Segmentação de Imagem

Desempenhos variados conforme execução sequencial ou em GPU

Gabriel do Vale Rios, Engenharia de Computação. Insper-2018.

Introdução

Seja por motivos profissionais, ou apenas um *hobby*, é necessário que exista meios de se destacar áreas específicas em uma imagem. Destacar no caso seria segmentar uma região da imagem e destacá-la, a fim de se trabalhar somente com o que é desejável.

Assim, este trabalho busca comparar o desempenho de segmentação de imagem sequencial x GPU, utilizando da técnica de filtragem de borda, para maior precisão e melhor desempenho.

Implementação

Todo código citado neste documento pode ser baixado e executado a partir do endereço abaixo:

https://github.com/GabrielValeRios/superComp2018/tree/master/projeto4

Para a implementação e gestão de todo o código feito para esta simulação, se utilizou a linguagem C++, python e Cmake. O hardware utilizado para compilação e execução das simulações segue abaixo, junto de suas informações:

- Processador: Intel® Core™ i7-4500U CPU @ 1.80GHz 2.40GHz
- Memória (RAM): 16.0 GB
- Tipo de Sistema: Sistema Operacional de 64 bits, processador com base em x64, 4 cores
- Sistema Operacional utilizado: Distribuição Linux (Ubuntu 18.04 LTS)
- Placa Nvidia GeForce - 840M

O código busca segmentar uma imagem a partir de *Seeds*, que são pontos específicos na imagem que servem para diferenciar a região que se quer destacar do plano de fundo. Assim, podem ser fornecidas múltiplas seeds do tipo f_g, para destacar diferentes regiões, e múltiplas seeds do tipo b_g, para informar diferentes regiões que pertencem ao plano de fundo. Com a diferenciação da região desejada do plano de fundo, é utilizado o algoritmo de Dijkstra, para de fato separar as regiões. Tal algoritmo se traduz no trabalho como SSSP (*Single Source Shortest Path)*

Como imagem *source*, pode ser fornecida uma imagem com extensão .pgm. Está imagem, para que a separação das regiões seja facilitada, pode passar por uma filtragem de borda, onde a imagem resultante é igual à de entrada, porém com bordas que definem melhor regiões diferentes. Para isso,

foi utilizada a biblioteca “thrust”, responsável por transferir os dados a serem calculados para a GPU, e junto de ferramentas do CUDA, foi criado um kernel dedicado para executar a função de bordas em si. A execução de dados em GPU é mais rápida, justificando o aumento de performance pelas múltiplas threads que podem ser geradas na GPU. O ganho de tempo é justificado pela quantidade massiva de threads em si, e não apenas pela performance individual de cada thread.

- Arquivos em C++

Os arquivos deste projeto para gerar os executáveis, em GPU, são:

- Imagem.cpp, responsável pela leitura de uma imagem, geração de uma nova imagem e fornecer diversos dados sobre a imagem (tamanho, pixels, etc)
- Main.cpp, responsável por executar o algoritmo de SSSP da Nvidia nas imagens, medições de tempo para comparação de desempenho e montar os vetores necessários para o código da Nvidia.
- Border_filter.cu, para transformar uma imagem .pgm em outra imagem .pgm, com bordas.

Para gerar o executável sequencial:

- Imagem.cpp, idem ao explicado acima.
- Main.cpp, responsável por executar o algoritmo de SSSP sequencialmente.

A ideia é comparar o quão mais rápido o processamento em GPU é do que o processamento sequencial. Para isso, serão medidos os tempos de cálculo do SSSP e o tempo de montagem da imagem resultante. Tal imagem irá destacar as áreas selecionadas através das seeds f_g.

- Arquivos em Python

Para que a simulação dos testes fosse feita de maneira menos tediosa, existem arquivos em python que ajudam a realizar os testes:

- runSEQ_SEQ.py, para rodar a versão sequencial do SSSP
- installCuda.py, para instalar dependências
- runBorderFile.py, para gerar a imagem com bordas
- runGPU_SEG.py, para rodar a versão em GPU do SSSP

NOTA: Mais informações de como realizar os testes e usar seus resultados para interpretação podem ser encontradas em README, no link:

https://github.com/GabrielValeRios/superComp2018/tree/master/projeto4

Simulação e Comparação

Para a simulação e análise de resultados, foram escolhidas algumas imagens, que se encontram no repositório citado pelo link do Github.

