reduzindo significativamente o tempo de processamento. Observa-se que a eficiência diminui conforme aumentamos o número de threads, comportamento esperado devido à sobrecarga de comunicação e gerenciamento dos processos. A solução proposta se mostrou escalável e eficaz para processar grandes imagens astronômicas, validando a importância da computação paralela na área de ciência de dados e astronomia.

A paralelização reduziu o tempo de execução de 25 minutos para pouco mais de 2 minutos.

O código se mostrou escalável e muito eficiente até 16 nucleos.

A eficiência diminui com o número de nucleos, mas isso é esperado devido à:

sobrecarga de comunicação entre processos,

tempo ocioso em alguns núcleos,

e limitações físicas do hardware (como cache, I/O, etc).

A versão paralela do algoritmo obteve um speedup de até 11.8× com 16 nucleos, mantendo uma eficiência superior a 70%, o que demonstra uma ótima escalabilidade e aproveitamento dos recursos computacionais disponíveis.