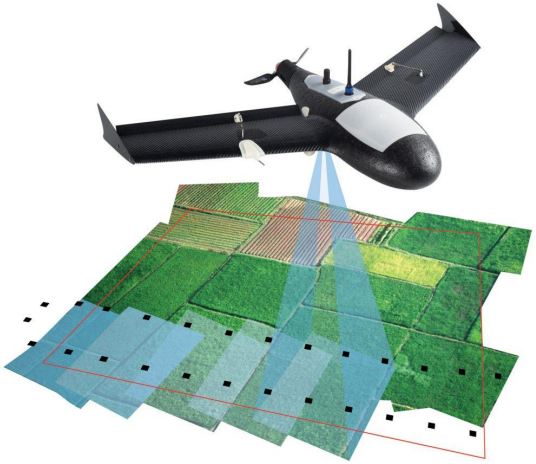
Clusterização K-Means Paralelo Aplicado na Classificação de Alvos em Imagens de Alta Resolução

Luís Paulo Manfré Ribeiro

Sumário

- Motivação
- Introdução
  - PAD, PDI
  - GPUs
  - CUDA
  - Classificação de Imagens e K-Means
- K-Means Paralelo
- Metodologia e Resultados
- Conclusões

Motivação



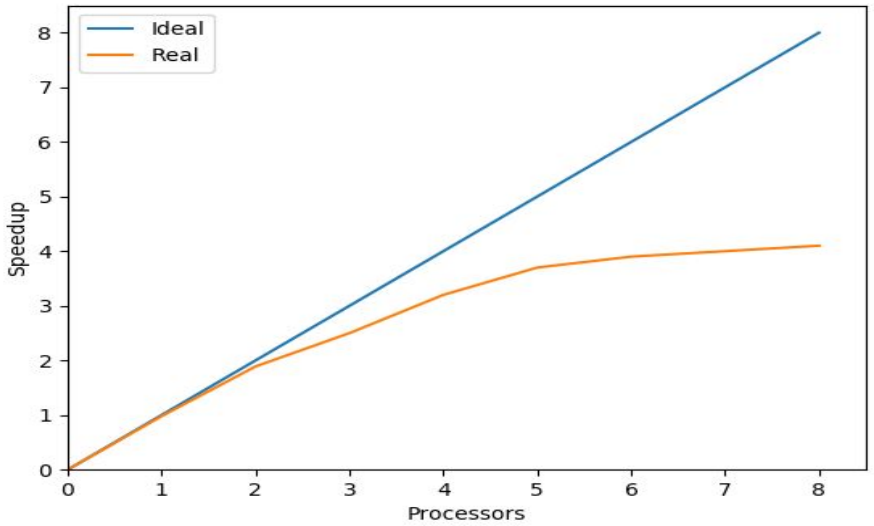
Introdução

- PAD (Processamento de Alto Desempenho) aplicado em PDI (Processamento Digital de Imagens)
  - Aumentar velocidade de processamento com paralelismo e computação distribuída.
- ID (Imagem digital)
  - Representação 2D de uma imagem;
  - Matriz de valores digitais chamados pixels.
- PDI
  - Aplicação de algoritmos computacionais em IDs a fim de obter determinado objetivo;
  - Exemplo: classificação de regiões em uma dada ID.

Introdução

- Speedup (S_P) é a relação entre o tempo de processamento serial (T_{Ser}) e o paralelo (T_P) dada pela seguinte fórmula:

