Master in High Performance Computing

ISTEDDAS: a new direct N-Body code on GPU to study merging compact-object binaries in star clusters

Supervisor(s):
Mario Spera,
Ivan Girotto

Candidate:
Mattia Mencagli

7th edition
2020–2021

Master in
High Performance Computing

Contents

Abstract

On February 11th, 2016, the LIGO and Virgo scientific collaborations announced the first direct detection of gravitational waves (GWs), a signal caught by the LIGO interferometers on September 14th, 2015, and produced by the coalescence of two stellar-mass black holes. The discovery represented the beginning of an entirely new way to investigate the Universe. The latest gravitational-wave catalog by LIGO, Virgo, and KAGRA brings the total number of gravitational-wave events to 90, and the count is expected to significantly increase in the next years when additional ground-based and space-born interferometers will be operational. From the theoretical point of view, we have only fuzzy ideas on where the detected events came from, and the answers to most of the five Ws and How for the astrophysics of compact binary coalescences are still unknown.

However, two main formation channels have been proposed so far for the formation of merging compact objects (neutron stars - NSs, and black holes BHs). In the isolated binary channel, two progenitor stars are bound since their formation, evolve, and then turn into (merging) compact objects at the end of their life, without experiencing any kind of external perturbation. This scenario is driven by single and binary stellar evolution processes, and it is sometimes referred to as the "field" scenario because it assumes that binaries are born in low-density environments, i.e., that they evolve in isolation. In contrast, in the dynamical channel, two compact objects get very close to each other after one (or more) gravitational interactions with other stars or compact objects. This evolutionary scenario is quite common in dense stellar environments (e.g., star clusters), and it is driven mainly by stellar dynamics. In reality, the two formation pathways might have a strong interplay. In star clusters, the orbital parameters of binaries might be perturbed by many passing-by objects.

One of the main issues is that most stars form in dense stellar environments, and numerical simulations of merging compact-object binaries in such crowded stellar systems are very challenging. However, to investigate the origins and the properties of merging compact objects we need a powerful N-body code, which can handle, at the same time, the large spatial and time-evolutionary scales of star clusters ($\sim$pc and $\sim$Gyr), and the small scales typical of tight binaries ($\sim$AU and $\sim$days). Therefore, in this thesis, I discuss the innovative algorithms behind ISTEDDAS, a new direct N-body code I developed in C++ from scratch that natively supports CUDA to run on Graphics Processing Unit (GPU) supercomputers. I coupled ISTEDDAS with the few-body code TSUNAMI, which numerically integrates the orbits of tight systems (e.g., binaries or three-body encounters) with very high accuracy, and also with the population-synthesis code SEVN, which

includes up-to-date prescriptions for the evolution of both single and binary stars.

In this Thesis, I will explain the complex machinery behind ISTEDDAS. In particular, the second, third, and fourth chapters are overviews of ISTEDDAS, TSUNAMI, and SEVN, respectively. In those chapters, I will go through the implementation details of the codes and I will explain how they are interfaced with each other. In the fourth chapter, I will show some results that validate the first version of the ISTEDDAS code.

Chapter 1

The direct N-body code ISTEDDAS on GPU

In this chapter, I will give an overview of the strategies adopted in our code to numerically solve the N-body problem using GPUs. ISTEDDAS (the word for "stars" in Sardinian language) is a direct N-body code that implements the Hermite 6th order time integrator combined with the block time steps method, and the Ahmad-Cohen neighbours scheme. ISTEDDAS is written by combining utilities of C and C++ and it uses CUDA, MPI, and OpenMP to exploit GPU workstations as well as GPU clusters [21]. The main features of ISTEDDAS are described in the following sections.

1.1 The Hermite integrator

The Hermite time integrators (4th, 6th and 8th order) represent the state-of-the-art for direct N-body simulations [2, 4]. The 4th order scheme is, by far, the most used algorithm in this context [1, 6, 20, 25, 31, 32], but the 6th order can be more accurate and almost as fast as the 4th order when implemented on GPUs [24]. The Hermite integrators are based on the Taylor series of positions, velocities, and accelerations and their most important feature is that they need to evaluate the distances between particles just once per time step. For instance, this is a huge advantage compared to a classic 4th order Runge-Kutta method, which needs to evaluate accelerations 4 times per integration step.

