https://eventos.utfpr.edu.br//sicite/sicite2020

Aplicação de técnicas de óptica adaptativa em GPU

Application of adaptive optics techniques on GPU

Gustavo Valente Gulmine
ggulmine@alunos.utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Curitiba, Paraná, Brasil

Alexandre José Tuoto Silveira Mello
ajmello@professores.utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Curitiba, Paraná, Brasil

RESUMO

O uso de instrumentos ópticos tem possibilitado diversos avanços para a ciência. Porém, quando se trata de observações astronômicas, muitas distorções podem ocorrer. Para resolver isso, uma das técnicas utilizadas é a chamada Óptica Adaptativa, que tradicionalmente sofre com a necessidade elevada de processamento. Nos últimos anos, várias técnicas alternativas vêm sendo propostas com o intuito de reduzir esse custo computacional. A proposta deste trabalho é desenvolver uma nova alternativa aos métodos tradicionais de óptica adaptativa, utilizando GPU para aceleração de *hardware* em uma variante de *software* de baixo custo computacional. Para receber a alteração, foi selecionada a função *shift*, responsável por deslocar os valores em uma imagem. A implementação foi feita com o uso de uma GPU NVIDIA, aliada ao uso do *Toolkit CUDA*, da biblioteca OpenCV para C++ e do *software* MATLAB. Os resultados são promissores, apontando para uma melhora significativa no desempenho da função, mas que ainda se mostra mais lenta do que na versão original por conta de excessivas cópias de memória.

PALAVRAS-CHAVE: Programação paralela. Óptica Atmosférica. C++.

Recebido: 19 ago. 2020.
Aprovado: 01 out. 2020.

ABSTRACT

The use of optical instruments allowed science to achieve a lot of advancements. Despite that, in regard of astronomical observations, a lot of distortions may occur. To solve that, techniques known as Adaptive Optics are used, but they traditionally demand a high processing power. In the past few years, a lot of alternative techniques have been proposed to reduce this computacional cost. This work proposes to develop a new alternative by using a GPU hardware acceleration in a low computational cost software variant. The shift function, responsible for shifting the values in an image, was selected to be altered. For the modification to be implemented, a NVIDIA GPU was used, alongside the CUDA Toolkit, the OpenCV library for C++ and MATLAB. The results are promising, pointing out a significant increase in the function performance, although it is still slower than the original version due to an excessive use of memory copies.

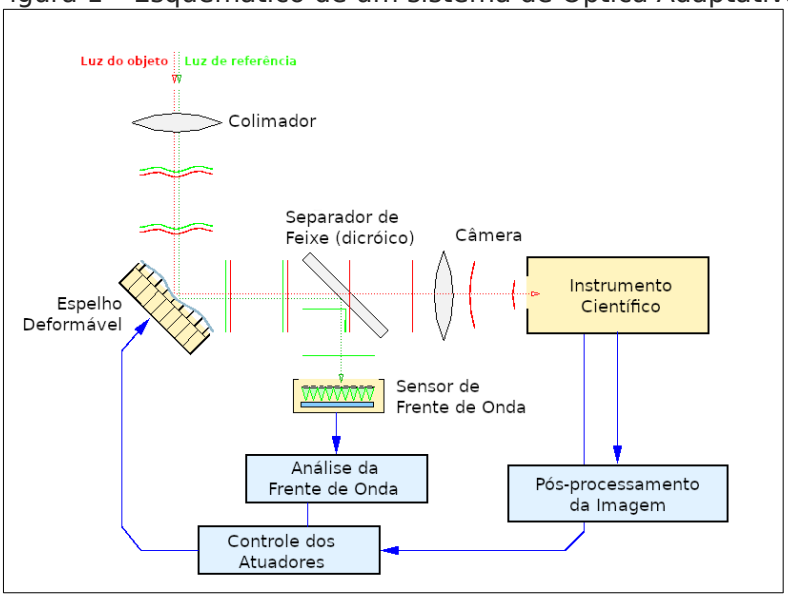
KEYWORDS: *Parallel programming. Atmospheric optics. C++.*

INTRODUÇÃO

O uso de instrumentos ópticos que permitem ultrapassar os limites da fisiologia humana sempre foi de muita importância para a ciência. Porém, um problema recorrente, em especial no uso de telescópios, é a ocorrência de distorções. No estudo de corpos celestes, essas distorções ocorrem majoritariamente pelas diferentes composições e densidades da atmosfera terreste. A fim de corrigir isso, diversas soluções vem surgindo nas últimas décadas, compondo um conjunto de técnicas chamadas de Óptica adaptativa.

