

Projeto 4-Segmentação de imagens

Felipe Frid Buniac | ID:15610505

Dezembro 2018

Contents

1	Resumo de projeto	1
2	Introdução	2
2.1	Segmentação de Imagens	2
2.2	Problema	2
3	Descrição dos Programas	3
3.1	Sequencial	3
3.2	Em GPGPU	3
4	Resultados e Análises	3
4.1	Teste Inicial de comprovação	3
4.1.1	Tempo Sequencial	5
4.1.2	Tempo em GPGPU	5
4.2	Evolução de um teste para provar um bom resultado	5
4.3	Resultados	6
4.3.1	Imagen 1: Neve	6
4.3.2	Imagen 2: Laranja	6
4.3.3	Imagen 3:Ski	7
4.4	Hardware Utilizado	8
5	Conclusão	9

1 Resumo de projeto

Este projeto aplica os conceitos de GPGPU em um problema real de segmentação de imagens. A ideia principal é a implementação de um programa utilizando a biblioteca nvGraph para a busca de caminhos mínimos. Será necessário ler a imagem, receber múltiplos pontos para as chamadas sementes de frente e sementes de fundo e com isso montar o grafo a partir da imagem original usando nvGraph e calcular os caminhos mínimo criando como saída uma imagem máscara com cor branca para a frente e preta para o fundo. Em um segundo passo a implementação de um filtro de bordas utilizando CUDA C e a construção de grafos com a função de Single Source Shortest Path (SSSP) utilizando a imagem de bordas é aplicada. Para as medições de tempo e desempenho do programa foi utilizado cudaEvent[12].

2 Introdução

2.1 Segmentação de Imagens

Uma segmentação de imagem é sua divisão em regiões conexas, contendo objetos de interesse ou o fundo da imagem. A segmentação iterativa de imagens é baseada na adição de marcadores em objetos de interesse para direcioná-los do fundo (que contém todos os outros objetos e plano de fundo da imagem). A figura abaixo exemplifica este tipo de interação.

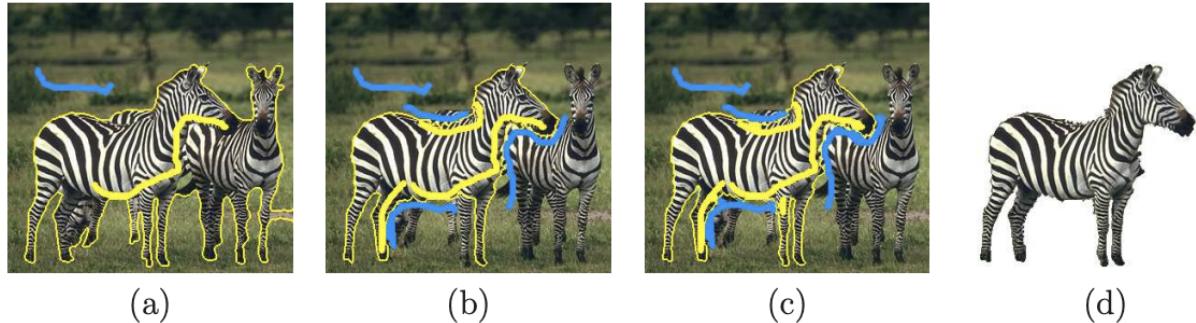


Figure 1: As figuras de (a-d) mostram a sequência de intervenções do usuário para segmentação final, onde o usuário adiciona marcadores para reconhecimento enquanto o computador exibe os resultados da delineação em tempo real[1].

Um algoritmo simples e muito eficaz para este problema é o IFT[2] (Image Foresting Transform), que representa a imagem usando um grafo e trata a segmentação de imagens como um problema de caminho mínimo entre os vértices (pixels) semente e todos os outros pontos da imagem. Cada pixel (vértice) está conectado com os 4 pixels (vértices) adjacentes. O custo de cada aresta pode ser dado de duas maneiras:

1. Magnitude da diferença entre os valores dos pixels na imagem original;
2. Magnitude da diferença entre os valores dos pixels no imagem de bordas (gradiente morfológico ou Laplace)

Mais sobre IFT pode ser encontrado em [3, 4, 5].

