Implementação Paralela em GPU para Geração de Fractais de Corais

Matheus Tregnago[1], **Samuel Francisco Ferrigo**[1], **André Luís Martinotto**[1]

[1]Campus Universitário da Região dos Vinhedos
Universidade de Caxias do Sul (UCS) – Bento Gonçalves – RS – Brasil

{matregnago,sfferrig,almartin}@ucs.br

Resumo. *Este artigo descreve uma implementação paralela desenvolvida em GPU para a geração de fractais de corais utilizando o algoritmo Agregação Limitada por Difusão (DLA). A implementação foi desenvolvida na linguagem de programação C++ e paralelizada através da plataforma CUDA. Atingiu-se um speedup de* 40 *vezes, porém, observou-se uma estabilização do speedup, devido ao limite máximo de threads ativas simultaneamente na GPU.*

1. Introdução

Os fractais são objetos que se caracterizam por manter a sua forma invariável, independentemente da escala de observação. A geometria fractal é aplicada no estudo de estruturas complexas e na simulação de fenômenos naturais, como a formação de corais [Assis 2008]. Contudo, a geração dessas estruturas possui um elevado custo computacional, uma vez que resulta de processos iterativos que exigem grande poder de processamento para formar modelos de elevada complexidade. Neste contexto, a programação paralela surge como uma alternativa para acelerar a geração de fractais. Entre as tecnologias disponíveis, as unidades de processamento gráfico (GPU, do inglês *Graphics Processing Unit*) destacam-se pela sua eficiência ao dividir as instruções de processamento entre múltiplos núcleos, permitindo a execução paralela de tarefas [Kirk and Hwu 2010].

Este trabalho apresenta o desenvolvimento de uma aplicação paralela em GPU, utilizando a plataforma CUDA (*Compute Unified Device Architecture*), com o objetivo de gerar fractais de corais tridimensionais por meio do algoritmo de Agregação Limitada por Difusão (DLA, do inglês *Diffusion-limited aggregation*) e avaliar o ganho de desempenho obtido com a paralelização.

2. Geração de Fractais de Corais

O termo fractal é utilizado para definir objetos que apresentam a mesma estrutura em diferentes escalas. Esses objetos são formados por padrões que se repetem em versões menores, preservando uma semelhança com a estrutura original. Entre os algoritmos utilizados para a geração de fractais, destacam-se os modelos baseados em processos estocásticos, que simulam fenômenos naturais de forma probabilística [Fuzzo 2009].

O algoritmo frequentemente utilizado para a geração de corais consiste na DLA, um modelo estocástico empregado para simular processos de agregação de partículas. Nesse modelo, partículas executam movimento aleatório até entrarem em contato com um aglomerado já existente, ao qual passam a aderir, contribuindo progressivamente para o crescimento e a complexidade da estrutura resultante [Witten and Sander 1981].

A Figura 1 apresenta o fluxograma do algoritmo desenvolvido neste trabalho. A versão sequencial foi implementada na linguagem de programação C++ e consiste na simulação do crescimento do coral em uma grade tridimensional, inicialmente configurada com uma semente no centro do domínio. A cada iteração, uma nova partícula é inserida em posição aleatória e realiza uma caminhada até se agregar à estrutura existente ou ser descartada ao exceder os limites definidos. As partículas agregadas têm suas coordenadas armazenadas e suas vizinhanças marcadas como possíveis pontos de crescimento. O algoritmo é executado até que um número desejado de partículas seja agregado.

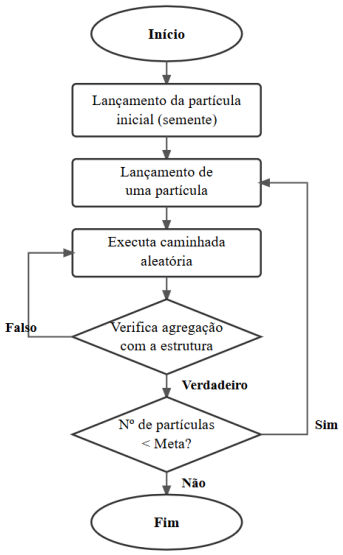

Figura 1. Fluxograma da implementação do algoritmo DLA

Após a conclusão da simulação, a estrutura gerada é exportada para um arquivo no formato *Wavefront*[1], que pode ser importado em um software de modelagem e visualização 3D, como o Blender[2]. Dessa forma, torna-se possível a renderização, a aplicação de texturas e a realização de ajustes finais. A Figura 2 ilustra o processo de exportação e apresenta um exemplo de um modelo gerado e visualizado no Blender.

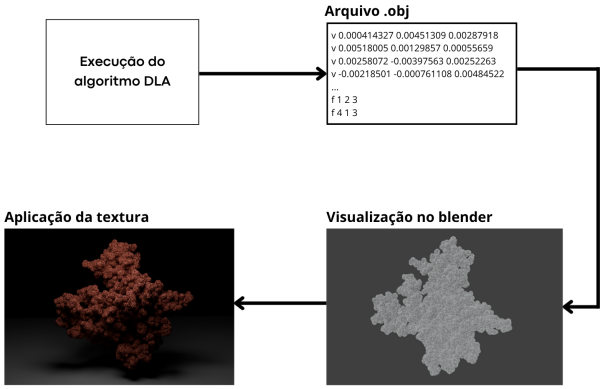

Figura 2. Visualização do Fractal de Coral

[1]https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml
[2]https://www.blender.org/

A implementação paralela[3] foi desenvolvida utilizando a plataforma CUDA [NVIDIA 2025], pela abordagem descrita no trabalho desenvolvido por Zsaki (2016). Nesta, a execução é organizada em blocos de *threads*, onde cada *thread* é responsável por simular a caminhada aleatória de uma partícula de forma independente. Quando ocorre a agregação, a estrutura compartilhada é atualizada por meio de operações atômicas, garantindo a consistência e a marcação de novas posições potenciais de crescimento. Ao final da execução, as coordenadas das partículas são transferidas da memória da GPU para a memória da máquina hospedeira.