Sistemas que usam de Óptica adaptativa são tradicionalmente compostos de um ou mais sensores de frente de onda (SFO), responsáveis por capturar a forma da onda a ser processada e reconstruída. Os dados obtidos à partir dessa reconstrução são então utilizados para regular os atuadores de um espelho deformável, que realiza a correção na imagem a ser captada por uma câmera, tal como exemplificado na Figura 1.

Figura 1 – Esquemático de um sistema de Óptica Adaptativa



Fonte: http://cfao.ucolick.org/images/aoscheme.gif, acesso em: 15/05/2020. Tradução: Autoria própria (2020).

O processamento dos dados obtidos é feito com o intuito de gerar um mapeamento tri-dimensional da atmosfera. Isso é feito em duas etapas: A reconstrução da frente de onda óptica e uma tomografia para a separação da turbulência em camadas (1). O método mais tradicional para a realização desse processamento é o de multiplicação vetor-matriz (MVM)(2), que engloba ambas as etapas de uma única vez. Porém, esse método possui um custo computacional muito elevado para funcionar em tempo real.

Alternativas vem sendo propostas em ambas as etapas do processamento, como os trabalhos de Ellerbroek (3) e Rosensteiner (4) na reconstrução da frente de onda e algoritmos como o Kaczmarz (5) no tratamento da turbulência. Somado a isso, existem várias soluções que propõe o uso de um *hardware* mais potente, como o uso de unidades de processamento gráfico (*graphics processing unit, GPU*)(6) ou processadores mais eficazes (7).

Uma nova alternativa para o tratamento da turbulência é o algoritmo desenvolvido por Mello e Pipa (1), o qual se diferencia por tratar os dados com o uso de técnicas de processamento de imagens, substituindo o uso das custosas multiplicações matriciais.

Tomando como base a implementação desse algoritmo, a proposta desse trabalho é realizar uma aceleração de *hardware* no algoritmo por meio do uso de uma GPU NVIDIA, a fim de melhorar seu desempenho. Essa aceleração se dará com a seleção de uma função do programa que ofereça o devido suporte à mudança proposta, e posterior implementação da alteração.

MATERIAIS E MÉTODOS

Nessa seção serão tratados os tópicos abrangendo os materiais utilizados para o projeto, subdivididos entre componentes de *hardware* e *software*, e a metodologia utilizada para implementação e teste da alteração proposta.

Os principais componentes de *hardware* estão descritos na tabela 1. Vale ressaltar que qualquer outro modelo de GPU NVIDIA pode ser utilizado, desde que possua suporte ao uso do *Toolkit CUDA*, porém, com uma potencial alteração no desempenho.

Tabela 1 - Componentes de *Hardware* utilizados.

Componente	Descrição
GPU	NVIDIA TitanXp
Processador	Intel i7-8700
Memória	DDR4 16GB 2667MHz
Placa-mãe	ASUS B360M-PLUS GAMING/BR

Fonte: Autoria própria (2020).

Quanto aos componentes de *software*, por este trabalho se tratar de uma adaptação de um código já existente, boa parte dos *softwares* utilizados foram herdados da implementação original.

Para organização geral do funcionamento do programa, foi utilizado o *software* MATLAB, versão R2017b (8). Nos scripts em MATLAB é feito o uso de funções compiladas em C++, que

utilizam da biblioteca OpenCV (9) para melhorar o desempenho geral do programa. Ambos os *softwares* citados foram herdados da implementação original.

A única adição foi o uso do *Toolkit CUDA (10)*, um conjunto de ferramentas de desenvolvimento disponibilizado pela NVIDIA para programação nas GPUs da empresa. O *Toolkit CUDA* vem com um compilador próprio, capaz de compilar também bibliotecas tradicionais em C++, tornando a adaptação proposta mais fácil, uma vez que é possível realizar a mudança sem provocar grandes alterações nas demais partes do código.

