MC970/MO644 Programação Paralela: Paralelização da Transformada de Fourier em GPU

Deângeli Gomes Neves 188954
Klairton de Lima Brito 188948

Junho de 2016

1 Descrição do problema

A transformada de Fourier é um importante operador matemático que pode ser utilizado para realizar a interpretação de um sinal, a partir das componentes senoidais geradas por ela. O algoritmo que descreve a transformada de Fourier para sinais discretos (**DFT**-*Discrete Fourier Transform*) tem complexidade $\theta(n^2)$. Entretanto, utilizando algumas propriedades matemáticas dessa transformada é possivel projetar um algoritmo que computa a DFT em $\theta(n \log n)$, a **FFT** (*Fast Fourier Transform*), onde n tem que ser um multiplo da potência de 2. O conceito da transformada de Fourier pode ser estendido para sinais bidimensionais discretos, como por exemplo imagens digitais. A figura 1 mostra uma imagem (esquerda) e seu espectro (direita) gerado pela transformada de Fourier.

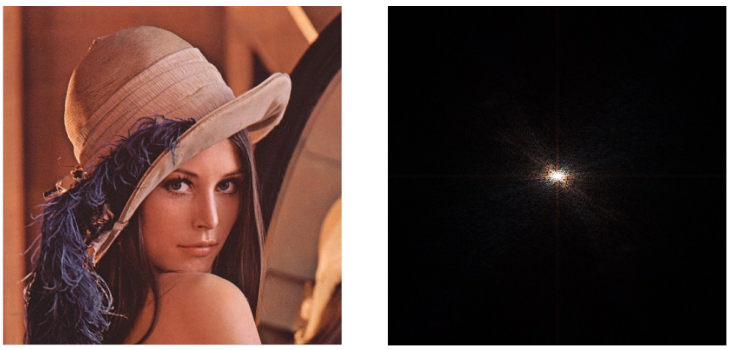

Figura 1: Imagem de entrada (esquerda) para o algoritmo da **FFT**. Espectro (direita) gerado a partir da transformado de Fourier

2 Paralelizando o problema

A transformada da Fourier aplicada em imagens pode ser dividida em duas etapas. Na primeira etapa é computada para cada linha a transformada 1D, na etapa seguinte o mesmo processo é aplicado para cada coluna. A ordem de computar linhas ou colunas é indiferente para o algoritmo, a unica restrição é que em cada etapa só é computada linhas ou colunas. Além disso, uma célula qualquer da matriz só depepende de dados da sua linha e da sua coluna, mas em etapas distintas.
Utilizando o **gprof** podemos notar que existe um grande "gargalo de computação" no algoritmo serial da **FFT**. O processo de computar a **FFT** para cada linha e coluna da imagem está sendo o processamento mais pesado do algoritmo, como pode ser visto na tabela 1.

% time	cumulative seconds	seconds	calls	self Ts/calls	Total Ts/calls	name
73.85	0.96	0.96				fft_by_col_CPU(float*, float*, int, int)
14.62	1.15	0.19				fft_by_row_CPU(float*, float*, int, int)
7.69	1.25	0.10				bit_reverse_CPU(unsigned int, unsigned int)
2.31	1.28	0.03				image_2_gray_image(t_image*)
0.77	1.29	0.01				gray_image_2_complex_image(t_gray_image*)
0.77	1.30	0.01				alloc_complex_image(int, int)

Tabela 1: Profile gprof

Ainda analisando os dados fornecidos pelo **gprof**, percebe-se que o tempo para computar todas as colunas é significativamente maior se comparado ao tempo para computar todas as linhas. Esse fato é decorrente do cálculo de índices dos elementos na vertical serem mais complexos de que o cálculo de índices dos elementos na horizontal.
Portanto, com base nessa análise e nas informações sobre o algoritmo, foi decicido que à abordagem utilizada para paralelização, em **GPU**, do algoritmo seria feita em cada uma das etapas (anteriormente mencionado) separadamente. A figura 2 ilustra de forma resumida como foi realizada a paralelização da **FFT**, em **CUDA**.

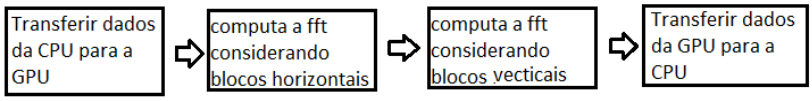

Figura 2: Fluxograma da paralelização.

Na primeira etapa foi definido, na **GPU**, que os dados seriam devididos em *chuncks* (pedaços) proporcionais ao número de colunas, como mostrado na figura 3. Na segunda etapa os *chuncks* são proporcionais ao número de linhas, ilustrado na figura 4. Como existe um grande reúso de dados em cada *chunk*, utilizou-se a shared memory visando obter um melhor desempenho.

