LAMAR - Leapfrog-based time Advancement for Multibody Astrophysical Rendering with GPUs

James Sunseri
Department of Astrophysical Sciences, Princeton University, 4 Ivy Lane, Princeton, NJ 08540, USA
(Dated: May 14, 2024)

In this report we present a direct summation N-body code to simulate collisionless particles interacting through gravity. This code makes use of a 2nd order symplectic leapfrog integrator commonly known as the Verlet scheme which employs a standard Drift-Kick-Drift (DKD) procedure to updating particle positions and velocities. This code makes use of NVIDIA Graphical Processing Units (GPUs) through the `CUDA` programming language. Writing this code to exploit the extreme parallel nature of the GPU allows us to simulate $\sim 10^5$ particles on short time scales. This extreme parallelization allows us to simulate complex systems commonly found in astrophysical contexts. We present several simulations using the LAMAR code.

I. INTRODUCTION

Collisionless N-body simulations have been one of the most powerful tools in astrophysics for years. N-body simulations are used to make predictions for a wide variety of systems ranging from planetary to cosmological in scale. On paper, the most naïve implementation is quite simple, we simply have to refer to Newton's laws of gravitation to write down what the acceleration each particle should experience due to the gravitational pull from all the other particles

$$\mathbf{F}_i = m_i \ddot{\mathbf{x}}_i = m_i \sum_{j\neq i}^{N} G m_j \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_i - \mathbf{x}_j|^3} \ , \tag{1}$$

where $\mathbf{x}_i$ is the position of the i-th particle, G is Newton's gravitational constant, and m_j refers to the mass of the j-th particle. There are many more ways to make this more complicated, but for the purposes of understanding collisionless stellar and galactic systems such as globular clusters and stellar disks, this is plenty sufficient. This idea is at the core of the physics behind state-of-the-art large cosmological N-body simulations such as `Gadget-IV` [1] or `RAMSES` [2] and other smaller scale state-of-the-art N-body integrators such as `REBOUND` [3]. Admittedly, those codes are far more sophisticated as they include cleverly optimized algorithms to avoid the $\mathcal{O}(N^2)$ complexity scaling required by the Brute-Force algorithm (arising from the required double `for` loop) amongst other huge optimizations to improve memory usage and reduce inefficient/unnecessary computations.

There are many ways to update the dynamics of the particles in our simulation once we've calculated the total acceleration on each particle. In Section II we will discuss exactly how updating the positions and velocities of particles is done with the GPU architecture, Section III highlights the results of our code LAMAR by showcasing some example simulations of physical systems, and finally Section IV will conclude with some remarks about the project and have a discussion of potential future improvements of the code.

II. METHODS

In this section we explain the methodology behind the `LAMAR` code by discussing the parallelization scheme, force calculation technique, and finally the integration scheme.

A. GPU Parallelization

The GPU architecture is designed to perform enormously large amounts of computations in parallel because they were originally designed to handle graphics computations for computer screens. These devices are optimized for calculating updates to displays which are really 3D arrays with dimensions $(N_{\text{pixels,x}} \times N_{\text{pixels,y}} \times 4)$ where the 4 refers to the Red, Green, Blue, and Alpha (RGBA) color channels. The GPU architecture also follows a similar design to reflect this need, whereby it is designed to be a grid filled with blocks of individual threads which are not nearly as sophisticated/complex as a common CPU thread, but they can perform basic calculations quickly. This design allows the GPU to disperse it's numerous threads in parallel to tackle a problem. Matrix operations are the most commonly used applications of GPUs but they can be used in other contexts such as our N-body solver.

To understand how we leverage the GPU, we must first describe the brute force algorithm. This algorithm is shown schematically below.

```
# at any given time step
for i in range(N):
    a = 0
    for j in range(N):
        if i != j:
            a += Calc_Acceleration(i, j, dt)
    Update_Particle(i, a, dt)
```

from here we can see where the $\mathcal{O}(N^2)$ complexity scaling comes from, at every time step we have to iterate N^2 times to update all the particles. Where the GPU comes in handy is that the first `for` loop can actually be computed in parallel by distributing all of the particles onto their own individual GPU thread. This distribution of the `for` loop means that this algorithm effectively has a $\mathcal{O}(N)$ scaling as long as there are sufficient GPU threads available to distribute the particles to their own individual GPU thread.

