MPI Parallel Barnes-Hut variants

Rodrigo Morante Blanco*
rodrigomorante@gmail.com
Universidade Federal do Paraná
Curitiba, Paraná, Brazil

Wagner M. Nunan Zola†
wagner@inf.ufpr.br
Universidade Federal do Paraná
Curitiba, Paraná, Brazil

ABSTRACT

The n-body problem appears in numerous fields, from stellar simulations to, more recently, AI applications. The Barnes-Hut (BH) method is a tree-based method that allows better scalability when dealing with this problem. In this work several variants of the BH method are proposed which use MPI to distribute the workload.

KEYWORDS

n-body problem, Barnes-Hut method, Message Passing Interface

1 INTRODUCTION

The n-body problem appears in numerous fields, from stellar simulations to, more recently, AI applications [12] [15]. The Barnes-Hut (BH) method is a tree-based approximation which allows better scalability when dealing with this problem. Recent applications of BH in AI methods and in the visualisation of high dimensional data [12] [9] have determined an increase of interest to accelerate BH in modern multicore CPUs [5][6] and in distributed versions. In this work several variants of the Barnes-Hut method are proposed which use MPI to distribute the workload. We have analyzed our implementations in a modern multicore CPU. Future applications of our parallel/distributed methods could be combined with other state of the art techniques to further accelerate the Barnes-Hut method in multicore and CPU/GPU clusters. The classical n-body problem can be described thus: Let there be n celestial bodies, interacting gravitationally, whose corresponding masses, positions and velocities are known at a given initial time. The *n-body problem* refers to the computation of each body's position as time progresses. For each of the n bodies, $n-1$ forces must be computed for a total of $n(n-1)$ forces or $O(n^2)$ computations. In order to reduce the number of computations a tree-based method may be used. The rest of this work is organized as follows: Section 2 discusses the fundamentals necessary to understand the Barnes-Hut method. Related works are described in Section 3. Our MPI parallel approaches will be presented in Section 4. The results and discussions about the experiments will be presented in Section 5. Finally, the conclusions and future works are presented in Section 6 and Section 7, respectively.

2 THE BARNES-HUT METHOD

The *Barnes-Hut method* [2] makes use of an octree to partition the space according to the position of the bodies. Let l be the length of the (cubic) cell, d the distance between the object and the cell's barycenter and δ the distance between the cell's center and barycenter. Let also θ be the *opening angle parameter*. If $d \geq \frac{l}{\theta} + \delta$ then the object and the cell are considered to be sufficiently apart: in this case the barycenter of the objects within the cell is used to compute the total force on the object, otherwise the cell is traversed recursively and each of the cells contained within it is tested, eventually using the bodies within them to compute the force. A simpler version of the aforementioned equation can be used which does not take into consideration δ. In that case the opening criterion can be simplified as $d^2 \geq \frac{l^2}{\theta^2}$ which makes use of no square roots, thus driving down the computational cost. By taking $\theta = 0$ all the cells are open, thus obtaining the same results as in the quadratic version. The cost for a tree-based method is $O(n \log n)$.

3 RELATED WORKS

Efficient methods were developed in [4] to generate and traverse pointer-based octrees in GPUs in BH gravitational simulations. The performance of this code was analyzed by [14], including comparisons with codes written for multicore CPUs. Another simulator which is run in GPUs was presented by [3]. This code uses a sparse, pointer-based octree, but with a different structure. The trees are generated and traversed in pre-order. The bodies are ordered according to their Morton keys. This and the following stages where there exist data transfers associated to the reordering of the internal nodes uses approximately 66% of all the time devoted to the generation of the tree. A possible advantage of ordering the leaves according to their Morton order is that the efficiency of the cache is improved during the tree traversal.

The use of implicit octrees during the traversal phase in multicore CPUs was presented in [6].

The use of parallelism using SIMD/AVX2 vectorization was explored in several works which involve the n-body problem and tree-traversals [13], [8], [1], specially where there were direct computations of gravitational forces. Such is the case of [7] which uses the dual tree method and that does not use intrinsic SIMD instructions directly, leaving the work of vectorization to the compiler. This is possible mainly because in these methods the interactions are of the cell-cell (C-C) type, which are easier to deal with with automatic vectorization. C-C interactions are equivalent to applying the direct method between bodies which are contained in the cells and, at this point, becomes a regular computation pattern.