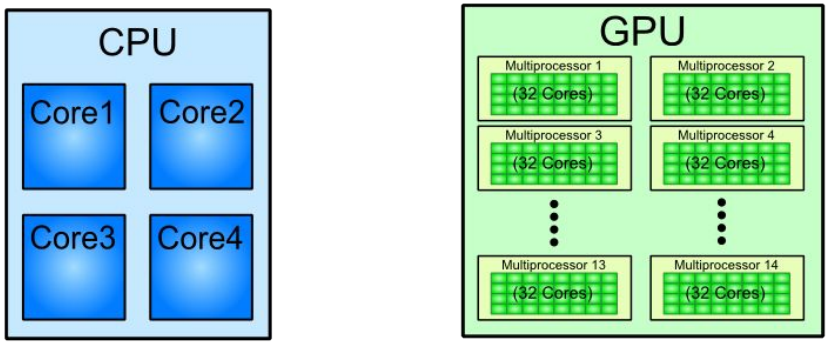
$$S_P = \frac{T_{Ser}}{T_P}$$



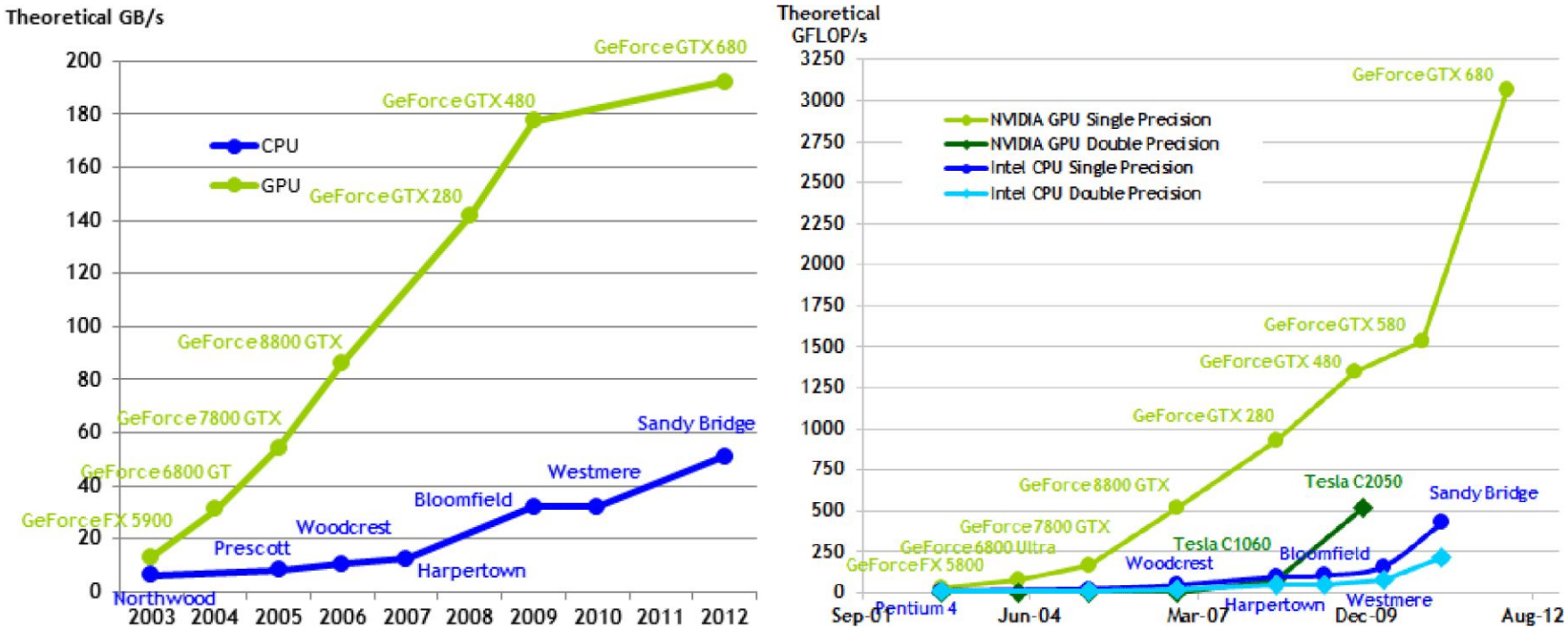
Introdução

- GPUs (*Graphics Processing Unit*)
  - Melhor relação desempenho / custo
  - Melhor relação desempenho / tamanho
  - Melhor relação desempenho / consumo de energia