Ao fim de cada simulação do arquivo Main.cpp, são gerados tempos no terminal, que podem ser usados para comparação. Os tempos são:

1. Para o código Main.cpp sequencial:
    - SSSP TIME, tempo de execução do algoritmo de Dijkstra
    - BuildImage Time, tempo para gerar a nova imagem

2. Para o código Main.cpp em GPU:
    - buildVecTime, tempo para montagem de vetores necessários ao código de SSSP da Nvidia
    - SSSP TIME x2. Serão gerados 2 tempos de SSSP, para cálculo das seeds f_g e b_g. Deve-se somar os dois tempos para obter o tempo de processamento total desta atividade.
    - buildImageTime, tempo para gerar a nova imagem

Para as comparações de desempenho, buildVecTime não foi utilizado, pois na versão sequencial não há necessidade de se construir vetores de pré-processamento. Portanto serão feitas comparações de tempo entre tempos de SSSP e de montagem da imagem resultante.

Os testes foram baseados para duas diferentes imagens:

- bolas.pgm
- ski.pgm

Para cada imagem, foram testados diferentes números de seeds f_g e b_g. As configurações entre os dois tipos de seeds foram:

- (f_G = 1, b_g = 1)
- (f_G = 2, b_g = 1)
- (f_g = 2, b_g = 2)
- (f_g = 3, b_g = 3)

As imagens utilizadas em GPU passaram antes pelo algoritmo de filtro de bordas. Assim, as imagens utilizadas no cálculo de SSSP já são aquelas cujo filtro está incluso, porém nada impede que as imagens originais fossem introduzidas diretamente nos algoritmos de cálculo de SSSP.

Assim, obteve-se os resultados:

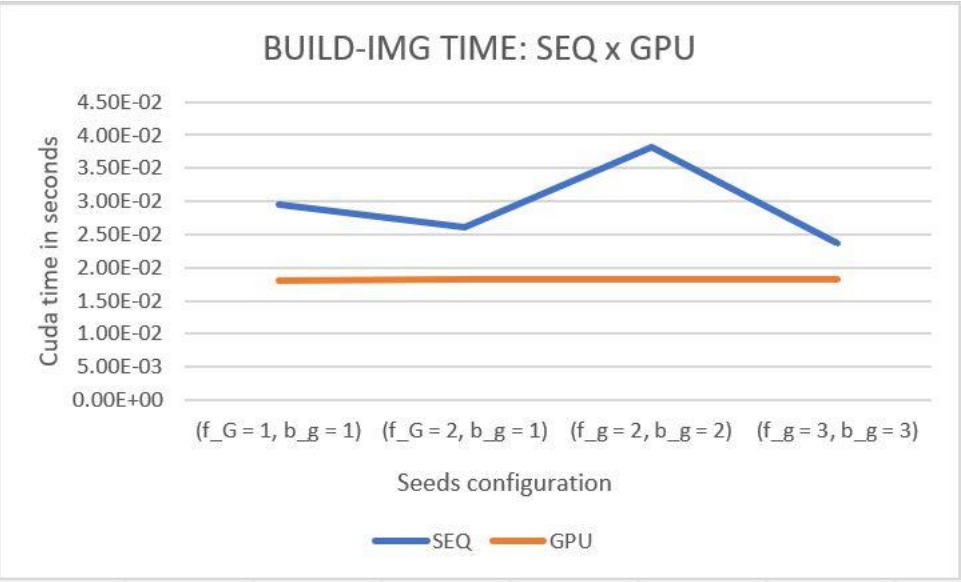

Figura 1 Resultados para bolas.pgm

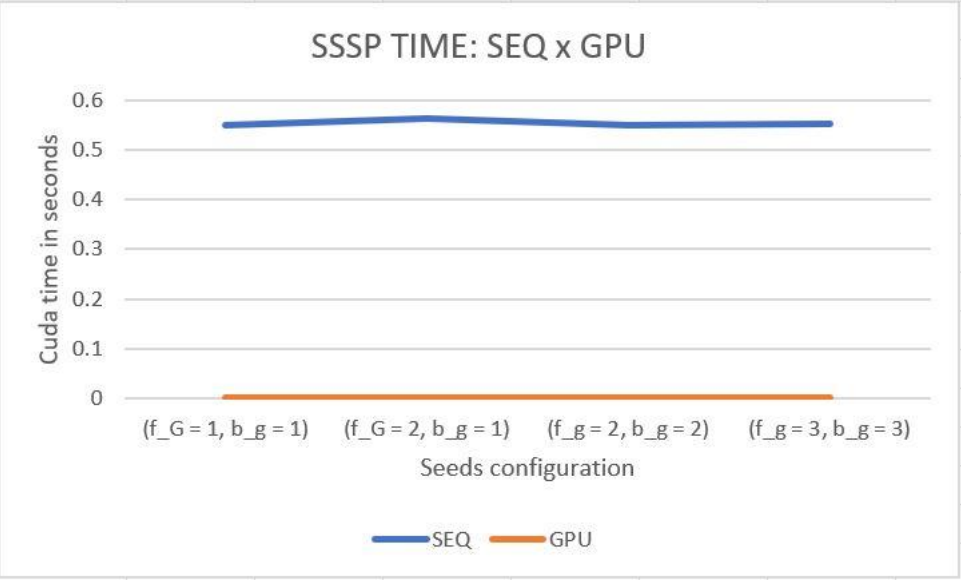

Figura 2 Resultados para bolas.pgm

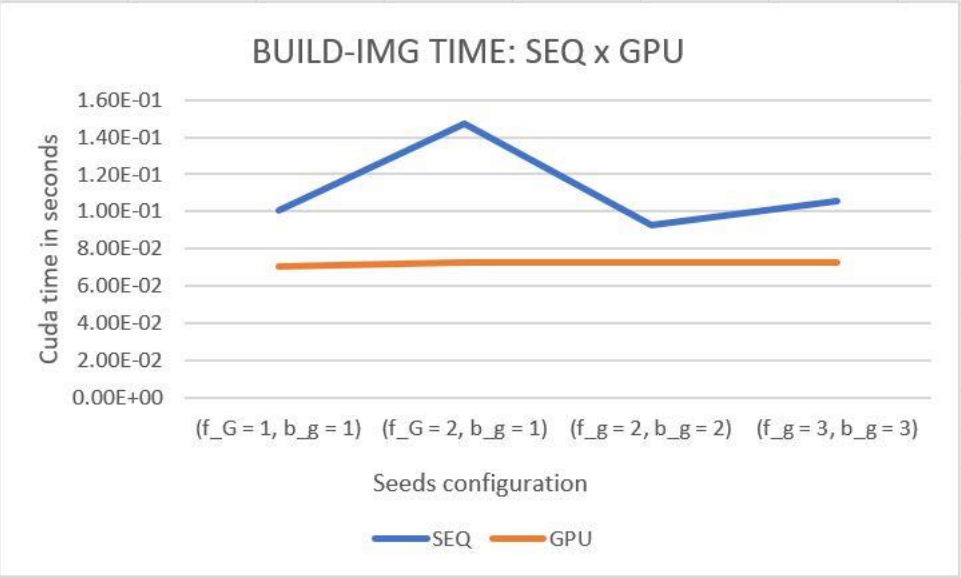

Figura 3 Resultados para ski.pgm

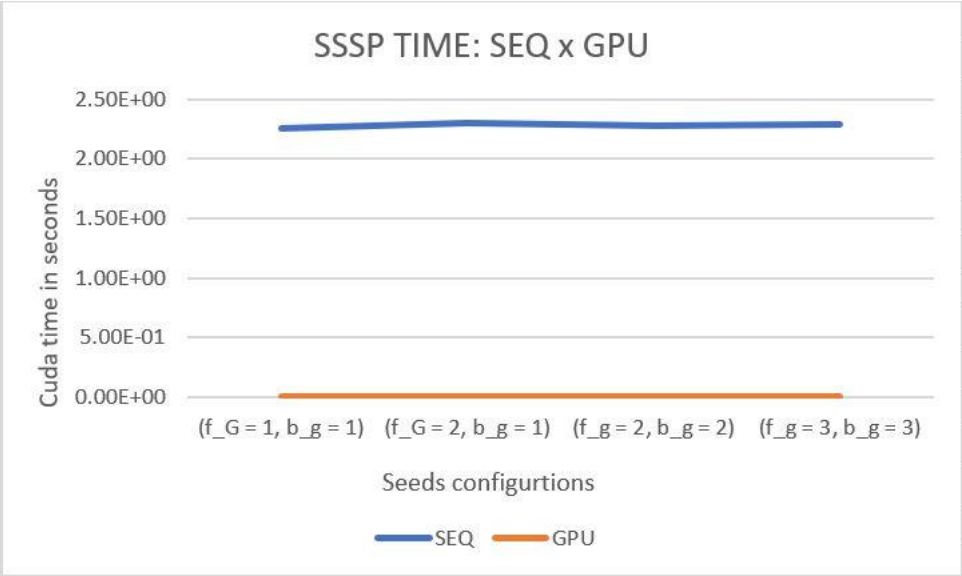

Figura 4 Resultados para ski.pgm

Análise e Conclusão

Pode-se observar nos gráficos gerados que existe um ganho notável de tempo ao se processar dados em GPU. Para o cálculo de SSSP, este ganho fica mais evidente ainda, com ganhos de desempenho de até 20 vezes (**Figura 4**). Para os gráficos **Figura 1** e **Figura 3**, as imagens não foram geradas em GPU, porém nota-se que nos algoritmos onde a GPU foi utilizada apresentaram um tempo um pouco menor do que o algoritmo sequencial.

Conclui-se que, para situações onde é necessário o processamento em paralelo de diversos itens independentes entre si, a GPU apresenta enorme vantagem sobre o processamento sequencial. Para segmentação de imagens, é muito comum se utilizar da GPU justamente por este ganho no processamento.