In this work we study several MPI parallel variants of the BH method which in future implementations could be combined with the techniques described in this section to further accelerate the Barnes-Hut method in multicore and CPU/GPU clusters.

4 MPI PARALLEL BARNES-HUT VARIANTS

In this work some variants of the Barnes-Hut method are presented, all of which use the Message Passing Interface (MPI) to distribute the workload. Sparse (that is, pointer-based) octrees are used in all cases. With respect to the octree proper, two variants were considered: Full octree (each node builds the octree with all of the bodies)

*Both authors contributed equally to this research.
†Both authors contributed equally to this research.

and partial octree (each node builds the octree with part of the bodies). With respect to the workload, two variants were also considered: Static load (which each node computes the forces on a fixed range of bodies and said range depends solely on the node's rank and it is the same for all the steps of the simulation) and dynamic load (each node computes the forces on a range of bodies which varies as the simulation advances).

4.1 Full octree

At the beginning of each simulation step the bodies are sorted according to their Morton keys. Each working node builds the octree with all the bodies and computes the forces between part of the bodies and the octree. In the static variant, after the octree is built, the forces are computed on a range of the bodies. The range depends solely on the rank of the node. In the dynamic variant the *master-slave* paradigm is used (thus, to be compared to the rest of the methods, there needs to be an extra node). The master node sends a range (given by two indices) to each slave node, which computes the forces between the bodies in said range with the octree. When finished, the slave asks for more bodies. This process is repeated until all the bodies are processed by one of the slave nodes.

4.2 Partial octree

The variants where a partial octree is built need less memory at any given point in time than the full octree versions, which makes them suitable in memory-constrained situations, or simply when the number of bodies is too high. At the beginning of each simulation step the bodies are sorted according to their Morton keys. The bodies are then scattered among the nodes (which are organized in a ring), along with the limits of the space. Each node builds an octree with the received bodies and limits. The rest of the simulation step is divided in as many *phases* as nodes are. In each phase the node computes the forces that the bodies in its octree exert on a number of bodies. Afterwards, the bodies and the partial forces computed are sent to the next node in the ring. During the first phase, the node computes the forces between the bodies with which the octree was created. At the end of the last phase each node receives the bodies to it originally allotted and their corresponding, fully computed, forces. In the dynamic variant, after each simulation step, in order to make the times spent by each node more similar to each other's, the loads may need to be redistributed. This redistribution should take place before each simulation step. For the first simulation step all the times could be set to a given number, say, one. Thus initially all nodes receive the same number of bodies. In practice, it is noted that after some simulation steps it is better to make the load of all the nodes the same again, at least when using several nodes in the same computer.

4.3 Costzones

In order to lower the time needed to complete a step of the simulation a strategy called *costzones* can be used [11]. For each body, during the force-computation phase, a counter is incremented whenever a force is effectively computed (be it with a barycenter or directly with another body). For the next time step each process will deal with a range of objects such that the total cost in every range (that is, the number of interactions computed) is approximately the same, the rationale being that the positions of the bodies do not change by much between time steps.

5 RESULTS

The code was implemented as a number of C++ template classes. To obtain the following results single-precision floats were used. The experiments were performed on a single computer with an Intel Xeon Silver 4314 CPU @ 2.40GHz processor with 16 physical cores, 32GB of RAM, running Ubuntu 20.04.3 LTS. The opening parameter was set to $\theta = 0.5$. One million objects following a Plummer distribution [10] were simulated. In Figure 1 the time needed for each of the tasks performed during the first step of the simulation is represented for each of the variants developed, the number of working MPI processes being 16. Notice how the variants which employ a full octree are faster, and of these, the one with dynamic allocation of work is the fastest. The largest amount of time is devoted to the calculation of the forces, therefore this must be the focus of attention in the future to drive down the cost of these variants. In the plots corresponding to the variants with partial octrees, the white regions of the bar plots correspond to the stages in which MPI communications occur. Due to the high number of messages exchanged between the nodes and the fact that each node needs the incomplete calculations performed by the previous node to carry on its own computations, these variants are very sensible to delays from any particular node. The speedups related to each variant for the first step of the simulation are represented in Figure 2. The trend of the full octree variants' speedups, although increasing sublinearly, indicates that a plateau has not been reached yet. Given more nodes, therefore, it is expected that the speedups will still increase. The partial variants' parallel speedups are inferior. These variants also present poorer performance with respect to the full variants. The effect of using costzones to ameliorate the performance is small in any given time step, but the cumulative result can be quite significant, as seen in Figure 3. The dynamic full octree variant still outperforms the full octree variant modified to use costzones, but this could change in a real CPU cluster.