CPU/GPU Architecture Comparison



Introdução



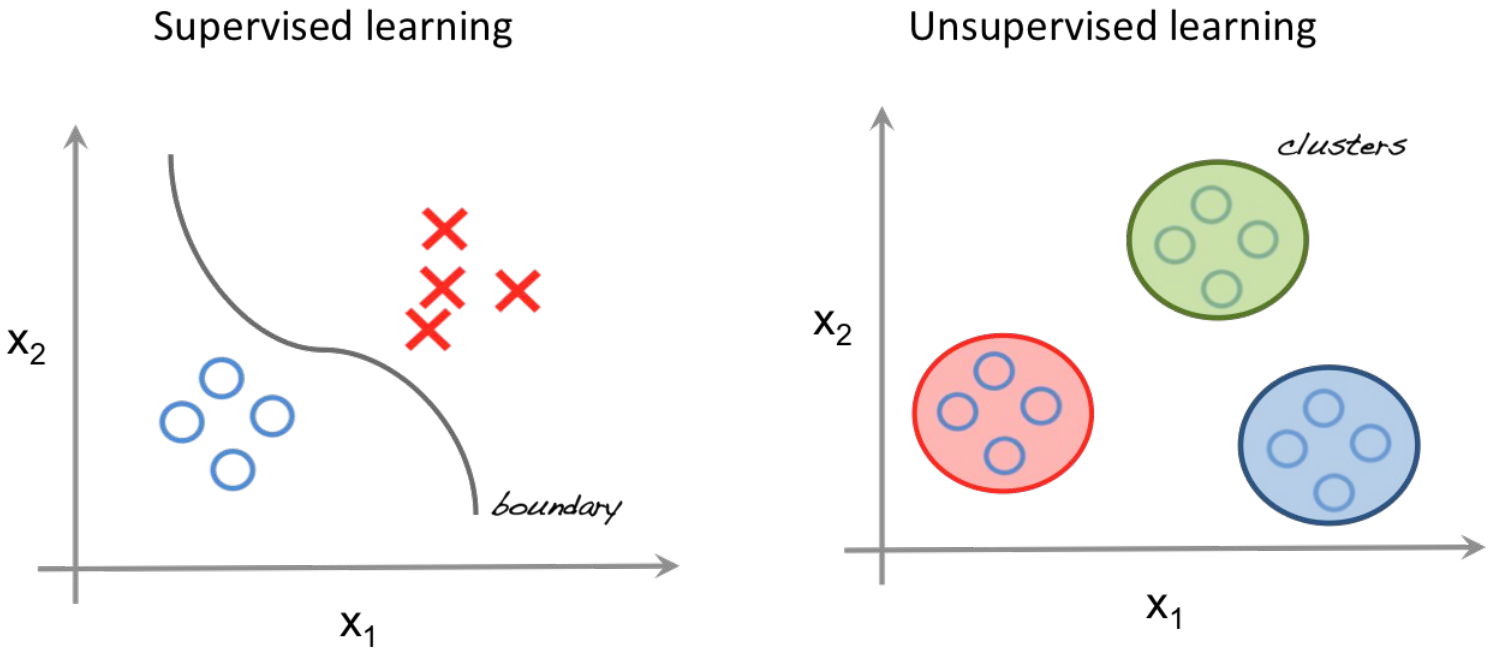
Comparativo da evolução CPU versus GPU

Introdução

- NVIDIA CUDA
  - API (*Application programming interface*);
  - Utiliza a linguagem C.

