universität wien

A direct summation 3d simulation of a globular cluster

Focus project for Methods of Computational Astrophysics

Leonardo Quiñonez - a12411091@unet.univie.ac.at

July 7, 2025, Wien

1 State the problem

The N-body problem is a great test bed for computational methods, multiple different techniques have been developed to tackle it, some of which were explored during the course. (Hahn 2025)

The goal of this focus project is to do a direct summation 3d simulation of a globular cluster. Starting from a Plummer Model for the globular cluster, a leapfrog integrator would use efficient computations of the accelerations of each body to make a 3d simulation.

Using constant equal masses and dimensionless units the equations to model the acceleration are described by:

$$a(X) = \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{\mathbf{X}_i - \mathbf{X}_j}{\|\mathbf{X}_i - \mathbf{X}_j\|^3} \tag{1a}$$

A first version of the project would be to make the simulation using the direct summation approach which has a complexity of $\mathcal{O}(N^2)$ for N particles. Using the different libraries we will attempt to find an efficient parallelized implementation of the direct summation. This would be called by a leapfrog integration used to evolve the system and make a 3D simulation.

2 Approach

The project would have the following steps:

1. Efficient Direct Summation
2. Time Integration
3. 3D Simulation
4. Initial Conditions

Efficient Direct Summation: We will compare different implementations for the computation of the accelerations of N bodies. We will focus on testing their perfomance on high number of bodies.

- Python loops

- Vectorized Numpy
- Numba (Loops)
- Numba (Parallel)
- Jax (CPU)

Jax will only be tested on the CPU since the hardware available has no N-Vidia GPU.

Time Integration: A leapfrog method is chosen for its simplicity but also energy conservation properties. The Drift Kick Drift approach is selected to simplify the implementation and have only one acceleration calculation per time step.

3D simulation: A simple as possible 3d simulation would be attempted, a search on libraries that allow the animation would be done.

Initial Conditions: The Plummer model used during the class exercises is selected for the initial conditions.

3 Results

3.1 Efficient Direct Summation

3.1.1 Taichi

Originally the plan was to compare the implementations described in the approach section, but when searching for 3D animation libraries the Taichi Lang library[1] was found. It allows the usage of Vulkan[2] to access the Intel integrated GPU. The perfomance this allows for the 3D simulation and the Acceleration calculation is considerable, so it was additionally included in the comparison.

