Study and analysis of parallel N-body simulations

Guilherme G. Arcencio[1], **Hélio Crestana Guardia**[1]

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
São Carlos – Brazil

garcencio@estudante.ufscar.br, helio.guardia@ufscar.br

Abstract. *The N-body Problem involves predicting the motion of a group of astronomic particles under the influence of each other's gravitational field. Due to the nonexistence of closed-form analytical solutions for N bigger than 2, numerical methods must be employed, and those do not scale well for large numbers of bodies. Thus, we developed and investigated parallel, high performance N-body simulations in GPUs and multi-core CPUs, and found that, even with dynamic time steps and synchronization requirements, GPUs massively reduce the time required by the simulations. While multi-core CPUs also allow for large speedups, they are vastly outperformed by the former.*

1. Introduction

In celestial mechanics, the N-body Problem is the prediction of the motion of N particles under the gravitational influence of each other [Meyer et al. 2009]. Given that, for $N > 2$, the problem has no general, closed-form analytical solution, it must be tackled with numerical methods and simulations.

N-body simulations are important to the study of the evolution of cosmological structures, as well as molecular dynamics, in which the gravitational interactions are replaced by Coulomb forces [Board et al. 1999]. However, each simulation step involves calculating the force vectors between every pair of particles, resulting in $O(N^2)$ complexity. Therefore, direct methods do not scale well for large values of N without the use of parallel computing.

In this paper, we develop and analyze parallel implementations of an N-body simulation for GPUs and multi-core CPUs. We also include dynamic time steps in the numerical simulations, which both enhances numerical stability and introduces synchronization challenges.

2. N-body simulations

Given N celestial bodies, the N-body problem consists of the following system of differential equations:

$$\ddot{\mathbf{q}}_i = \sum_{j \neq i} \frac{G m_j}{(||\mathbf{q}_j - \mathbf{q}_i||^2 + \epsilon^2)^{\frac{3}{2}}} (\mathbf{q}_j - \mathbf{q}_i) \qquad i = 1, \ldots, N, \tag{1}$$

where $\mathbf{q}_i$ is the position of the i-th body in three dimensional space, m_j is the mass of the j-th body, G is the gravitational constant, and ϵ is a softening constant used to avoid the singularities where $\mathbf{q}_i = \mathbf{q}_j$. Although the system can be simulated by simply evaluating all interactions in $O(N^2)$ time – the "direct method" – there are algorithms which use approximations to reduce time complexity, such as the hierarchical, tree-based Barnes-Hut algorithm [Barnes and Hut 1986], with $O(N \log N)$ complexity, and the Fast Multipole Method [Greengard and Rokhlin 1987], with linear complexity.

In order to maintain numerical stability, dynamic time steps may be used during numerical integration according to particle density. One of the most common time step criterion is $\Delta t \leq \sqrt{\eta\epsilon/||\mathbf{a}||}$, where $\mathbf{a}$ is the acceleration vector of the particle and η is a tolerance parameter [Grudić and Hopkins 2020].

N-body simulations are a popular topic in the high performance computing literature. Both direct and hierarchical algorithms were parallelized in [Zecena et al. 2013]. Their work studied performance gains and energy savings in GPU and multi-core CPU implementations and demonstrated that, using either algorithm, multi-threading greatly reduces the simulation runtime, while the use of GPUs increased efficiency by orders of magnitude. Those experiments were replicated in [Pinto et al. 2015] and [Boukhary and Colmenares 2019], to similar results.

We complement those previous works by providing an open source implementation of parallel N-body simulations and analyzing its performance. We also include dynamic time steps in the simulation to improve numerical stability and study its impact when compared to previous, fixed time step versions.

3. Methodology

Our N-body simulations were implemented by numerically integrating the system in (1) using Verlet integration [Hairer et al. 2003]:

$$\mathbf{q}_{i,n+1} = \mathbf{q}_{i,n} + (\mathbf{q}_{i,n} - \mathbf{q}_{i,n-1})\frac{\Delta t_n}{\Delta t_{n-1}} + \ddot{\mathbf{q}}_{i,n}\frac{\Delta t_n + \Delta t_{n-1}}{2}\Delta t_n, \qquad (2)$$

where the position $\mathbf{q}_i$ of each particle is updated using its previous two positions and its current acceleration vector. In the first iteration, there is only one previous position available, and thus initial velocities are used instead:

$$\mathbf{q}_{i,1} = \mathbf{q}_{i,0} + \mathbf{v}_{i,0}\Delta t_0 + \tfrac{1}{2}\ddot{\mathbf{q}}_{i,0}\Delta t_0^2. \qquad (3)$$

The n-th time step is given by $\Delta t_n = \sqrt{\eta\epsilon/|| \max_{1\leq i \leq N} \ddot{\mathbf{q}}_i^n||}$, i.e., the smallest required time step among all particles in the current iteration. The parameters used were $\eta = 0.05$ and $\epsilon = 0.001$ and the initial conditions of the N bodies were randomly generated. Algorithm 1 describes how Equations (1), (2) and (3) are used throughout the simulation.

The multi-core implementation was developed in C using the OpenMP API. The loops at lines 2 and 6 of Algorithm 1 were parallelized by having each thread responsible for calculating the acceleration and then updating the positions of a subset of particles. Line 4 is parallelized by the reduction clause in the “omp parallel for” directive.

The GPU version was implemented in CUDA, with four kernels used throughout the program. The main kernel parallelizes the loop at line 2 of Algorithm 1 and uses data tiling and shared memory for better performance. A parallel reduction kernel is used to parallelize line 4 and avoid extra memory transfers. Finally, the loop at line 6 is parallelized by two kernels: one using Equation (2) and the other Equation (3).