A FFT dentro de cada bloco é dividida em estágios. Uma thread só pode avançar para outro estágio se todas as outras threads tiverem terminado aquele

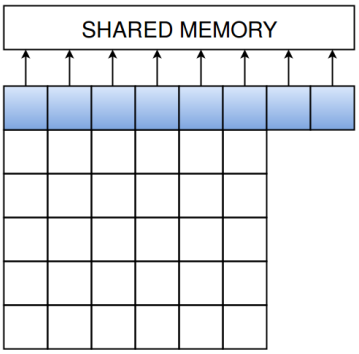

Figura 3: Bloco por linhas.

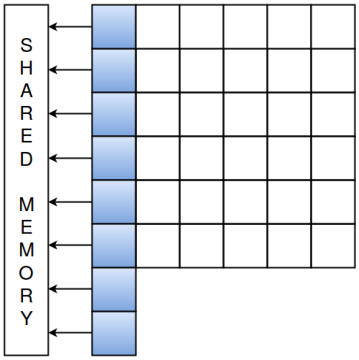

Figura 4: Blocos por colunas.

mesmo estágio. A figura 5 ilustra os estagios e a computação requerida para computar a **FFT** de 8 pontos. O *pipeline* que descreve a computação em cada *chunck* é ilustrado na figura 6.

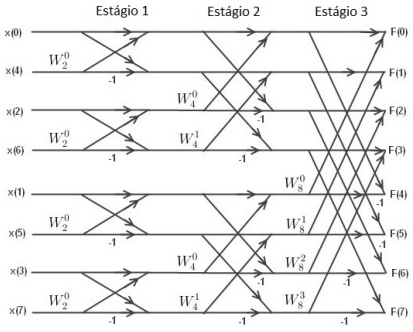

Figura 5: Exemplo de **FFT** 1D para 8 pontos.

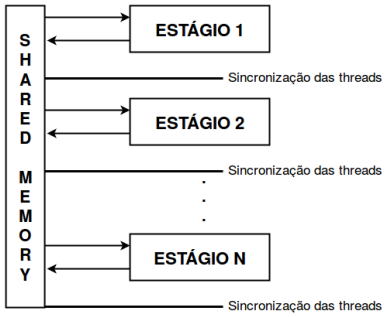

Figura 6: Pipelina da computação de cada *chunck*.

3 Limitações e Dificuldades

A abordagem proposta possue uma limitação com relação as dimensões da imagem de entrada. Isso ocorre porque a GPU fornece no máximo 1024 threads por chunk. Portanto a maxima dimensão permitida é 1024x1024. Para tentar contornar essa limiatação, pensou-se em dividir cada vetor linha e coluna, em mais de um chunck. Entretanto, este abordagem requer que exista troca de informação entre os chunks,o que "talvez" não exista no CUDA.
Diante desse limitação, fez uma breve revisão bibliografica sobre outras abordagens, disponiveis na internet, para a paralelização da **FFT**. Entretanto, as abordagens estudadas requeriam também a troca de dados entre blocos. A

cuFFT (biblioteca imeplementada pela a NVidia que fornece um conjunto de funções para computar a **FFT**) não possue essa limitação. Portanto, deve existir alguma abordagem para construir uma função escalonavel, em GPU, para calcular a **FFT** de um sinal;

4 Resultados

Os testes foram realizados utilizando imagens no formato ppm. Para verificar o desempenho da versão paralela com relação a versão serial, foi feito o cálculo de *speedup*. Os resultados podem ser vistos na tabela 2.

Arquivo	Dimensões	Tempo Serial	Tempo Paralelo	*speedup*
input01.ppm	256x256	32.66 ms	3.31 ms	9.85
input02.ppm	512x512	132.97 ms	12.61 ms	10.54
input03.ppm	1024x1024	1177.88 ms	47.66 ms	24.71

Tabela 2: Resultados

Para verificar a qualidade da abordagem utilizada para paralelizar do problema, foi feito um simples benchmarking entre à abordagem desenvolvida e a implementação da Nvidia para o problema (cuFFT). Os resultados podem ser visto na tabela 3.

Arquivo	Dimensões	cuFFT	Init + cuFFT	Abordagem desenvolvida
input01.ppm	256x256	1.114 ms	121.687 ms	2.298 ms
input02.ppm	512x512	1.784 ms	124.614 ms	8.429 ms
input03.ppm	1024x1024	3.984 ms	130.386 ms	33.202 ms

Tabela 3: Resultados da comparação com o cuFFT