Visando alterar o mínimo possível do código original e trazer uma grande melhora no desempenho, foi escolhida a função *shift* para receber a alteração. Essa função foi selecionada principalmente por ser utilizada diversas vezes ao longo de cada execução do algoritmo, podendo impactar mais significativamente no resultado.

A função *shift* é responsável por deslocar os pixeis em uma “imagem” que contém os dados do SFO. Seu funcionamento se dá em duas etapas: um deslocamento inteiro e um parcial ou ”subpixel”. Por conta dessa estrutura, essa função é muito propícia para paralelização com o uso de uma GPU, que será implementada através da substituição das funções da biblioteca OpenCV que realizavam os deslocamentos por implementações manuais em CUDA.

A fim de averiguar se haverá uma melhora no tempo em relação a implementação original, serão feitos testes utilizando dois conjuntos de matrizes, X e Y, que representam a inclinação horizontal e vertical da turbulência atmosférica, tal como é a informação recebida de um SFO. As matrizes que compõe esses conjuntos possuem quatro tamanhos diferentes, sendo eles 41x41, 47x47, 89x89 e 101x101. Os tempos serão coletados utilizando a biblioteca *chrono* do C++, e se subdividem entre os tempos dos deslocamentos inteiro, parcial e total, além dos tempos de cópia de memória, no caso da nova implementação.

RESULTADOS E DISCUSSÃO

As tabelas 2a e 2b apresentam os resultados dos testes para a versão original do código. Como é perceptível nas tabelas, o tempo de execução do deslocamento parcial é muito mais elevado do que o deslocamento inteiro. Isso se enquadra dentro do esperado, uma vez que esse tipo de operação é extremamente complexa de ser realizada num processador. Além disso, há uma visível diferença entre os conjuntos, sendo o conjunto Y mais rápido para todos os tamanhos de matrizes testados.

Diferentemente dos testes no processador, os resultados obtidos com os testes na GPU demonstraram que o deslocamento parcial acabou sendo mais rápido que o deslocamento inteiro, tal como apresentado nas tabelas 3a e 3b. Além disso, é possível

Tabela 2 - Tempos relativos aos deslocamentos inteiro, parcial e total na execução no processador.

(a) Conjunto de matrizes X.

Tamanho da Matriz	Deslocamento Inteiro (ms)	Deslocamento Parcial (ms)	Total (ms)
41x41	4	78	82
47x47	4	82	86
89x89	19	217	236
101x101	22	228	250

(b) Conjunto de matrizes Y.

Tamanho da Matriz	Deslocamento Inteiro (ms)	Deslocamento Parcial (ms)	Total (ms)
41x41	3	67	70
47x47	3	79	82
89x89	12	140	152
101x101	15	121	137

Fonte: Autoria própria (2020).

verificar que os tempos de execução dos deslocamentos foram estáveis para os quatro tamanhos testados. Apesar de contra-intuitivo, isso é coerente com o que era esperado, já que são criadas threads para cada "pixel" da matriz e, dessa forma, enquanto o hardware suportar a criação de todas as threads necessárias de uma única vez, o tempo se manterá aproximadamente constante. O que não se mantém constante são os tempos de cópia, os maiores responsáveis pelo alto tempo total de execução da nova versão do código.

Tabela 3 - Tempos relativos aos deslocamentos inteiro, parcial e total e às cópias de memória na execução em GPU.

(a) Conjunto de matrizes X.

Tamanho da Matriz	Cópia Sistema p/ GPU (ms)	Deslocamento Inteiro (ms)	Deslocamento Parcial (ms)	Cópia GPU p/ Sistema (ms)	Total (ms)
41x41	70	10	5	99	183
47x47	62	10	4	82	158
89x89	95	10	4	130	239
101x101	88	9	4	148	249

(b) Conjunto de matrizes Y.