6 CONCLUSIONS

Several variants of the Barnes-Hut method for solving the n-body problem were implemented to distribute the workload MPI was used. We have analyzed our implementations in a modern multicore CPU. Future applications of our parallel/distributed methods could be combined with other state of the art techniques to further accelerate the Barnes-Hut method in multicore and CPU/GPU clusters. In the full octree variants the largest amount of time is devoted to the computation of the force, while in the partial octree variants the largest amount of time is devoted to communications between the participant nodes. The dynamic full octree variant outperforms all the other variants and shows great promise if deployed in an heterogeneous network.

7 FUTURE WORK

To improve the cache locality an implicit octree, as described in [6], will be used. This octree is laid out as an array: it is expected that the lack of pointers and the contiguity of memory will improve the

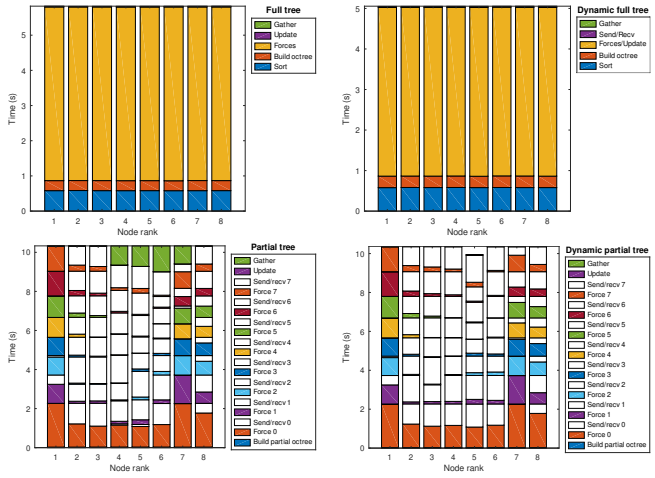

Figure 1: Time consumption for the first step of the simulation, 8 working MPI processes, 10^6 objects.

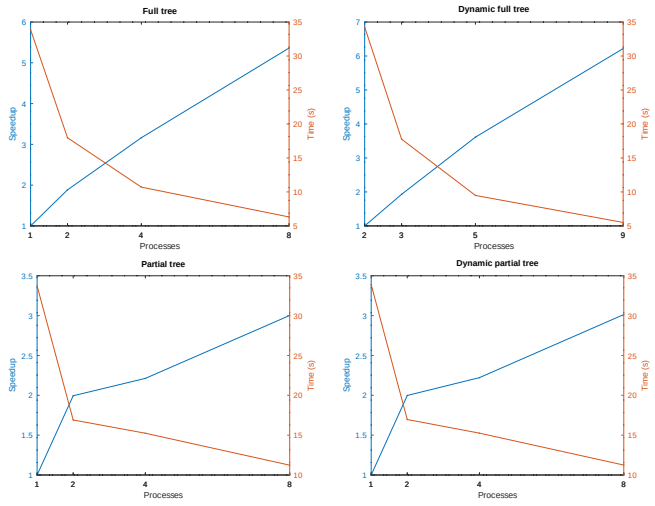

Figure 2: Speedups for 2, 4 and 8 working MPI processes with respect to a single working process for the first step of the simulation, 10^6 objects.