2.2 Problema

O problema com a seção acima é que a segmentação de imagens de maneira iterativa necessita de resultados quase instantâneos para que o processo seja agradável para o usuário. Uma solução para este problema seria utilizar bibliotecas de processamento de grafos em GPGPU para tentar acelerar este processo para imagens de alta resolução. Este processo seria dividido em 3 etapas

1. Implementar o processo completo em um algoritmo sequencial eficiente;
2. Usar ferramentas de GPGPU para acelerar cada etapa do processo sequencial.
3. Quantificar o tempo que cada etapa do processamento leva e comparar com a implementação sequêncial

Neste projeto utilizaremos a biblioteca nvGraph [6] com a técnica de SSSP [7, 8] que já existe uma implementação desta com nvGraph [9, 10].

3 Descrição dos Programas

3.1 Sequencial

O código sequencial é uma implementação sequencial do algoritmo Single Source Shortest Path e sua utilização para a criação de uma segmentação de imagens em frente e fundo. O projeto aceita múltiplas sementes de frente de fundo, o algoritmo SSSP é executado duas vezes (uma para frente e uma para fundo), na função SSSP cada ponto é adicionado a fila de prioridade mais de uma vez, mas só é processado se houver chance de melhorar seu caminho, o leitor de PGM é extremamente limitado e não suporta comentários no PGM.

3.2 Em GPGPU

Ao utilizar a biblioteca de nvGraph passamos a utilizar gráficos e utilizar o formato CSC [11]. Para isso devemos gerar 3 listas importantes que compõem a CSC do SSSP com nvGraph. Geramos uma lista de "weights" (peso de cada aresta), "source indices" (a raiz de cada aresta, de onde esta sai) e "destination offsets" (intervalo na lista de source indices que aponta para um nó), informações para a montagem do gráfico. Com isso temos uma função que analisa cada pixel gerando esses valores de acordo com pixels vizinhos e são utilizados como entrada na função de SSSP que é executada duas vezes (uma para frente e uma para o fundo) com as respectivas sementes definidas na entrada pelo usuário.

4 Resultados e Análises

Foram feito análises de tempos no programa sequencial e no programa rodando na GPGPU. O mecanismo utilizado para fazer a medição de tempo é um timer que está integrado ao mecanismo de execução de código da GPGPU, que permite registrar a ocorrência de eventos na GPGPU e calcular quanto tempo passou entre pares de ocorrências. Foram analisados 4 tipos de medições de desempenho (usando cudaEvent) para cada programa:

- **montagem do grafo**
- **cálculo dos caminhos mínimos**
- **montagem da imagem segmentada**
- **tempo total do programa**

4.1 Teste Inicial de comprovação

Foi feito um teste inicial para comprovação do funcionamento dos programas.

Foi utilizada a imagem abaixo:

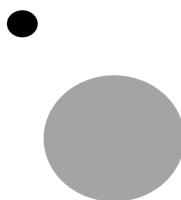


Figure 2: Imagem do teste inicial, um exemplo simples para segmentação

O teste foi dividido em 3 partes:

1. colocar uma semente dentro do círculo grande e uma semente de fundo fora dos dois círculos.
2. colocar uma semente dentro do círculo pequeno e uma semente de fundo fora dos dois círculos.
3. colocar duas sementes de frente, uma dentro do círculo grande e uma semente dentro do pequeno e uma semente de fundo no background branco.