Introdução

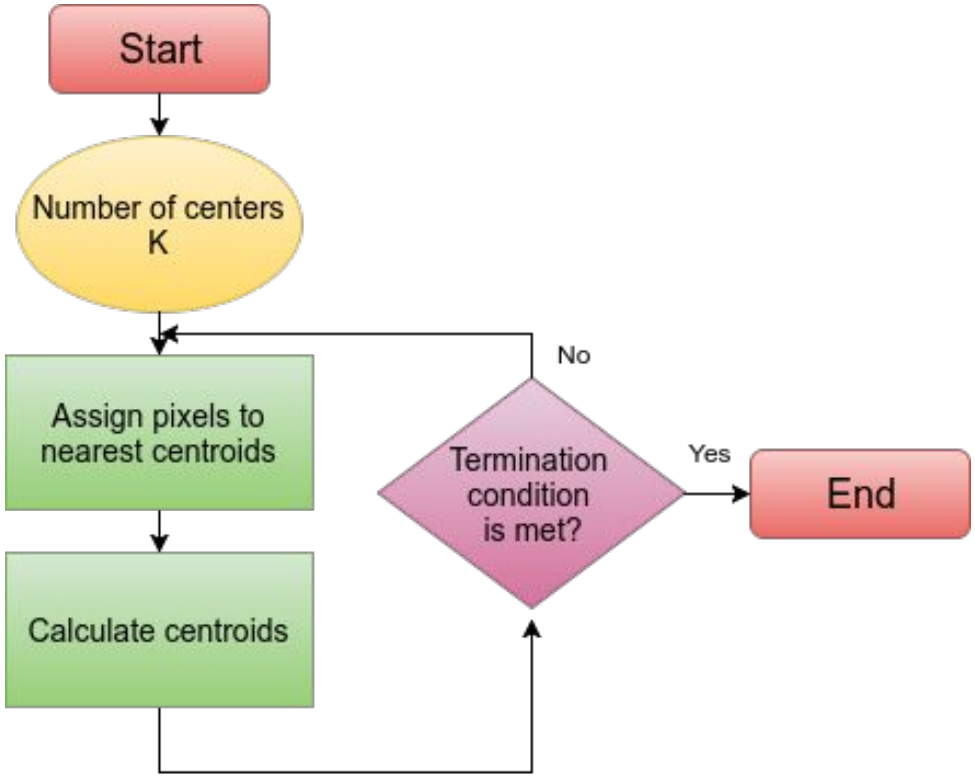
- Classificação de imagens



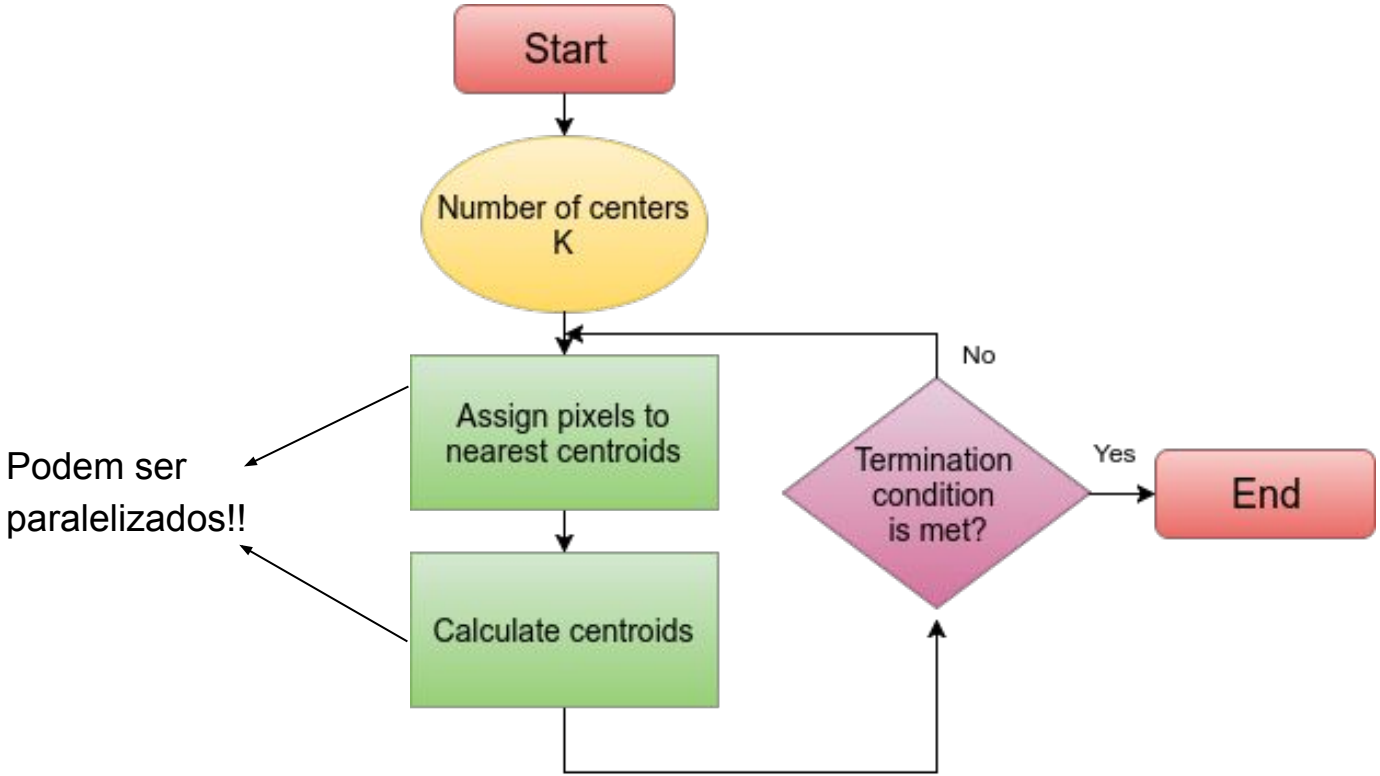
Introdução

- Clusterização K-Means
  - Um dos métodos mais comuns de propósito geral que utiliza Aprendizagem Não Supervisionada (clusterização);
  - Encontra clusters de tamanhos similares;
  - Número de clusters (*k*) é especificado.

Abordagem da Clusterização K-Means Paralelo



Abordagem da Clusterização K-Means Paralelo



Abordagem da Clusterização K-Means Paralelo

Primeira abordagem: própria implementação