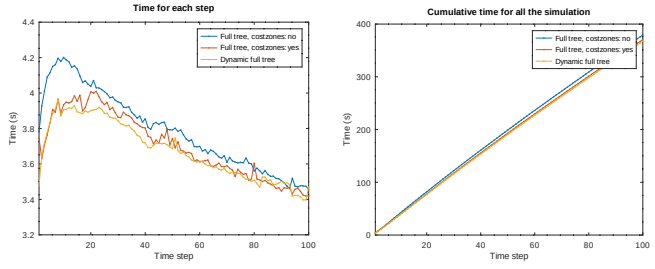

Figure 3: Impact of the use of costzones. Left: Individual step times. Right: Cumulative time. Full octree method, 100 time steps, 10^6 objects, 16 working MPI processes.

number of cache hits. A real cluster will be used, thus each process will further parallelize the computations by using OpenMP. Whether the behavior observed in the experiments described in Section 5 would be replicated in a real cluster or not will be addressed in future works. Although speedup improvements in the partial octree variants mainly depend on optimizations to be further investigated in the communication patterns to implement the workload distribution, we also expect that the use of implicit octrees can also benefit these variants to some extent, as the use of this method can possibly accelerate traversal time in the partial tree sections at the nodes.

8 ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES)-Finance Code 001.

REFERENCES

[1] Nitin Arora, Aashay Shringarpure, and Richard W. Vuduc. 2009. Direct N-body Kernels for Multicore Platforms. In *ICPP 2009, International Conference on Parallel Processing, Vienna, Austria, 22-25 September 2009*. 379–387.

[2] J. E. Barnes and P. Hut. 1986. A hierarchical $O(n \log n)$ force calculation algorithm. *Nature* 324 (1986), 446.

[3] Jeroen Bédorf, Evghenii Gaburov, and Simon Portegies Zwart. 2012. A sparse octree gravitational N-body code that runs entirely on the GPU processor. *J. Comput. Phys.* 231, 7 (April 2012), 2825–2839.

[4] Martin Burtscher and Keshav Pingali. 2011. Chapter 6 - An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm. In *GPU Computing Gems Emerald Edition*, Wen-mei W. Hwu (Ed.). Morgan Kaufmann, Boston, 75 – 92.

[5] David M. Chan, Roshan Rao, Forrest Huang, and John F. Canny. 2018. t-SNE-CUDA: GPU-Accelerated t-SNE and its Applications to Modern Data. *Proceedings - 2018 30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2018* (2018), 330–338.

[6] Armando Delgado, Rodrigo Morante Blanco, and Wagner Nunan Zola. 2019. Caminhamento Paralelo Barnes-Hut com Vetorização AVX2. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho* (Campo Grande). SBC, Porto Alegre, RS, Brasil, 454–461. https://doi.org/10.5753/wscad.2019.8691

[7] Benoit Lange and Pierre Fortin. 2014. Parallel Dual Tree Traversal on Multi-core and Many-core Architectures for Astrophysical N-body Simulations. In *Euro-Par 2014: Proceedings of the 20th International European Conference on Parallel and Distributed Computing*. 716–727.

[8] Rainer Spurzem Long Wang et al. 2015. NBODY6++GPU: ready for the gravitational million-body problem. *Monthly Notices of the Royal Astronomical Society* 450, 4 (2015), 4070–4080.

[9] Bruno Henrique Meyer, Aurora Trinidad Ramirez Pozo, and Wagner M Nunan Zola. 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 (2021), 1–26.

[10] H. C. Plummer. 1911. On the Problem of Distribution in Globular Star Clusters: (Plate 8.). *Monthly Notices of the Royal Astronomical Society* 71, 5 (03 1911), 460–470.

[11] J.P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. 1995. Load Balancing and Data Locality in Adaptive Hierarchical N-Body Methods: Barnes-Hut, Fast Multipole, and Radiosity. *J. Parallel and Distrib. Comput.* 27, 2 (1995), 118–141.

[12] Laurens van der Maaten. 2014. Accelerating t-SNE using Tree-Based Algorithms. *Journal of Machine Learning Research* 15, 93 (2014), 3221–3245.

[13] Rio Yokota. 2012. An FMM Based on Dual Tree Traversal for Many-core Architectures. *CoRR* abs/1209.3516 (2012).

[14] Ivan Zecena, Martin Burtscher, Tongdan Jin, and Ziliang Zong. 2013. Evaluating the Performance and Energy Efficiency of N-Body Codes on Multi-Core CPUs and GPUs. In *32nd IEEE International Performance Computing and Communications Conference (IPCCC'13)*.

[15] Wenbo Zhu, Zachary T. Webb, Kaitian Mao, and José Romagnoli. 2019. A Deep Learning Approach for Process Data Visualization Using t-Distributed Stochastic Neighbor Embedding. *Industrial & Engineering Chemistry Research* 58, 22 (2019), 9564–9575.