Resultado:

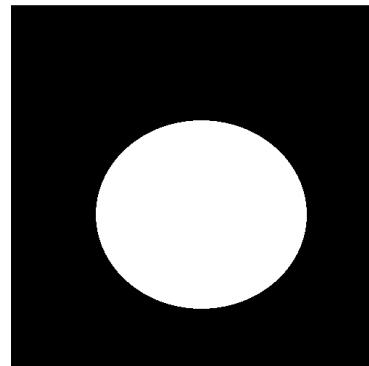


Figure 3: Resultado para o primeiro teste

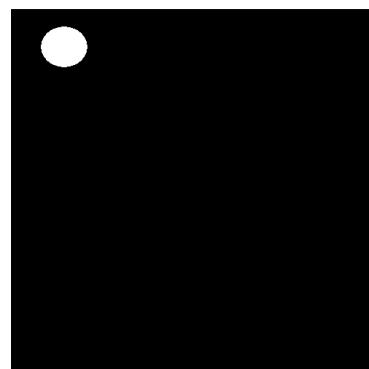


Figure 4: Resultado para o segundo teste

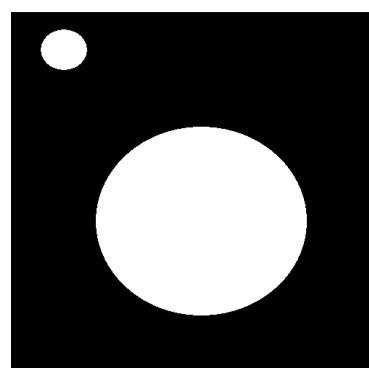


Figure 5: Resultado para o terceiro teste

Para a mesma entrada e com mesmo número de sementes para a mesma imagem tivemos o resultado de tempo como abaixo:

4.1.1 Tempo Sequencial

- Sementes foreground: 1
- Sementes background: 1
- Tempo total de execução foi: 7407.96
- Tempo de cálculo de caminhos mínimos foi: 893.283
- Tempo de montagem da imagem segmentada foi: 0.789056

4.1.2 Tempo em GPGPU

- Sementes foreground: 1
- Sementes background: 1
- Tempo total de execução foi: 7408.03
- Tempo de montagem de gráfo foi: 0.001888
- Tempo de cálculo de caminhos mínimos foi: 886.636
- Tempo de montagem da imagem segmentada foi: 0.790176

Comparação de tempo para teste inicial				
	Montagem Gráfo (ms)	Montagem Img Segmentada (ms)	Cálculo Caminhos Mínimos (ms)	Total (ms)
SEQUENCIAL	N/A	0.789056	893.283	7407.96
GPU	0.0018888	0.790176	886.636	7408.03

Figure 6: Tabela comparando resultados do teste preliminar

Os testes foram feitos em 3 imagens com diferentes resoluções. Os testes tiveram evoluções com número de sementes.

4.2 Evolução de um teste para provar um bom resultado

Para provar que o programa de fato faz uma segmentação de qualidade um dos exemplos tiveram 4 iterações para chegar em um resultado com qualidade alta (Neste exemplo fizemos a pessoa ficar em preto e o fundo em branco . Na imagem abaixo o objetivo era segmentar a pessoa (sem seu cachorro) de todo o resto da imagem.



Figure 7: Imagem Original

Os resultados ficaram como abaixo:

Para a imagem acima temos:



Figure 8: Evolução da Imagem Original

- a): Imagem Original
- b): 8 seeds background e 8 de foreground (tempo total GPGPU:314635 ms)
- c): 14 seeds background e 8 de foreground (tempo total GPGPU:303474 ms)
- d): 20 seeds background e 8 de foreground (tempo total GPGPU:314192 ms)
- e): 25 seeds background e 8 de foreground (tempo total GPGPU:316212 ms)

4.3 Resultados

4.3.1 Imagem 1: Neve

- **Tamanho da imagem:** 6,916,628 bytes (6.9 MB)