To evolve a stellar system from time t_0 to time t_1 ($\Delta t = t_1 - t_0$, 1 integration step), the Hermite 6th order integrator takes advantage of a predictor-evaluation-corrector scheme. The prediction step performs a Taylor expansion on stars' positions ($\boldsymbol{r}$)[1],

[1]Throughout the manuscript, I use bold letters to refer to three-dimensional vectors.

velocities ($\boldsymbol{v}$) and accelerations ($\boldsymbol{a}$), using time derivatives up to $\dddot{\boldsymbol{a}}$:

$$\boldsymbol{r}_p = \boldsymbol{r}_0 + \boldsymbol{v}_0 \Delta t + \boldsymbol{a}_0 \frac{\Delta t^2}{2} + \dot{\boldsymbol{a}}_0 \frac{\Delta t^3}{6} + \ddot{\boldsymbol{a}}_0 \frac{\Delta t^4}{24} + \dddot{\boldsymbol{a}}_0 \frac{\Delta t^5}{120}, \tag{1.1}$$

$$\boldsymbol{v}_p = \boldsymbol{v}_0 + \boldsymbol{a}_0 \Delta t + \dot{\boldsymbol{a}}_0 \frac{\Delta t^2}{2} + \ddot{\boldsymbol{a}}_0 \frac{\Delta t^3}{6} + \dddot{\boldsymbol{a}}_0 \frac{\Delta t^4}{24}, \tag{1.2}$$

$$\boldsymbol{a}_p = \boldsymbol{a}_0 + \dot{\boldsymbol{a}}_0 \Delta t + \ddot{\boldsymbol{a}}_0 \frac{\Delta t^2}{2} + \dddot{\boldsymbol{a}}_0 \frac{\Delta t^3}{6}, \tag{1.3}$$

where the quantities labeled with p are the predicted ones and the quantities labeled with 0 come from the corrector performed in the previous step, or from initial conditions.

During the evaluation step, the code computes the total force on the star i from all the other stars in the system, using the classical Newtonian formula and the predicted values $\boldsymbol{p}$, $\boldsymbol{v}$, and $\boldsymbol{a}$:

$$\boldsymbol{a}_{i,1} = \sum_{j \neq i}^{N} \frac{m_j}{|\boldsymbol{r}_{ij,p}|^3} \boldsymbol{r}_{ij,p}, \tag{1.4}$$

$$\dot{\boldsymbol{a}}_{i,1} = \sum_{j \neq i}^{N} \frac{m_j}{|\boldsymbol{r}_{ij,p}|^3} \left(\boldsymbol{v}_{ij,p} - 3 \frac{\boldsymbol{r}_{ij,p} \cdot \boldsymbol{v}_{ij,p}}{|\boldsymbol{r}_{ij,p}|^2} \boldsymbol{r}_{ij,p} \right), \tag{1.5}$$

$$\begin{aligned}\ddot{\boldsymbol{a}}_{i,1} = \sum_{j \neq i}^{N} \frac{m_j}{|\boldsymbol{r}_{ij,p}|^3} \Bigg[\boldsymbol{a}_{ij,p} - 6 \frac{\boldsymbol{r}_{ij,p} \cdot \boldsymbol{v}_{ij,p}}{|\boldsymbol{r}_{ij,p}|^2} \boldsymbol{v}_{ij,p} + \\ + \frac{3}{|\boldsymbol{r}_{ij,p}|^2} \left(5 \frac{(\boldsymbol{r}_{ij,p} \cdot \boldsymbol{v}_{ij,p})^2}{|\boldsymbol{r}_{ij,p}|^2} - \boldsymbol{r}_{ij,p} \cdot \boldsymbol{a}_{ij,p} - |\boldsymbol{v}_{ij,p}|^2 \right) \boldsymbol{r}_{ij,p} \Bigg],\end{aligned} \tag{1.6}$$

where the quantities with subscript "ij,p" are the relative predicted values between the stars i and j (e.g. $\boldsymbol{r}_{ij,p} = \boldsymbol{r}_{j,p} - \boldsymbol{r}_{i,p}$), and the gravitational constant it is assumed as $G = 1$. Finally, in the corrector step, the predicted quantities are corrected to obtain a more accurate solution using the newly evaluated accelerations. Positions and velocities are corrected as:

$$\boldsymbol{v}_c = \boldsymbol{v}_0 + (\boldsymbol{a}_1 + \boldsymbol{a}_0) \frac{\Delta t}{2} - (\dot{\boldsymbol{a}}_1 - \dot{\boldsymbol{a}}_0) \frac{\Delta t^2}{10} + (\ddot{\boldsymbol{a}}_1 + \ddot{\boldsymbol{a}}_0) \frac{\Delta t^3}{120}, \tag{1.7}$$

$$\boldsymbol{r}_c = \boldsymbol{r}_0 + (\boldsymbol{v}_c + \boldsymbol{v}_0) \frac{\Delta t}{2} - (\boldsymbol{a}_1 - \boldsymbol{a}_0) \frac{\Delta t^2}{10} + (\dot{\boldsymbol{a}}_1 + \dot{\boldsymbol{a}}_0) \frac{\Delta t^3}{120}. \tag{1.8}$$

During the correction phase, a new time step for each star is computed using the new accelerations, as in [24]:

$$\Delta t = \eta_h \sqrt[6]{\frac{|\boldsymbol{a}_1||\ddot{\boldsymbol{a}}_1| + |\dot{\boldsymbol{a}}_1|^2}{|\boldsymbol{a}_1^{(5)}||\dddot{\boldsymbol{a}}_1| + |\boldsymbol{a}_1^{(4)}|^2}}, \tag{1.9}$$

where η_h is an accuracy parameter that set to 0.4 [6, 8]. $\dddot{\boldsymbol{a}}_1$, $\boldsymbol{a}_1^{(4)}$, and $\boldsymbol{a}_1^{(5)}$ are the third, fourth, and fifth derivatives of the acceleration. Furthermore, $\dddot{\boldsymbol{a}}_1$ is saved for the next iteration since it is needed in the predictor for the Taylor expansion.

1.1.1 Polynomial interpolation for the sixth-order integrator

For each star, the third, fourth, and fifth derivatives of the acceleration at the next step ($\dddot{\boldsymbol{a}}_1$, $\boldsymbol{a}_1^{(4)}$, and $\boldsymbol{a}_1^{(5)}$) are estimated analytically during the corrector step using $\boldsymbol{a}_0$, $\boldsymbol{a}_1$, $\dot{\boldsymbol{a}}_0$, $\dot{\boldsymbol{a}}_1$, $\ddot{\boldsymbol{a}}_0$ and $\ddot{\boldsymbol{a}}_1$ [24]. It is convenient to define the following summations and differences between them as:

$$\boldsymbol{A}^+ = \boldsymbol{a}_1 + \boldsymbol{a}_0\,, \tag{1.10}$$
$$\boldsymbol{A}^- = \boldsymbol{a}_1 - \boldsymbol{a}_0\,, \tag{1.11}$$
$$\boldsymbol{J}^+ = h(\dot{\boldsymbol{a}}_1 + \dot{\boldsymbol{a}}_0)\,, \tag{1.12}$$
$$\boldsymbol{J}^- = h(\dot{\boldsymbol{a}}_1 - \dot{\boldsymbol{a}}_0)\,, \tag{1.13}$$
$$\boldsymbol{S}^+ = h^2(\ddot{\boldsymbol{a}}_1 + \ddot{\boldsymbol{a}}_0)\,, \tag{1.14}$$
$$\boldsymbol{S}^- = h^2(\ddot{\boldsymbol{a}}_1 - \ddot{\boldsymbol{a}}_0)\,, \tag{1.15}$$

where $h = \Delta t/2 = (t_1 - t_0)/2$. The coefficients of the polynomial interpolation at the midpoint $t = h$ are:

$$\boldsymbol{a}_{1/2} = \frac{1}{16}\left(8\boldsymbol{A}^+ - 5\boldsymbol{J}^- + \boldsymbol{S}^+\right)\,, \tag{1.16}$$
$$h\dot{\boldsymbol{a}}_{1/2} = \frac{1}{16}\left(15\boldsymbol{A}^- - 7\boldsymbol{J}^+ + \boldsymbol{S}^-\right)\,, \tag{1.17}$$
$$\frac{h^2}{2}\ddot{\boldsymbol{a}}_{1/2} = \frac{1}{8}\left(3\boldsymbol{J}^- - \boldsymbol{S}^+\right)\,, \tag{1.18}$$
$$\frac{h^3}{6}\dddot{\boldsymbol{a}}_{1/2} = \frac{1}{8}\left(-5\boldsymbol{A}^- + 5\boldsymbol{J}^+ - \boldsymbol{S}^-\right)\,, \tag{1.19}$$
$$\frac{h^4}{24}\boldsymbol{a}_{1/2}^{(4)} = \frac{1}{16}\left(-\boldsymbol{J}^- + \boldsymbol{S}^+\right)\,, \tag{1.20}$$
$$\frac{h^5}{120}\boldsymbol{a}_{1/2}^{(5)} = \frac{1}{16}\left(3\boldsymbol{A}^- - 3\boldsymbol{J}^+ + \boldsymbol{S}^-\right)\,. \tag{1.21}$$