Tamanho da Matriz	Cópia Sistema p/ GPU (ms)	Deslocamento Inteiro (ms)	Deslocamento Parcial (ms)	Cópia GPU p/ Sistema (ms)	Total (ms)
41x41	72	9	4	95	180

47x47	73	11	5	95	184
89x89	84	9	4	116	213
101x101	78	9	4	153	244

Fonte: Autoria própria (2020).

A fim de visualizar melhor a diferença entre os tempos nas duas versões, foi elaborada a tabela 4. Nela, constam as diferenças de tempo total, obtidas subtraindo o tempo total da execução no processador do tempo total da execução com a GPU, para ambos os conjuntos de matrizes, além da média dos conjuntos. É possível perceber uma certa tendência de que o tempo de execução no processador se aproxime do tempo de execução na GPU e, possivelmente, o ultrapasse.

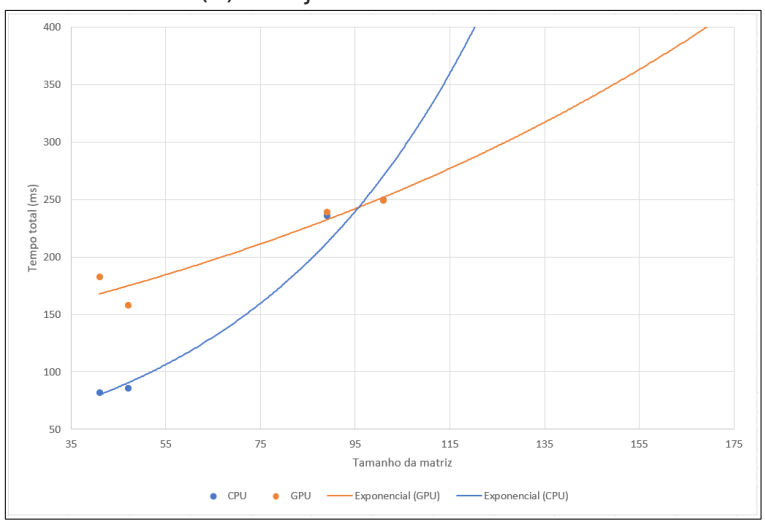
Tabela 4 - Diferenças de tempo total entre as execuções em GPU e no processador.

Tamanho da Matriz	Diferenças de tempo matrizes X (ms)	Diferenças de tempo matrizes Y (ms)	Diferenças de tempo média (ms)
41x41	102	109	106
47x47	72	102	87
89x89	3	61	32
101x101	-1	107	53

Fonte: Autoria própria (2020).

Figura 2 - Curvas de tendência de tempo total da função *shift*.

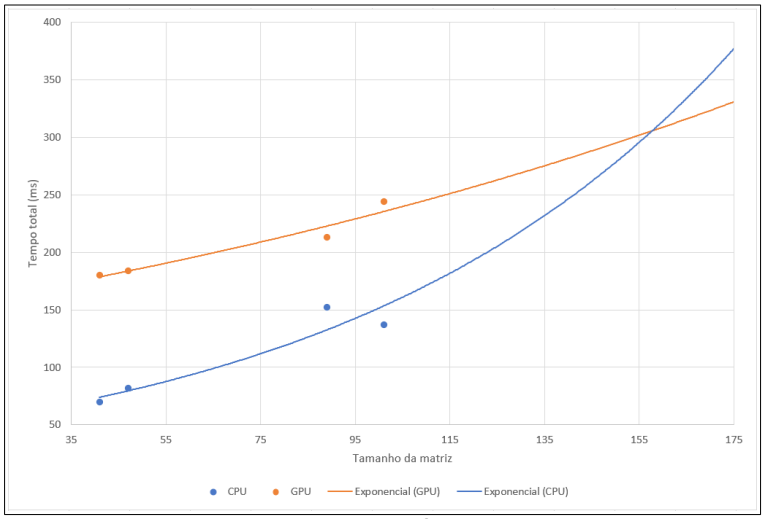
(a) Conjunto de matrizes X.