3.1.2 Performance Comparison

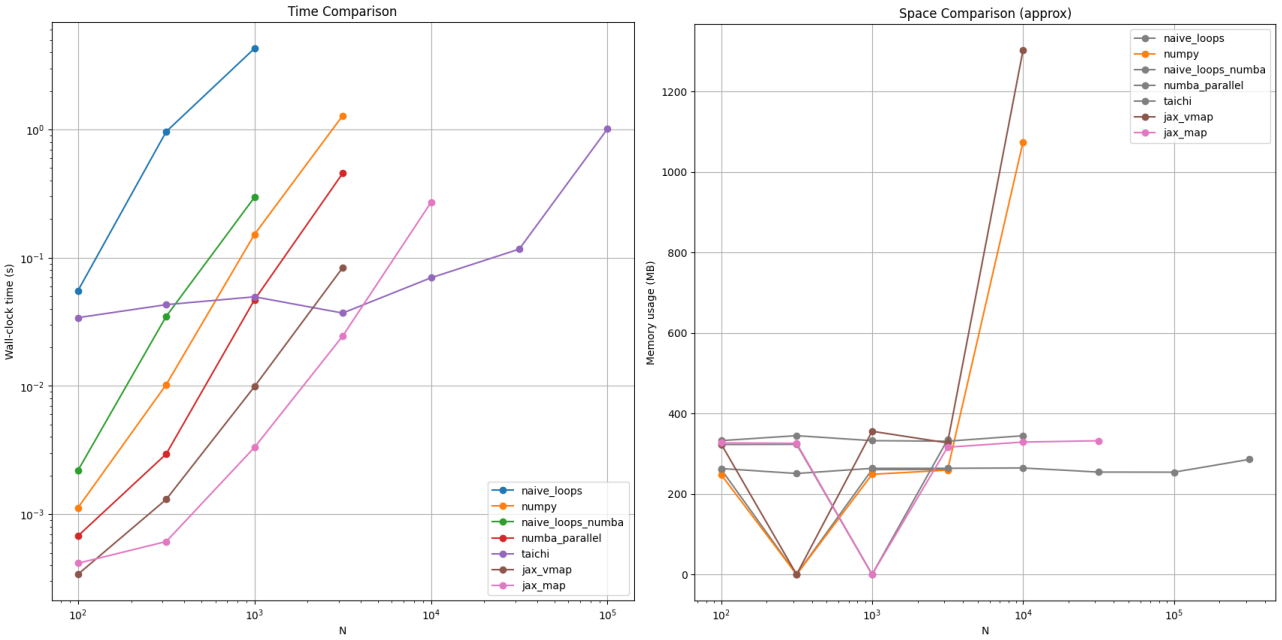
In order to compare the perfomance of the different methods we increase the number of stars exponentially until the time to perform a single acceleration computation exceeds 10s. In order to account for compilation time of jit, each implementation is run three times and the median is calculated. During this process multiple OOM(Out of Memory) errors were discovered. This prompted to the addition of a memory usage limit of 1GB as well, to accomplish this the process data is extracted from the operating system every tenth of a second, this means processes with short duration times would have unreliable calculation. Even with this inaccuracy this proved enough to avoid the OOM errors and see a full picture of all implementations performances.

Figure 1a shows the time comparison of the different implementations. As expected they show a polynomial complexity with exception of Taichi. This probably is due to the use of GPU which creates a minimum time higher than CPU based alternatives, but more tests would be needed to understand this properly. In general Jax and Taichi show the best performance, but Taichi managed to calculate one order of magnitude of stars more. It is important to note that Taichi and Jax are only using 32 bits to allow for a fair comparison. Given that the available GPU can't handle 64 bits.

Figure 1b shows the space comparison highlighting the interesting details. The numpy implementation which aimed to be a pure vectorized attempt calculates the distance of all bodies with each other, making an NxN matrix which explains the absurd memory consumption. Curiosly enough Jax

[1] Taichi Lang is a "domain-specific language embedded in Python" but for practical purposes it is installed any other library. https://www.taichi-lang.org/

[2] Vulkan Wikipedia Entry: https://en.wikipedia.org/wiki/Vulkan



(a) Time Comparison

(b) Space Comparison

Figure 1: Time and Space comparison capped to 10s and 1GB

using `vmap` presents a similar memory issue[3]. It seems to be the case that the current implementation requires as intermediate memory the vector of distances for each star, which again reaches what appears to be a NxN memory footprint. An improved implementation with Jax's `lax.map` gave better results as shown in the graph, but the additional effects of the change are yet to be understood. `lax.map` allows to set a batch size, the small tests on this increased progressively the memory usage without a visible impact on performance.

3.2 3D simulation

As

4 Discussion

5 Future

Test with an Nvidia gpu to compare Jax performance with Taichi on equal grounds. Implement a hierarchical method to reduce time complexity Do a measurement of the energy of the system and maybe implement a better integration given the results. It would be interesting as well to see the simulation works under a spectral method that allows higher number of bodies.

5.1 Tabellen

Cell 1	Cell 2	Cell 3
Cell 4	Cell 5	Cell 6
Cell 7	Cell 8	Cell 9

[3]Vmap Memory Usage Considerations: https://apxml.com/courses/getting-started-with-jax/chapter-4-automatic-vectorization-vmap/vmap-performance

References

Hahn, Prof Oliver (2025). *Methods of Computational Astrophysics*. Class notes, Methods of Computational Astrophysics (280522).