Segunda abordagem:

Versão modificada da implementação em github.com/bryancatanzaro/kmeans

Utiliza a biblioteca CUDA Thrust github.com/thrust/thrust

Metodologia

Os seguintes softwares e hardwares foram utilizados nos testes:

- OpenCV versão 3.2.0;
- CUDA versão 8.0;
- Intel CPU I7-6500U 2.5 GHz com 8GB de RAM;
- NVIDIA GPU GeForce 920MX com 2GB de memória.

Testes seriais: OpenCV

Testes paralelos: OpenCV + CUDA

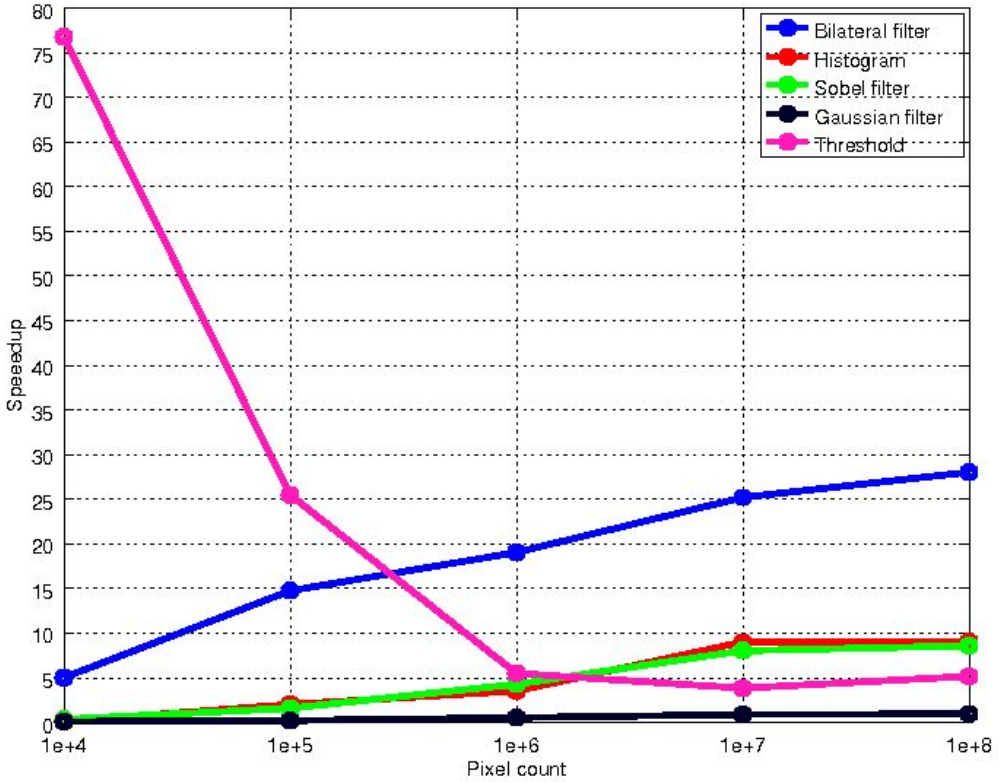
Testes de performance e consistência dos resultados foram feitos!

