Construção Paralela Lock-Free de Octrees Esparsas em GPU

Michel B. Cordeiro[1], **Wagner M. Nunan Zola**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)

Resumo. *Octrees são estruturas de dados frequentemente utilizadas para representar e organizar dados tridimensionais de maneira eficiente. Este trabalho propõe um algoritmo paralelo lock-free em GPU para construção de octrees esparsas com encadeamento de nodos com pointers.*

1. Introdução

Octree é uma estrutura de dados em forma de árvore usada principalmente para representar e organizar dados tridimensionais. Algoritmos em GPU frequentemente trabalham simultaneamente com grandes quantidades de elementos para maximizar o uso do processamento massivo. A construção *octrees* em simulações de N-corpos, por exemplo, é feita a cada passo da simulação, construindo toda a árvore em paralelo. A construção paralela eficiente de *octrees* em GPUs foi demonstrada [Burtscher and Pingali 2011], onde são colocados *locks* e barreiras no laço de espera ocupada, para que as *threads* de um bloco não gerem uma enorme quantidade de acessos à memória. Essas técnicas foram também empregadas em outras aplicações [Meyer et al. 2022, Chan et al. 2019] em aprendizado de máquina, que utilizam a construção paralela de *octrees*. Este trabalho propõe um algoritmo paralelo *lock-free* para a construção de *octrees* em GPU, eliminando o uso de *locks*, esperas ocupadas e barreiras. O algoritmo proposto obteve melhor desempenho em relação ao método de [Burtscher and Pingali 2011] em diferentes distribuições de dados de entrada.

2. Algoritmo Proposto

A *octree* será utilizada para armazenar corpos no espaço em simulações Barnes-Hut, onde cada nó interno da árvore representa um octante e, cada nodo folha possui no máximo um corpo. Na versão paralela do algoritmo, cada *thread* recebe um corpo e percorre a árvore até encontrar o octante no qual o corpo deve ser incluído. Se o octante estiver livre, basta inserir o corpo. Caso contrário, o octante deve ser subdividido até que o corpo que estava lá e o corpo que está sendo inserido estejam em octantes diferentes. Sendo assim, existe uma condição de corrida quando várias *threads* tentam incluir no mesmo nó. Para resolver esse problema, o algoritmo proposto por [Burtscher and Pingali 2011] utiliza *locks* para impedir que mais de uma *thread* modifique o mesmo nó. Enquanto uma *thread* está construindo a sub-árvore que divide os octantes, as outras ficam aguardando. Este trabalho propõe o algoritmo OctreeBuild-LF que utiliza a operação atômica *Compare and Swap* (AtomicCAS) para garantir que uma *thread* não sobrescreva as modificações feitas por outra *thread*. Essa operação compara o valor lido com o valor armazenado no momento da inserção. Se a comparação falhar, aquela região da memória já foi alterada, e o nó que armazenava um corpo virou um nó interno. A *thread* então percorre o nó interno e tenta realizar a inserção novamente. Para aumentar a eficiência do algoritmo, é importante que as gravações sejam feitas o mais rápido possível. Por isso, as *threads* atualizam a árvore a cada subdivisão do *octante* e cada *thread* possui um nó pré-alocado, removendo a necessidade de alocar memória durante a subdivisão do octante.

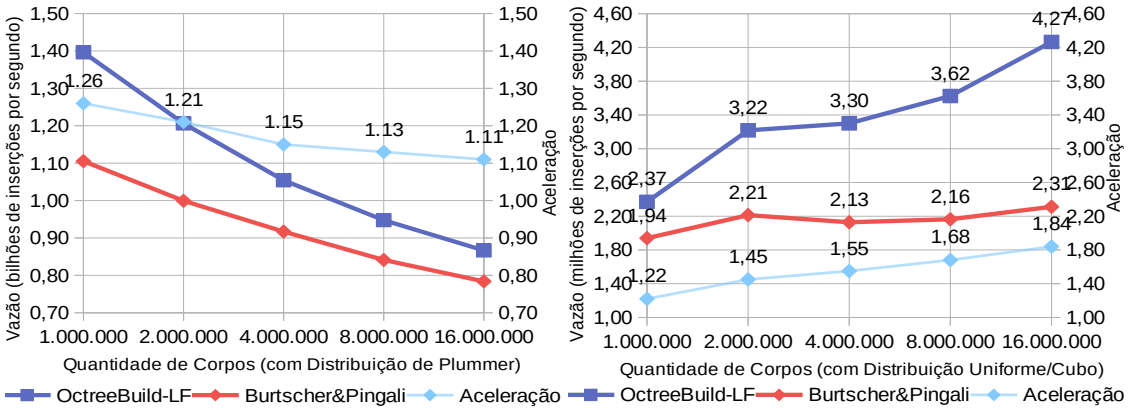

Figura 1. Resultado dos experimentos realizados.

3. Resultados e discussões

Para analisar a sua eficiência, o OctreeBuild-LF será comparado com o algoritmo desenvolvido por [Burtscher and Pingali 2011]. Dois cenários serão considerados: i) os corpos estão distribuídos seguindo a distribuição de Plummer, com os dados dispostos aleatoriamente na memória e densidade variável; ii) os corpos estão organizados em cubo uniforme no espaço, onde corpos próximos no espaço possuem dados próximos na memória. A distribuição ii é interessante, pois avalia cenários onde a entrada de dados apresenta maior localidade na distribuição e maior concorrência entre *threads*. Os testes foram executados em processador Intel Xeon Silver 4314 @ 2.40GHz e GPU NVIDIA A4500. A biblioteca CUDA versão 11.7 foi utilizada no sistema Linux Ubuntu 20.04.3 LTS. Os experimentos foram executados com entrada de 1, 2, 4, 8 e 16 milhões de corpos, e repetidos 30 vezes. A vazão média de inserções por segundo foi reportada, calculando-se os intervalos de confiança de 95%, mas não foram observados valores maiores que 0,5% em relação à média. Os resultados dos experimentos podem ser observados na Figura 1. Nota-se, primeiramente, que o OctreeBuild-LF apresenta maior vazão em ambos os cenários avaliados, alcançando uma aceleração de até 1,84 para 16 milhões de corpos na distribuição uniforme. Isso demonstra a eficiência do algoritmo proposto, principalmente quando há uma maior concorrência entre as *threads*. Também é possível observar que, quando os dados estão armazenados aleatoriamente na memória, a concorrência diminui conforme a quantidade de corpos aumenta, reduzindo a aceleração obtida pelo OctreeBuild-LF. Além disso, o aumento no tamanho da árvore também causa um aumento na quantidade de acessos irregulares à memória, diminuindo a vazão para ambos os algoritmos. Já na distribuição ii, a vazão aumenta com a quantidade de corpos, pois a localidade de acessos aos dados resulta em um acesso mais eficiente à memória. No entanto, é importante que outras distribuições sejam consideradas em trabalhos futuros.

Agradecimentos
Parcialmente suportado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processo 407644/2021-0.

Referências

Burtscher, M. and Pingali, K. (2011). An efficient CUDA implementation of the tree-based barnes hut n-body algorithm. In *GPU computing Gems Emerald edition*.

Chan, D. M., Rao, R., Huang, F., and Canny, J. F. (2019). GPU accelerated t-distributed stochastic neighbor embedding. *Journal Parallel and Distributed Computing*, vol 131.

Meyer, B. H., Pozo, A. T. R., and Nunan Zola, W. M. (2022). Global and local structure preserving GPU t-SNE methods for large-scale applications. *Expert Systems*, vol. 201.