Figure 9: Resultado da imagem

Comparação de tempo para imagem NEVE						
	Montagem Gráfo (ms)	Montagem Img Segmentada (ms)	Cálculo Caminhos Mínimos (ms)	Total (ms)	Background Seeds	Foreground Seeds
SEQUENCIAL	N/A	5.56758	15260.2	337160	8	25
GPU	0.002336	5.58771	20868.6	316212	8	25

Figure 10: Tabela com comparação de resultado

4.3.2 Imagem 2: Laranja

- **Tamanho da imagem:** 191,534 bytes (193 KB)

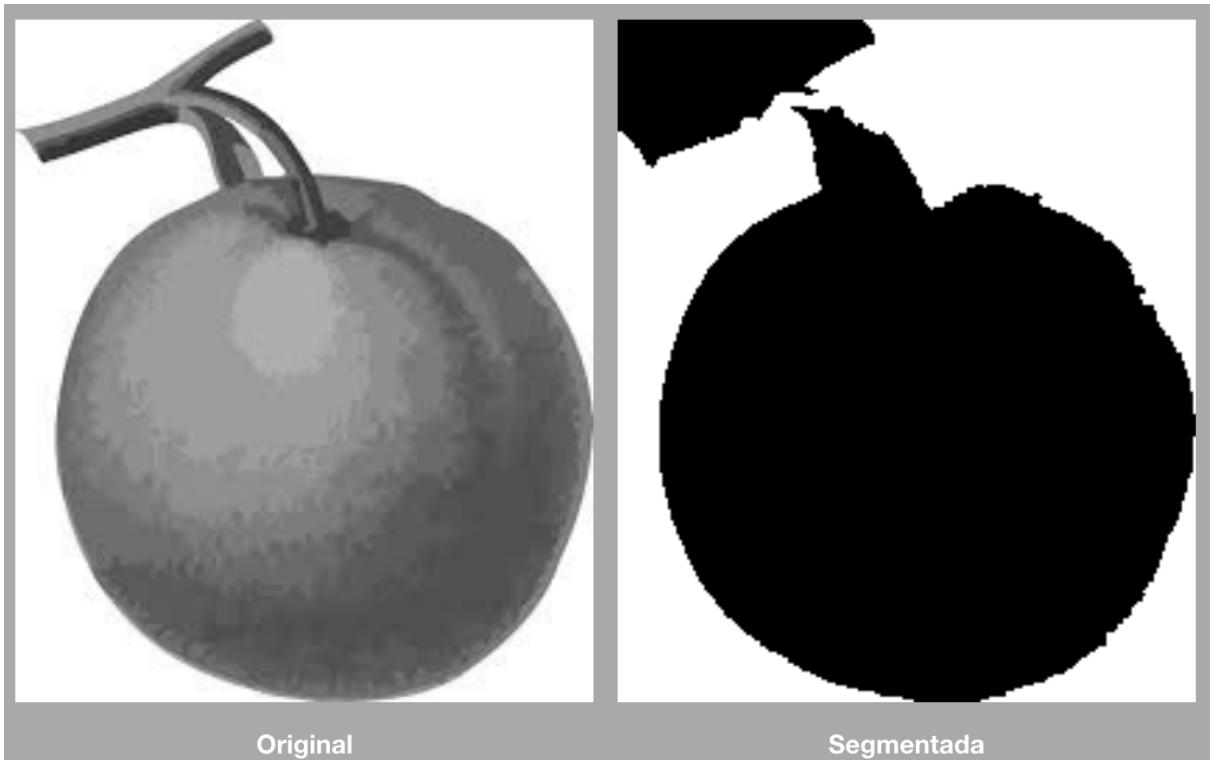


Figure 11: Resultado da imagem

Comparação de tempo para imagem LARANJA						
	Montagem Gráfo (ms)	Montagem Img Segmentada (ms)	Cálculo Caminhos Mínimos (ms)	Total (ms)	Background Seeds	Foreground Seeds
SEQUENCIAL	N/A	0.17104	484.796	1127.29	13	16
GPU	0.00224	0.169984	483.695	1110.78	13	16