Thus $\dddot{\boldsymbol{a}}_1$, $\boldsymbol{a}_1^{(4)}$, and $\boldsymbol{a}_1^{(5)}$ are obtained by expanding their midpoint counterparts to t_1:

$$\dddot{\boldsymbol{a}}_1 = \dddot{\boldsymbol{a}}_{1/2} + h\boldsymbol{a}_{1/2}^{(4)} + \frac{h^2}{2}\boldsymbol{a}_{1/2}^{(5)}\,, \tag{1.22}$$
$$\boldsymbol{a}_1^{(4)} = \boldsymbol{a}_{1/2}^{(4)} + h\boldsymbol{a}_{1/2}^{(5)}\,, \tag{1.23}$$
$$\boldsymbol{a}_1^{(5)} = \boldsymbol{a}_{1/2}^{(5)}\,. \tag{1.24}$$

Finally, by integrating, we obtain the sixth-order corrector:

$$\boldsymbol{v}_1 = \boldsymbol{v}_0 + h\left(\boldsymbol{A}^+ - \frac{2}{5}\boldsymbol{J}^- + \frac{1}{15}\boldsymbol{S}^+\right)\,, \tag{1.25}$$
$$\boldsymbol{r}_1 = \boldsymbol{r}_0 + h\boldsymbol{v}_1 + h^2\left(-\frac{2}{5}\boldsymbol{A}^- + \frac{1}{15}\boldsymbol{J}^+\right)\,, \tag{1.26}$$

here $\boldsymbol{v}_1$ and $\boldsymbol{r}_1$ correspond exactly to $\boldsymbol{v}_c$ and $\boldsymbol{r}_c$ in Eqs.1.7,1.8.

1.2 The block time steps method

Stars in an N-body system can have very different accelerations. This corresponds to have a large variety of evolutionary dynamical time scales. In this context, it is convenient to assign to all stars their individual time step, which becomes a function of the physical parameters that describe the kinematic state of stars. To avoid time-synchronization issues between the stars, and to simplify the parallelization process, the time step of the i-th star is discretized to powers of two: $\Delta t_i^{discretized} = 2^{-n}$, where n is an integer. In the code, by default, the maximum and minimum time steps are $\Delta t_{max} = 2^{-3}$ and $\Delta t_{min} = 2^{-35}$ respectively. For each star, we use Eq.1.9 to calculate the ideal time step Δt_i^{real}, but we always approximate n to the largest integer value, so the effective time step is $\Delta t_i^{discretized} \leq \Delta t_i^{real}$. Thus, particles are sub-divided into several groups (blocks) that share the same time step [3] and the groups are indexed according to n. The latter strategy is very effective to select, for each step, the stars that need to be integrated, since we need to update positions and velocities only for m stars per time step, where $m \leq N$. Specifically, stars with smaller time steps will be updated more often than stars with larger time steps, and for the latter, the kinematic state will be estimated using the predictor step only. This implies that the computational complexity is reduced from $O(N^2)$ to $O(mN)$. This is a significant gain in terms of performance, with a negligible cost in terms of the precision of the integrator.

Fig.1.1 shows a cartoon that explains the block time steps method. Stars in lower blocks have a smaller time step compared to stars in higher blocks, and the time steps are all integer multiples of the minimum one so that it is straightforward to synchronize the blocks with each other. In Fig.1.1, red arrows show that the stars in block 0 are always updated (time step dt), while the stars in block 1 are updated every two steps ($2dt$) and the stars in block 2 every four steps ($4dt$). Each star can jump from its block to another one, accordingly to its time step variation. The jump is possible only when the current block of the star is synchronized with the block where the star aims to jump. In Fig.1.1, these jumps are shown as green lines. It is worth noting that jumps toward lower blocks are always permitted (by construction, a block is always synchronized with its lower blocks); in contrast, stars can jump toward higher blocks only at synchronous times (forbidden transitions are shown as yellow arrows that end up on red crosses).