B. Force Calculations

To compute the total force (and then acceleration) of each particle we use 1 with what is commonly known as a gravitational softening kernel. We use a softening kernel because the divergence found in computing $\mathbf{F}_i$ is non-physical as the separation of particles becomes sufficiently small (See Chapter 2 of [4]). In this case the softening kernel is defined as

$$S(\mathbf{r}) = -\frac{1}{\sqrt{r^2+\epsilon^2}} \ , \tag{2}$$

and the force softening kernel $S_F(\mathbf{r})$ is the derivative of the softening kernel (ϵ is the softening scale parameter). Using this we can rewrite what the acceleration will be of the i-th particle

$$\ddot{\mathbf{x}}_i = \sum_{j\neq i}^{N} G m_j S_F(|\mathbf{x}_j - \mathbf{x}_i|)\frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|} \tag{3}$$

$$= \sum_{j\neq i}^{N} G m_j \frac{\mathbf{x}_j - \mathbf{x}_i}{\left(|\mathbf{x}_i - \mathbf{x}_j|^2 + \epsilon^2\right)^{3/2}} \ , \tag{4}$$

and we can also define the potential of the particle as well to be

$$\Phi_i = \sum_{j\neq i}^{N} G m_j S(|\mathbf{x}_j - \mathbf{x}_i|) \tag{5}$$

$$= -\sum_{j\neq i}^{N} G m_j \frac{\mathbf{x}_j - \mathbf{x}_i}{\sqrt{|\mathbf{x}_i - \mathbf{x}_j|^2 + \epsilon^2}} \ . \tag{6}$$

This is how we compute the forces and potentials for all of our particles in our simulation. Depending on the system we are simulating we can choose ϵ accordingly. A natural choice for ϵ is of order the inter-particle separation.

C. Time Integration

To numerically integrate the system through time we have discretize time and update the positions and velocities of each particle in incremental time steps. The time steps can be variable or constant in size so long as they are sufficiently small. In this code, we use the 2nd-order Leapfrog method also known as the Verlet method which employs a DKD update scheme. This is an explicit scheme which means it is relatively simple to implement in comparison to implicit schemes. The collionless N-body problem is a symplectic system which has a seperable Hamiltonian of the form

$$H(\mathbf{p},\mathbf{q}) = K(\mathbf{p}) + V(\mathbf{q}) \ , \tag{7}$$

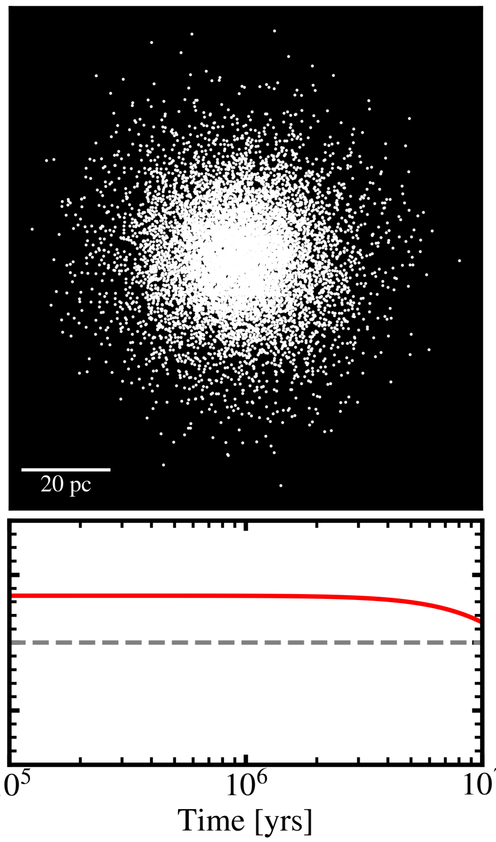

FIG. 1: This figure shows our simulation of M15 using a King distribution. The top panel shows the positions of the stars and the bottom panel shows the deviation from the Virial Theorem as a function of time.

and the total energy of the system is conserved $\dot{H} = 0$. Symplectic systems have the neat property that the volume element in phase-space is conserved. Commonly used explicit schemes for numerical integration that are not symplectic by design such as the famous Runge-Kutta 4th-order (RK4) are not ideal for these systems because they do not preserve these properties of the symplectic system. A common approach to devise an explicit symplectic integration scheme is through an operator splitting approach where we update one variable in time, and then use that updated variable to compute the remaining variable(s). If one can write the split operations in matrix form as a series of linear equations, then if the determinant of the Jacobian matrix at each step is unity the scheme is symplectic. These symplectic integrators are strictly time reversible and preserve what is known as the shadow Hamiltonian which is a first order approximation of the true Hamiltonian of the system. The simplest symplectic scheme to integrate the system is known as the symplectic Euler scheme where we update the velocity and then use that updated velocity to update the position, this is only a 1st-order accurate scheme though which is not ideal. In `LAMAR` we use the