Figure 12: Tabela com comparação de resultado

4.3.3 Imagem 3:Ski

- **Tamanho da imagem:** 3,668,591 bytes (4.3 MB)



Figure 13: Resultado da imagem

Comparação de tempo para imagem SKI						
	Montagem Gráfo (ms)	Montagem Img Segmentada (ms)	Cálculo Caminhos Mínimos (ms)	Total (ms)	Background Seeds	Foreground Seeds
SEQUENTIAL	N/A	3.06157	3742.14	90415.7	9	6
GPU	0.002464	3.08294	3740.8	90530.1	9	6

Figure 14: Tabela com comparação de resultado

4.4 Hardware Utilizado

O hardware utilizado neste projeto foi uma instância Linux da AWS.

```
ubuntu@ip-172-31-49-34:/home/ec2-user/folder$ sudo lshw
*-cpu
  product: Intel(R) Xeon(R) CPU E5-2686 v3 @ 2.38GHz
  vendor: Intel Corp.
  physical id: 0
  version: 4.2.0
  serial: ecc279a6-2de-3439-995b-85273bc21dc3
  width: 64 bits
  capabilities: smbios-2.7 dmi-2.7 vsyscall128
  configuration: boot=normal uuid=698878C-CED0-3934-895B-85273BC21DC3
*-cpu
  description: Motherboard
  physical id: 0
  bus info: pci@0000:00:00.0
  capabilities: pci-addr
  configuration: latency=64
+-cpu0
  description: CPU
  product: Intel(R) Xeon(R) CPU E5-2686 v3 @ 2.38GHz
  vendor: Intel Corp.
  physical id: 400
  bus info: pci@0000:00:00.0
  slot: CPU 0
  size: 249MHz
  capacity: 249MHz
  capabilities: pci
  configuration: latency=64
+-cpu1
  description: CPU
  vendor: Intel
  physical id: 400
  bus info: pci@0000:00:00.1
  slot: CPU 1
  size: 249MHz
  capacity: 249MHz
  capabilities: pci
  configuration: latency=64
+-cpu2
  description: CPU
  vendor: Intel
  physical id: 400
  bus info: pci@0000:00:00.2
  slot: CPU 2
  size: 249MHz
  capacity: 249MHz
  capabilities: cpufreq
+-cpu3
  description: CPU
  vendor: Intel
  physical id: 400
  bus info: pci@0000:00:00.3
  slot: CPU 3
  size: 249MHz
  capacity: 249MHz
  capabilities: cpufreq
+-mem0
  description: System Memory
  physical id: 0
  bus info: pcimem@0000:00:00.0
  size: 3968MB
+-dimm0
  description: DIMM RAM
  physical id: 0
  slot: DIMM 0
  size: 16GB
  width: 64 bits
+-dimm1
  description: DIMM RAM
  physical id: 1
  slot: DIMM 1
  size: 16GB
  width: 64 bits
+-pci
  description: Host bridge
  product: 440FX - 8244FX IIC [Natoma]
  vendor: Intel Corporation
```