1.3 The Ahmad-Cohen neighbours scheme

Star clusters may contain a large number of stars (up to $\sim 10^6$ for massive GCs), and this makes direct N-body simulations very challenging because of the high computational complexity of $O(N^2)$. In order to deal with such numerical challenges, a neighbour procedure that requires fewer total force calculations can be introduced. A very common strategy for direct N-body simulations is to adopt the Ahmad-Cohen (AC) neighbour scheme [5, 17]. In this method, the force on a star is split into two contributions that evolve over different time scales. The first contribution depends only on near stars and the other one is owed to distant stars. It is apparent that the latter

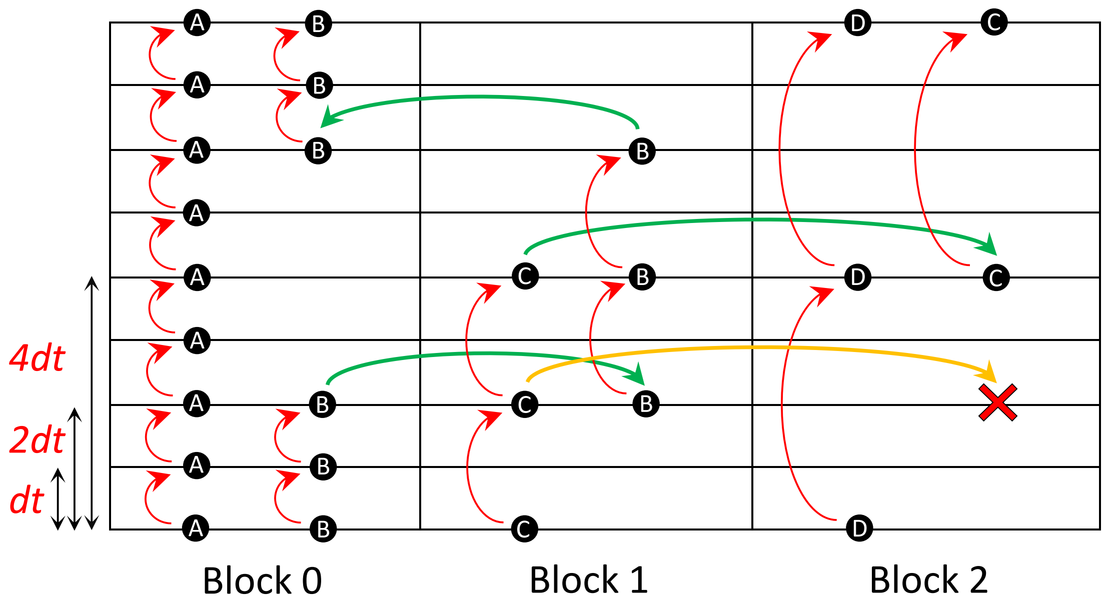

Figure 1.1: A cartoon of the block time steps method with four particles (indicated as "A", "B", "C" and "D") as an example. Horizontally, three example blocks are placed, with three time steps: dt, $2dt$, $4dt$. The vertical axis represents the discretized time. The red arrows represent a particle doing its time step. The green arrows represent a particle changing time step, and therefore block. The yellow arrows represent a particle that is trying to change time step, but it fails because the arriving block is not synchronized with the its actual block. At the bottom of the cartoon, particles "A" and "B" are in block 0, "C" is in block 1 and "D" is in block 2. Particles "A" and "D" do not change block, "A" keeps doing time steps dt (fast evolution) while "D" $4dt$ (slow evolution). "B" changes block after two steps, then it switch back to the previous block. "C" tries to change time step when blocks 1 and 2 were not synchronized, failing; later it was able to successfully change it when the two block were synchronized.

force evolves over much slower time scales than the one from close stars, therefore it does not have to be recalculated with the same frequency. This implies fewer operations per step and a significant performance gain. The combination with the block time steps method described above is straightforward: each star will be in two separate blocks, corresponding to two (discretized) time steps Δt_{close} and Δt_{far}, calculated for close and distant stars, respectively. For each star, we only compute the forces from its neighbours, proceeding with a time step Δt_{close} (neighbour steps), and the forces from distant particles are approximated using a Taylor expansion; in contrast, every k steps (with $k\Delta t_{close} = \Delta t_{far}$), we compute the force from both close and distant particles self-consistently. ISTEDDAS implements the AC neighbours scheme entirely on the GPU. This is a significant advantage in terms of performance since, for each step, there is no need to communicate the neighbours between the CPU and the GPU, reducing communication latencies.