A fim de verificar isso, foram elaborados os gráficos presentes nas figuras 2a e 2b, utilizando uma linha de tendência exponencial. O tamanho das matrizes nos gráficos foi atrelado apenas ao tamanho de um dos lados, o que é possível por se tratarem de matrizes quadradas. Conforme os graficos

demonstram, apesar da quantidade extremamente pequena de pontos para traçar uma curva de tendência, há, de fato, a tendência de que os tempos de execução na GPU sejam, em dado ponto, menores que na CPU, como já aconteceu nas matrizes de tamanho 101x101 do conjunto X.

(b) Conjunto de matrizes Y.



Fonte: Autoria própria (2020).

CONCLUSÃO

Com a implementação e teste da nova versão do código, foi possível realizar a aceleração de *hardware* esperada no algoritmo. Os resultados, apesar de promissores, não foram os ideais, pois apesar de o tempo de execução dos deslocamentos da função shift terem sido muito melhores, o tempo total de execução dessa função foi pior do que na implementação original. Isso ocorreu por conta das várias cópias de memória necessárias para que a nova versão estivesse de acordo com as demais partes do código.

Isso nos permite delinear um caminho muito claro de prosseguimento do projeto, que viria com uma continuação da implementação, com o intuito de reduzir a necessidade de tantas cópias de memória.

AGRADECIMENTOS

Deixo aqui o agradecimento ao apoio do Instituto Serrapilheira (número de concessão Serra-1709-18844), da NVIDIA Corporation e da UTFPR. Deixo também o agradecimento aos Professores Doutores Alexandre José Tuoto Silveira Mello e Daniel Rodrigues Pipa por cederem o algoritmo por eles desenvolvido para a realização do projeto.

REFERÊNCIAS

(1) MELLO, A. J. T. S.; PIPA, D. R. **Matrix-free tomographic reconstruction for atmospheric turbulence.** Monthly Notices of the Royal Astronomical Society, v. 488, p. 395–400, set. 2019. DOI: 10.1093/mnras/stz1782.

(2) FUSCO, T. et al. **Optimal wave-front reconstruction strategies for multiconjugate adaptive optics.** J. Opt. Soc. Am. A, v. 18, n. 10, p. 2527–2538, out. 2001. Disponível em: https://doi.org/10.1364/JOSAA.18.002527. Acesso em: 19 jul. 2020.

(3) ELLERBROEK, B. L. **Efficient computation of minimum-variance wave-front reconstructors with sparse matrix techniques.** J. Opt. Soc. Am. A, v. 19, n. 9, p. 1803–1816, set. 2002. Disponível em: https://doi.org/10.1364/JOSAA.19.001803. Acesso em: 19 jul. 2020.

(4) ROSENSTEINER, M. **Wavefront reconstruction for extremely large telescopes via CuRe with domain decomposition.** J. Opt. Soc. Am. A, v. 29, n. 11, p. 2328–2336, nov. 2012. Disponível em: https://doi.org/10.1364/JOSAA.29.002328. Acesso em: 19 jul. 2020.

(5) ROSENSTEINER, M.; RAMLAU, R. **Kaczmarz algorithm for multiconjugated adaptive optics with laser guide stars.** J. Opt. Soc. Am. A, v. 30, n. 8, p. 1680–1686, ago. 2013. Disponível em: https://doi.org/10.1364/JOSAA.30.001680. Acesso em: 19 jul. 2020.

(6) BITENC, U. et al. **Suitability of GPUs for real-time control of large astronomical adaptive optics instruments.** J Real-Time Image Proc., v. 14, p. 743–751, abr. 2018. Disponível em: https://doi.org/10.1007/s11554-017-0702-7. Acesso em 04 abr. 2020.

(7) BARR, D. et al. **Reducing adaptive optics latency using Xeon Phi many-core processors.** Monthly Notices of the Royal Astronomical Society, v. 453, p. 3222–3233, 2015. DOI: 10.1093/mnras/stv1813.

(8) THE MATHWORKS, INC. MATLAB version 9.3.0.713579 (R2017b). Natick, Massachusetts, 2017.

(9) BRADSKI, G. The OpenCV Library. Dr. Dobb's Journal of Software Tools, 2000.

(10) NVIDIA, INC. CUDA Toolkit v10.1. [S.l.], 2019. Disponível em: https://docs.nvidia.com/cuda/archive/10.1/.