Amostra de Dados

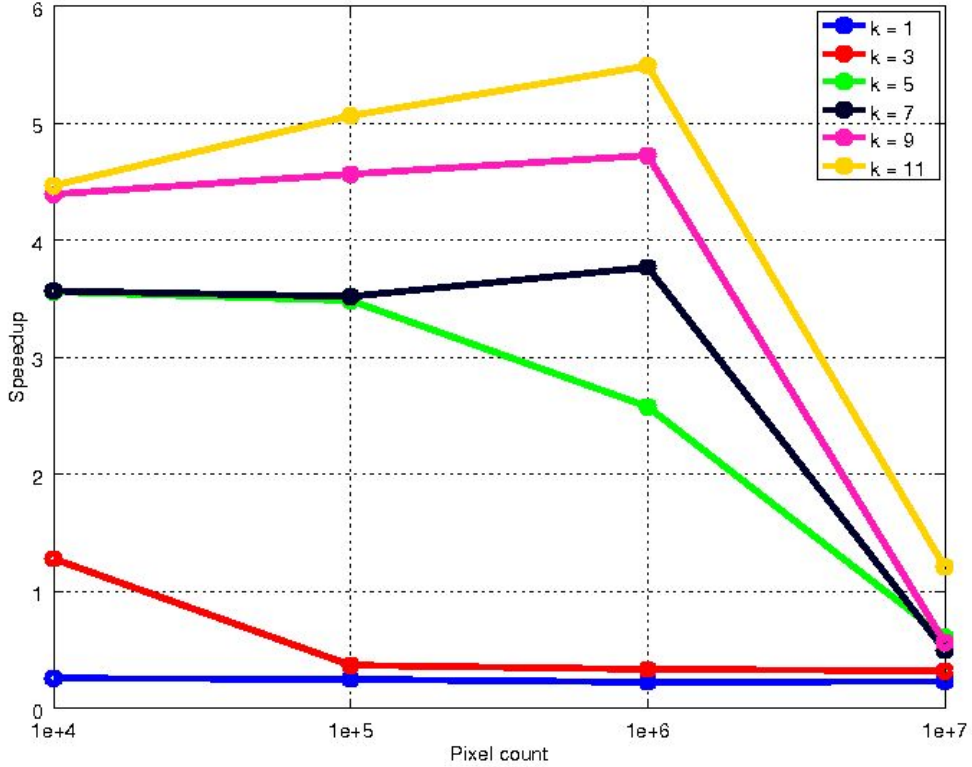
Amostra de Dados

	Resolução	Tamanho (MB)	Bits	N. de bandas
VANT	10197×24786	964.14	8 (0-255)	4
Lena	512×512	0.480	8 (0-255)	3
RJ-1	1000×2000	22.89	8 (0-255)	3

Resultados de performance (algoritmos de PDI)