2nd-order Verlet scheme which is also commonly referred to as a Leapfrog scheme because of the operator splitting. The procedure is as follows

$$\mathbf{v}_i^{n+1/2} = \mathbf{v}_i^n + \ddot{\mathbf{x}}_i^n \frac{\Delta t}{2} \tag{8}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^{n+1/2} \Delta t \tag{9}$$

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \ddot{\mathbf{x}}_i^{n+1} \frac{\Delta t}{2} \tag{10}$$

where n denotes the temporal index and i denotes the i-th particle. This scheme requires us to compute the acceleration caused by gravitational forces twice for every time step, but the added gain of being second order accurate makes the additional cost worth it. Having a higher-order scheme than 1st-order also allows us to use larger time steps to achieve the same level of accuracy ultimately offsetting the computational cost. Updates to the velocity are often referred to in the literature as Drifts and updates to the position are referred to as Kicks, we can see from the equations that this scheme uses a Drift-Kick-Drift operator splitting which has been shown to be better than Kick-Drift-Kick operator splitting for 2nd-order. One can play this Drift-Kick operator splitting game over and over again to achieve extremely high-order schemes. The Yoshida scheme is 4th-order requiring 7 operator splittings in order of KDKDKDK. There are several symplectic integrators in `REBOUND`[3] that use this approach, and the Verlet scheme in particular is extremely popular in N-body cosmological codes like `GADGET-IV`[1] and `RAMSES`[2].

III. RESULTS

In this section we show the results of our simulations for two types of systems. We simulate globular clusters and we simulate stellar disks. The details of how we simulate these systems is discussed below. We summarize all the simulation parameters in Table I.

A. Globular Clusters

As an example of what our code can do, we simulated a model of the M15 globular cluster. To do so we assume King profile as done in [5] and we use the parameters in Table 5 of their work. To create the initial conditions we use a distribution function (DF) for the King profile described in [6] and implemented in `Galpy` [7]. The distribution function we used is

$$f(\mathcal{E}) = \frac{\rho_0}{(2\pi\sigma^2)^{3/2}} \left[\exp\left(\frac{\Psi - \frac{v^2}{2}}{\sigma^2} \right) - 1 \right] \tag{11}$$

for values of the relative energy $\mathcal{E} > 0$ and zero otherwise. This family of models are known as King models and were popularized for modeling globular clusters [8]. The King

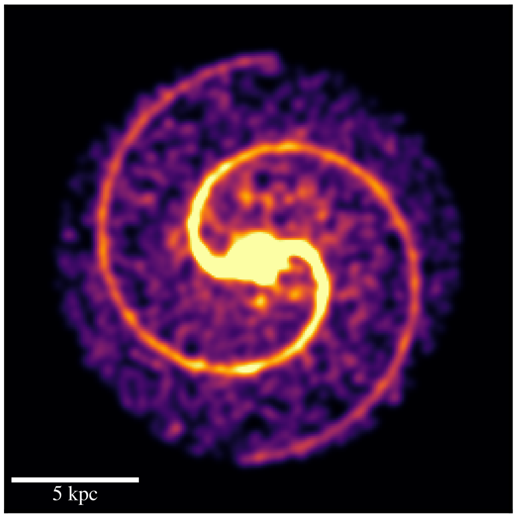

FIG. 2: This figure shows an example of a stellar disk simulated using `LAMAR`. The simulation is meant to crudely model the stellar disk in the Milky Way after $\sim 5 \times 10^6$ years.

DF is a spherical distribution which is why we write it as a function of the relative energy and relative potential define as

$$\mathcal{E} = -E + \Phi_0 = \Psi - \frac{1}{2} v^2 \tag{12}$$

$$\Psi = -\Phi + \Phi_0 \; . \tag{13}$$

We assume the system is isolated which means $\Phi(\infty) = 0$ which allows us to set $\Phi_0 = 0$ and thus the relative energy is just the binding energy $\mathcal{E} = -E$. Using `Galpy` we are able to sample this distribution function and feed the initial positions and velocities directly into `LAMAR` which we can evolve over time. In principle the system should be virialized which means we can apply the virial theorem

$$\langle K \rangle = -\frac{1}{2} \langle U \rangle \; , \tag{14}$$

where $\langle K \rangle$ and $\langle U \rangle$ can be computed by

$$\langle K \rangle = \frac{1}{2} \sum_i^N m_i v_i^2 \tag{15}$$

$$\langle U \rangle = \frac{1}{2} \sum_i^N m_i \Phi_i \; , \tag{16}$$

which is easily computed with `LAMAR`. In Figure 1 we show how the simulation agrees with the Virial Theorem to $\sim 1\%$ precision for the duration of the simulation. This is a nice test for `LAMAR`.