Figure 15: Informações da instância AWS parte 1

```
ubuntu@ip-172-31-49-34:/home/ec2-user/folder$ sudo lshw
*-ide
  description: IDE interface
  product: 8237180 PIIX3 IDE [Natoma/Triton II]
  vendor: Intel Corporation
  physical id: 1
  bus info: pci@0000:00:01.1
  version: 00
  width: 32 bits
  clock: 33MHz
  capabilities: ide bus_master
  configuration: latency=64
  resources: ioremap:0x00000000-0x0000000f iport:3f6 iport:376 iport:c100(size=16)
*-bridge UNCLAIMED
  description: PCI bridge [PIIX4 ACPI]
  product: 8237456/8237458 PIIX4 ACPI
  vendor: Intel Corporation
  physical id: 2
  bus info: pci@0000:00:01.3
  version: 00
  width: 32 bits
  clock: 33MHz
  capabilities: bridge bus_master
  configuration: latency=64
+-display0 UNCLAIMED
  description: VGA compatible controller
  product: GD 5446
  vendor: VIA Technologies, Inc.
  physical id: 2
  bus info: pci@0000:00:02.0
  version: 00
  width: 32 bits
  clock: 33MHz
  capabilities: vga_controller bus_master
  configuration: latency=64
  resources: ioremap:0x00000000-0xffffffff memory:76004000-96004fff
*-network
  description: Ethernet interface
  physical id: 3
  bus info: pci@0000:00:03.0
  logical name: eth0
  version: 00
  serial: 00:0c:29:11:89:98
  width: 32 bits
  clock: 33MHz
  capabilities: ethernet bus_master cap_list
  configuration: broadcast driver=en drivername=en driverversion=2.6.1K ip=172.31.85.34 latency=0 link=yes multicast=yes
  resources: irq:79 memory:70000000-7000ffff memory:90000000-9000ffff
*-display1 UNCLAIMED
  description: VGA compatible controller
  product: GeForce GT 630 [NVIDIA]
  vendor: NVIDIA Corporation
  physical id: 3
  bus info: pci@0000:00:03.1
  version: 00
  width: 32 bits
  clock: 33MHz
  capabilities: vga_controller bus_master
  configuration: latency=248
  resources: ioremap:79 memory:70400000-747fffff memory:80000000-8fffffff memory:92880000-93fffff iport:c100(size=128)
*-generic0
  description: Unsigned class driver
  product: Generic Device
  vendor: XenSource, Inc
  physical id: 4
  bus info: pci@0000:00:04.0
  version: 01
  width: 32 bits
  clock: 33MHz
  capabilities: bus_master
  configuration: latency=0
  resources: ioremap:40000000-4000ffff
```

Figure 16: Informações da instância AWS parte 2

5 Conclusão

Analizando os tempos finais da segmentação das imagens sequencialmente e na GPGPU, podemos ver uma pequena melhora quando a imagem é segmentada utilizando nvGraph. A hipótese inicial deste projeto era de que a melhora seria maior. O que pode explicar esse tempo próximo entre o programa sequencial e o em GPGPU é o fato de que os dois programas rodam a função SSSP 2 vezes (uma para frente e outra para trás) ou a forma de copiar vetores estar atrasando o programa em GPGPU, alterando estes pontos o ganho poderia ser maior. Além do tempo, o programa roda com sucesso (segmenta de forma correta) e faz as segmentações com qualidade.

Os arquivos descritos neste relatório podem ser encontrados em : https://github.com/febuniac/SegImage_GPU.

References

- [1] <http://www.inf.ufrgs.br/sibgrapi2010/files/01.pdf>
- [2] <https://github.com/igordsm/supercomp/blob/master/gpu/projeto/enunciado.pdf>
- [3] <http://citeserx.ist.psu.edu/viewdoc/download?doi=10.1.1.374.8573&rep=rep1&type=pdf>
- [4] http://www.vision.ime.usp.br/~pmiranda/mo815_2s09/aulas/aula_ift.pdf
- [5] <http://www.ic.unicamp.br/~afalcao/mo815-grafos/ift04.pdf>
- [6] <https://docs.nvidia.com/cuda/nvgraph/index.html>
- [7] https://courses.cs.washington.edu/courses/cse373/01sp/Lect24_2up.pdf
- [8] <https://www.techiedelight.com/single-source-shortest-paths-dijkstras-a/>
- [9] <https://developer.nvidia.com/nvgraph1>
- [10] <https://developer.nvidia.com/discover/shortest-path-problem>
- [11] https://www.scipy-lectures.org/advanced/scipy_sparse/csc_matrix.html
- [12] <https://devblogs.nvidia.com/how-implement-performance-metrics-cuda-cc>