Algorithm 1 N-BODY SIMULATION(Q, V, M, k)

Input: N bodies of initial positions Q, initial velocities V, and masses M, and number of steps k
Output: N updated positions after k iterations

```
1: for $s = 1$ to $k$ do
2:     for $i = 1$ to $N$ do
3:         calculate acceleration $\ddot{\mathbf{q}}_i$ using Equation (1)
4:     $a \leftarrow \max_{1 \leq i \leq N} ||\ddot{\mathbf{q}}_i||$
5:     $\Delta t \leftarrow \sqrt{\eta\epsilon/||a||}$
6:     for $i = 1$ to $N$ do
7:         update positions $Q$ using Equation (2) or (3)
8: return $Q$
```

All source code has been made available on GitHub[1] under the MIT License. Multi-core experiments were run on an Intel Xeon Silver 4208 with 16 physical cores and 32 logical cores, while GPU experiments were run on an NVIDIA GeForce RTX 2080 Ti. Every test was repeated five times so as to account for circumstantial differences in runtimes.

4. Results

Simulation results on multi-core CPU are shown in Figure 1. Multi-threading greatly reduced runtimes: from 177.2s to 10.5s when $N = 25000$, from 702.9s to 41.6s when $N = 50000$, and from 2837.4s to 181.9s when $N = 100000$. Speedups are large even when there are more threads than physical cores due to the high computational intensity in the program.

However, GPU runtimes, as shown in Figure 2, are many orders of magnitude smaller than CPU runtimes. When $N = 100000$, 10 simulations steps took less than one second, while even the best CPU performance required three minutes to complete. Block sizes for kernel executions had little impact on runtimes.

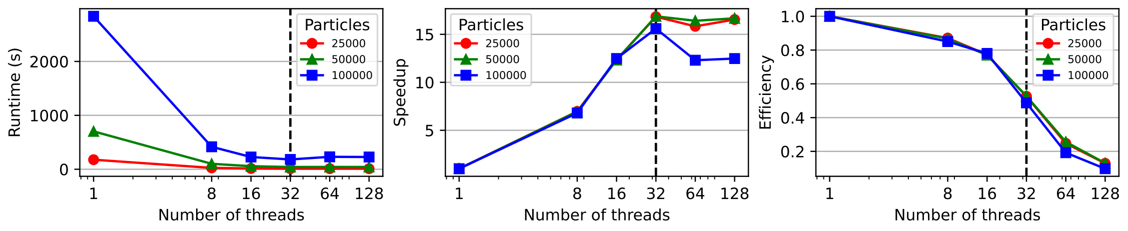

Figure 1. Simulation results on multi-core CPU after 10 steps: runtime, speedup and efficiency, respectively. The dashed line indicates the number of available cores (32).

5. Conclusions

While multi-core parallelization greatly improved N-body simulation performance, GPU implementations allow for astronomically smaller runtimes. The use of dynamic time steps, which introduces synchronization steps – finding the largest acceleration vector – had little to no impact on the excellent GPU performance.

[1]`https://github.com/GuiArcencio/parallel-nbody`

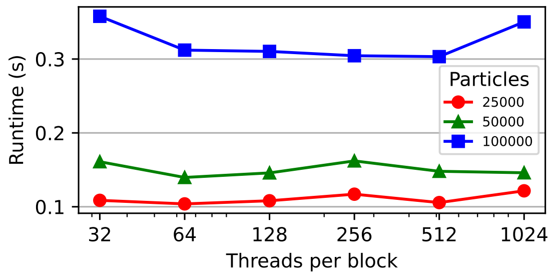

Figure 2. GPU implementation runtimes after 10 simulation steps.

Future work includes simulating and experimenting on larger numbers of particles, as well as introducing more realistic elements to the simulations, such as dark matter and gas nebulae. More parallelization tests may also be performed on different algorithms, such as the hierarchical Barnes-Hut method.

References

Barnes, J. and Hut, P. (1986). A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324(6096):446–449.

Board, J. A., Humphres, C. W., Lambert, C. G., Rankin, W. T., and Toukmaji, A. Y. (1999). Ewald and multipole methods for periodic N-body problems. In *Computational Molecular Dynamics: Challenges, Methods, Ideas*, pages 459–471, Berlin, Heidelberg. Springer.

Boukhary, S. and Colmenares, E. (2019). Study, analysis, and acceleration of an n-body simulation under many-core environments using an object oriented approach. In *2019 International Conference on Computational Science and Computational Intelligence*, pages 1506–1510. IEEE.

Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348.

Grudić, M. Y. and Hopkins, P. F. (2020). A general-purpose time-step criterion for simulations with gravity. *Monthly Notices of the Royal Astronomical Society*, 495:4306–4313.

Hairer, E., Lubich, C., and Wanner, G. (2003). Geometric numerical integration illustrated by the Störmer–Verlet method. *Acta Numerica*, 12:399–450.

Meyer, K., Hall, G., and Offin, D. (2009). *Introduction to Hamiltonian Dynamical Systems and the N-Body Problem*. Springer.

Pinto, V. G., Herbstrith, V. A., and Schnorr, L. M. (2015). Replicating the performance evaluation of an n-body application on a manycore accelerator. In *2015 International Symposium on Computer Architecture and High Performance Computing Workshop*, pages 19–24. IEEE.

Zecena, I., Burtscher, M., Jin, T., and Zong, Z. (2013). Evaluating the performance and energy efficiency of n-body codes on multi-core CPUs and GPUs. In *2013 IEEE 32nd International Performance Computing and Communications Conference*, pages 1–8. IEEE.