B. Stellar Disks

For this report we choose to highlight an attempt at creating a system similar to the stellar disk in the Milky

	$N_{\text{particles}}$	$t_{\max}$	Δt	ϵ
M15 Globular Cluster	10^4	10^7 yrs	10 yrs	3 pc
Milky Way Disk	4000	10^9 yrs	10^3 yrs	0.1 kpc

TABLE I: This table summarizes the parameters used in the simulations.

Way. To do this, we start with a base uniform disk sampled from $R = [1, 8]$ kpc, and then we add a set of particles following a $\cos(m\phi)$ spiral pattern, this is inspired by the surface density of a tightly wound spiral pattern which has the form

$$\Sigma(R, \phi, t) = H(R, t)e^{i[m\phi + f(R,t)]} \tag{17}$$

where the physical surface mass density is the real part of the expression. We have that $f(R, t)$ is the shape function and $H(R, t)$ is a slowly varying function of radius that gives the amplitude of the spiral pattern. The results of this simulation are shown in Figure 2. We use $m = 1$ for the simulation shown in Figure 2, and we draw a second spiral offset in angle by π radians. To keep the disk bound we use a central point mass representing a supermassive black hole. Because there is no dark matter halo in our simulation, we make the crude approximation that the black hole at the center contains all of the mass expected in dark matter out to $R \sim 8$ kpc. This configuration is inherently unstable so the inner most regions of the disk get sucked into the center with extreme gravity creating wild patterns. For the sake of demonstration we show a snapshot from the simulation at $\sim 5 \times 10^6$ years. There are surely better ways to create a proper disk, but within the time constraints of the project, this was the best we could do. We tried experimenting with razor-thin disks by using `Galpy` to sample the distribution function of both cold and warm 2D thin disks. Those are implemented and available to play with in the Github: https://github.com/James11222/LAMAR.

IV. CONCLUSIONS

To conclude, we wrote the code `LAMAR` which uses GPUs and a 2nd-order accurate Verlet Leapfrog scheme with gravitational softening to simulate collisionless gravitational N-body simulations. We used the code two simulate two types of systems, Globular Clusters and Stellar Disks. We simulated a model of the M15 Globular cluster using a King Distribution Function to create the initial conditions and we used several different initial distributions to simulate a stellar disk.

Discussion: This code was fun to write, and we learned how to write `CUDA` code for GPUs which will ultimately be useful in my computational research later on. With more time, it would be a good idea to implement a significantly more optimized version of the code which uses the Barnes-Hut algorithm and k-Tree data structures to create a complexity scaling of $\mathcal{O}(N \log N)$ which enables analyses of millions or billions of particles. That would allow us to look at significantly larger problems like the distribution of cosmological cold dark matter particles. Another improvement that could be made to the code is adaptive time stepping which reduces the numerical drift in phase-space.

ACKNOWLEDGEMENTS

We would like to thank Robel Geda for his advice and guidance on this project. This project would not have been possible without extensive discussions and his mentorship. We encourage anyone who reads and is interested in writing their own version to take a look at Robel's version of a GPU based N-body code as well called `KENDRICK` which `LAMAR` is based off of. This project made use of the Adroit computing cluster hosted by the Princeton Institute for Computational Science and Engineering (PICSciE) and Office of Information Technology's Research Computing at Princeton University.

[1] V. Springel, R. Pakmor, O. Zier, and M. Reinecke, Simulating cosmic structure formation with the GADGET-4 code, MNRAS **506**, 2871 (2021), arXiv:2010.03567 [astro-ph.IM].

[2] R. Teyssier, RAMSES: A new N-body and hydrodynamical code, Astrophysics Source Code Library, record ascl:1011.007 (2010).

[3] H. Rein and S.-F. Liu, REBOUND: Multi-purpose N-body code for collisional dynamics, Astrophysics Source Code Library, record ascl:1110.016 (2011).

[4] J. Binney and S. Tremaine, *Galactic Dynamics: Second Edition* (2008).

[5] A. Pasquali, G. De Marchi, L. Pulone, and M. S. Brigas, The global mass function of M 15, A&A **428**, 469 (2004), arXiv:astro-ph/0407553 [astro-ph].

[6] J. Bovy, Dynamics and astrophysics of galaxies, `https://galaxiesbook.org`, accessed: 2024-05-10.

[7] J. Bovy, galpy: A python Library for Galactic Dynamics, ApJS **216**, 29 (2015), arXiv:1412.3451 [astro-ph.GA].

[8] I. R. King, The structure of star clusters. III. Some simple dynamical models, AJ **71**, 64 (1966).