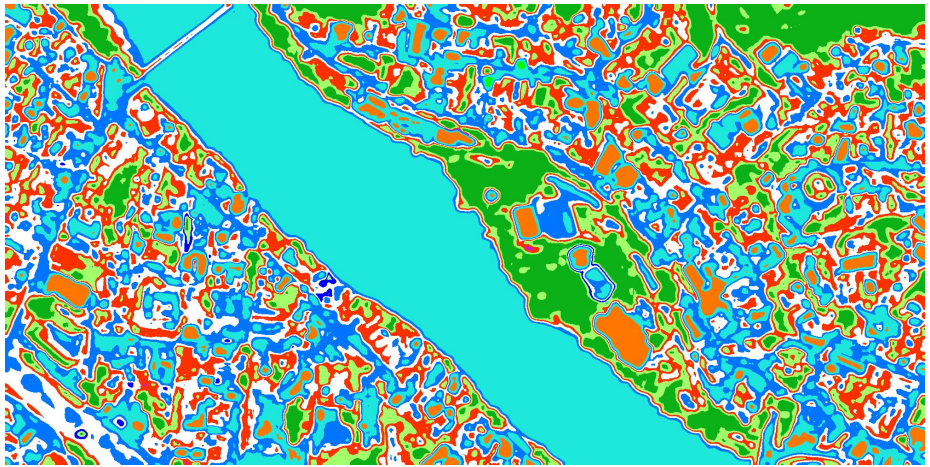


Resultados de performance (K-Means)

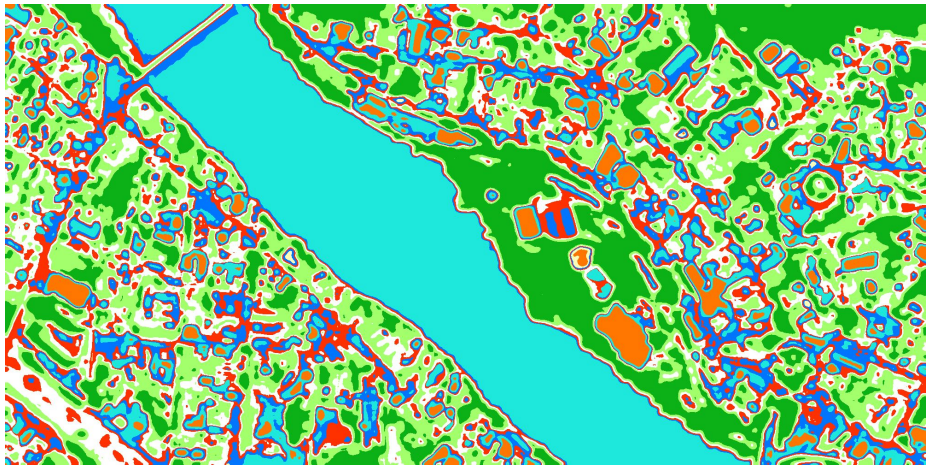


Resultados do K-Means

Paralelo



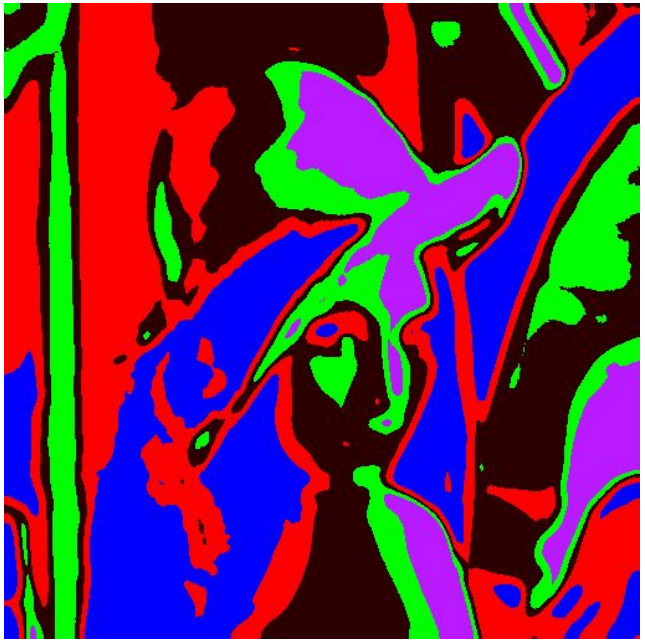
Serial



Resultados do K-Means

Paralelo

Serial



Conclusões

- Consistência dos resultados;
- Speedups;
- Problemas de memória, poderiam ser solucionados com hardware melhor;
- Projetos futuros (OpenCL, estudo dos tempos de transferência entre host (CPU) e device (GPU)).

Obrigado!

Códigos utilizados neste trabalho: https://github.com/izzylp/opencv-cuda