3. Testes e Resultados Obtidos

Os testes foram realizados em um computador com processador Intel Core i5-14600K (14 núcleos de 3,5 GHz) e 32 GB de memória RAM DDR5. A GPU utilizada foi uma NVIDIA RTX 4060 Ti (arquitetura Ada Lovelace), que possui 4.352 núcleos CUDA e 8 GB de memória global. As simulações foram executadas em uma grade tridimensional de $700 \times 700 \times 700$. Foram realizados testes variando de 10.000 a 200.000 partículas, com incrementos de 10.000. Cada teste foi executado 5 vezes, e foi utilizada a média aritmética dos tempos.

A Figura 3 apresenta um comparativo do tempo de execução das versões sequencial e paralela em função do número de partículas. Na implementação sequencial, o tempo de execução cresce com o aumento do número de partículas, embora a taxa de crescimento tenda a diminuir à medida que a estrutura do coral se expande, pois uma estrutura maior oferece mais pontos de contato, reduzindo o percurso médio da caminhada aleatória das partículas. A implementação paralela apresentou tempos de execução inferiores em todos os cenários avaliados, com ganhos à medida que o número de partículas aumenta, evidenciando a escalabilidade da solução e a capacidade da GPU de processar múltiplas caminhadas aleatórias de forma simultânea.

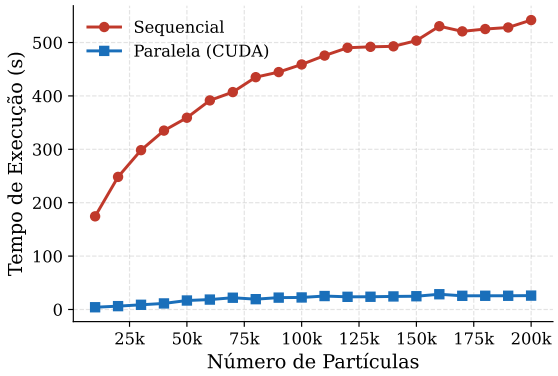

Figura 3. Tempos de execução em função do número de partículas

A Figura 4 apresenta o *speedup* obtido em função do número de partículas. Como pode ser observado, o *speedup* inicia em aproximadamente 40 vezes e sofre uma queda até estabilizar em torno de 20 vezes. Isso é justificado pelo aumento do número de partículas, que eleva a disputa por posições na grade, aumentando o custo das operações atômicas na memória global. Outro fator que influencia esse resultado é o limite do número *threads* da

[3]https://github.com/matregnago/dla-fractal-cuda

GPU utilizada, que suporta um máximo de 52.224 *threads* simultâneas. Acima de 50.000 partículas, a GPU atinge sua capacidade máxima e as partículas excedentes precisam aguardar a liberação de núcleos, impedindo ganhos adicionais de *speedup*.

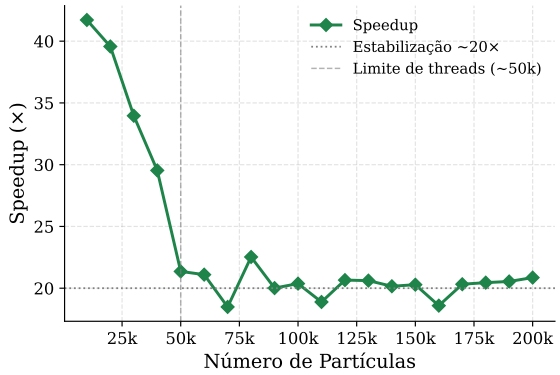

Figura 4. *Speedup* em função do número de partículas

4. Conclusão

Conclui-se que a paralelização do algoritmo DLA em GPU proporciona uma melhoria significativa no tempo de execução para a geração de fractais de corais. Observou-se ainda que o valor do *speedup* tende a sofrer uma redução seguida de estabilização a partir de 50.000 partículas. Esse comportamento é atribuído ao limite físico de *threads* simultâneas suportadas pela GPU RTX 4060 Ti e ao aumento da concorrência por acesso à memória global através de operações atômicas.

Como trabalhos futuros, sugere-se a paralelização de outros sistemas de crescimento orgânico, como Reaction-diffusion [Kopell and Howard 1973] e L-systems [Rozenberg and Salomaa 1980], utilizando a plataforma CUDA. Além disso, propõe-se avaliar o consumo energético da implementação, comparando a execução em CPU e em GPU.

Referências

Assis, T. A. (2008). *Geometria fractal: propriedades e características de fractais ideais*. Revista Brasileira de Ensino de Física.

Fuzzo, R. A. (2009). *Fractais: Algumas Características e Propriedades*. Encontro de Produção Científica e Tecnológica.

Kirk, D. B. and Hwu, W.-m. W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Kopell, N. and Howard, L. N. (1973). Plane Wave Solutions to Reaction-Diffusion Equations. *Studies in Applied Mathematics*, 52(4):291–328.

NVIDIA (2025). Cuda C++ programming guide. *NVIDIA Docs*.

Rozenberg, G. and Salomaa, A. (1980). *The Mathematical Theory of L Systems*. Publication No. 25 of the Mathematics Research Center, the U. Academic Press.

Witten, T. A. and Sander, L. M. (1981). Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon. *Phys. Rev. Lett.*, 47:1400–1403.