Métodos para simulação Barnes-Hut distribuída com MPI

Rodrigo Morante Blanco[1], **Wagner M. Nunan Zola**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – Curitiba – PR – Brazil

rodrigomorante@gmail.com, wagner@inf.ufpr.br

Resumo. *O método Barnes-Hut é aplicado em diversas simulações científicas. Técnicas desenvolvidas para eficiência do mesmo em sistemas paralelos podem também ser aplicadas a outros algoritmos. Neste trabalho criamos quatro variantes para execução distribuída do método com distribuição de trabalho entre nodos utilizando MPI. Os resultados obtidos demonstram potencial para a aplicação desses métodos em execuções distribuídas Barnes-Hut.*

1. Introdução

A simulação de um sistema formado por N corpos sujeitos à ação de forças é um das classes principais de programas paralelos estudados classicamente, tanto como *benchmarks* de sistemas computacionais de alto desempenho como em diversas aplicações científicas da atualidade [Arora et al. 2009]. O cálculo das forças e posterior atualização das posições e velocidades dos N corpos pelo método direto (partícula-partícula) apresenta complexidade quadrática $\mathcal{O}(N^2)$, tornando inviável seu uso em larga escala. Barnes e Hut propuseram um algoritmo aproximado (*BH*) de cálculo de forças de complexidade $\mathcal{O}(N \log N)$ baseado no caminhamento em *octrees*. Novas técnicas foram desenvolvidas recentemente para otimização do cálculo de forças no método BH, tanto em processadores *multicore* [Delgado et al. 2019] quanto em *manycore* [Meyer et al. 2021]. O trabalho anterior é um exemplo recente da aplicação do algoritmo BH no contexto de aplicações que necessitam analisar grandes volumes de dados de alta dimensionalidade.

No presente trabalho apresentamos um estudo inicial de possíveis métodos a serem empregados em versões distribuídas do algoritmo BH, com potencial aplicação no contexto dos trabalhos anteriormente descritos. Uma versão do algoritmo de Barnes-Hut distribuída com MPI foi desenvolvida, inicialmente implementando uma *octree* esparsa (baseada em ponteiros). Os experimentos foram realizados em um único computador simulando um *cluster* com múltiplos processos MPI. Com respeito à *octree* duas variantes foram consideradas: ***octree* completa**, na qual cada nodo utiliza todos os corpos para criar a *octree* e ***octree* parcial**, na qual cada nodo utiliza uma parte dos corpos para criar a *octree*. Com respeito à carga de trabalho duas variantes foram consideradas: **carga estática**, na qual cada nodo calcula as forças exercidas sobre uma parte dos corpos fixada *a priori* e **carga dinâmica**, na qual cada nodo calcula as forças exercidas sobre uma parte dos corpos. No caso da *octree* completa há um nodo que envia trabalhos aos outros nodos na medida que estes terminam. No caso da *octree* parcial o tempo empregado por cada nodo em um passo da simulação é utilizado para distribuir o trabalho do próximo passo.

2. Resultados

Os experimentos foram realizados em um processador Intel i7-10700 (2.90 GHz) de 8 núcleos e 16 GB de RAM, com sistema operacional Ubuntu 20.04. Foram simulados

Tabela 1. Tempo de execução em segundos para 10^6 corpos (melhor em negrito).

Número de nodos MPI	1		2		4		8	
Octree completa, carga estática	32.34	(0.0)	16.71	(0.4)	10.44	(1.2)	7.38	(0.9)
Octree completa, carga dinâmica	**32.07**	(0.0)	16.38	(0.1)	**9.70**	(0.1)	**6.82**	(0.1)
Octree parcial, carga estática	34.03	(0.0)	16.23	(1.1)	15.22	(7.2)	13.04	(8)
Octree parcial, carga dinâmica	32.24	(0.0)	**14.58**	(0.3)	13.46	(5.8)	8.66	(4)
Melhor *speedup*	1.00		2.21		3.31		4.70	

clusters de 1, 2, 4 e 8 nodos (sendo 8 o número de núcleos da CPU). Foi simulado um passo da evolução de um sistema com 10^6 corpos e os tempos coletados. Na Tabela 1 pode-se ver que geralmente os melhores resultados são obtidos quando os nodos têm a *octree* completa e recebem trabalhos de forma dinâmica. A Figura 1 mostra o tempo dedicado a cada tarefa por cada um dos quatro métodos criados e simulando 8 nodos MPI. Os métodos com *octree* parcial dedicam uma parte considerável do tempo na comunicação entre os nodos (mostrados entre parênteses na tabela).

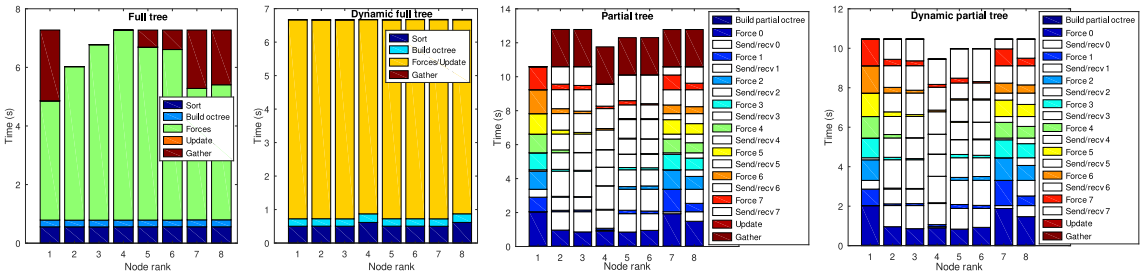

Figura 1. Tempo em segundos por nodo MPI, 10^6 corpos, 8 nodos MPI.

3. Conclusões e trabalhos futuros

O método de Barnes-Hut foi adaptado para ser distribuído utilizando MPI, inicialmente trabalhando sem emprego das estruturas de aceleração de nossos trabalhos anteriores. Os resultados foram obtidos em um único multiprocessador simulando um *cluster* e mostram que as estratégias empregadas são promissoras. O código deve ser testado em um *cluster* verdadeiro, onde o custo da transmissão de dados e impacto das latências podem ser mais significativos, especialmente para as variantes com *octrees* parciais. A otimização da comunicação MPI no caso das versões com *octrees* parciais deve ser mais estudada devido a sua importância na simulação de problemas em maior escala.

Referências

Arora, N., Shringarpure, A., and Vuduc, R. W. (2009). Direct N-body kernels for multicore platforms. In *ICPP 2009, International Conference on Parallel Processing, Vienna, Austria, 22-25 September 2009*, pages 379–387.

Delgado, A., Blanco, R. M., and Nunan Zola, W. (2019). Caminhamento paralelo Barnes-Hut com vetorização AVX2. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 454–461, Porto Alegre, RS, Brasil. SBC.

Meyer, B. H., Pozo, A. T. R., and Nunan Zola, W. M. (2021). Improving Barnes-Hut t-SNE algorithm in modern GPU architectures with random forest kNN and simulated wide-warp. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 17(4):1–26.