To identify neighbours, we use the external C++ library ANN (to see why we have chosen this library see the subsection 1.4.3) that builds a three-dimensional k-d tree structure using an approximate nearest neighbor searching algorithm. Actually, there is the possibility to use a fixed number or a variable number of neighbours.

Let us first discuss the case with a variable number of neighbours. To calculate the ideal number of neighbours, we evaluate the total number of operations performed with and without neighbours within a time step Δt_f:

$$\text{with neighbours:} \qquad N_{operations} = n_n \frac{\Delta t_f}{\Delta t_c} + (N - n_n)\,,$$

$$\text{without neighbours:} \qquad N_{operations} = \frac{\Delta t_f}{\Delta t_c} N\,,$$

dividing the above contribution we obtain the cost function:

$$Cost(n_n) = \frac{n_n}{N} + \frac{\Delta t_c}{\Delta t_f}\left(1 - \frac{n_n}{N}\right), \tag{1.27}$$

where n_n is the number of neighbours. The ideal number of neighbours is retrieved from the above equation in the minimum of the cost function. To compute both Δt_c and Δt_f we use the Hermite 4th order time step criterion [17]:

$$\Delta t = \eta_\ell \sqrt{\frac{|\boldsymbol{a}_1||\ddot{\boldsymbol{a}}_1| + |\dot{\boldsymbol{a}}_1|^2}{|\dot{\boldsymbol{a}}_1||\dddot{\boldsymbol{a}}_1| + |\ddot{\boldsymbol{a}}_1|^2}}\,, \tag{1.28}$$

where η_ℓ is a free parameter that we take as 0.05 for Δt_c and 0.1 for Δt_f. In Eq. 1.28, the time step depends on n_n through the accelerations. The cost function is computed for different values of n_n until the maximum value, that is [18]:

$$n_{n,max} = \frac{1}{4}\left(\frac{N}{4}\right)^{\frac{3}{4}}\,. \tag{1.29}$$

The ideal value for n_n is the one that minimizes the cost function. For each star, we save the radius of its neighbour sphere, r_{neigh}, which corresponds to the distance of the star's n_n-th neighbour. To stabilize the number of neighbors per star, that is to avoid too frequent neighbors' changes, we have introduced a threshold value, so if the minimum cost is bigger than 0.6 no neighbours are selected ($r_{neigh} = 0$). The three-dimensional k-d tree structure and the number of neighbours are updated every time all stars are synchronized (every Δt_{max}), therefore every time the code computes N^2 forces. However, r_{neigh} can change before one global synchronization and the next one. After the update of the tree structure, we calculate the local star density considering, for each star, the distance to its 6th nearest neighbor, and we use it to estimate the system's radial density profile, $\rho\,(r)$. Every Δt_f, we update r_{neigh} so the number of neighbours per star does not change significantly, that is:

$$r_{neigh,new} = r_{neigh,old}\left(\frac{\rho\,(r_{*,old})}{\rho\,(r_{*,new})}\right)^{\frac{1}{3}}\,, \tag{1.30}$$

where $r_{*,old}$ and $r_{*,new}$ are the position of the star in the system at the previous and current time-step respectively.

Let us now consider the case with a fixed number of neighbours. The more natural choice is to take as a fixed number of neighbours a multiple of 32 ($n_n \propto 32$). In this way, the force calculation on GPU during the neighbour step is more efficient since 32 is the number of threads in a single GPU warp. In this case, we do not use r_{neigh}, therefore, to always have the right neighbours, we compute the three-dimensional k-d tree structure every time that at least one star is doing a distant step. This last case, for ISTEDDAS, is much more stable than the configuration with a variable number of neighbours: the numerical errors are accumulated much slower with respect to the case with variable neighbours due to the decreased number of operations made, and the block distribution is more stable overall. Moreover, since it does much less work on the CPU, we get a reduction of the latencies between the CPU and GPU speeding up the simulation. Furthermore, using a fixed number of neighbors, there is no need to calculate and adjust the neighbor distance (see Eq. 1.30), which relies crucially on the assumption of spherical symmetry of the N-body system. This means that, with fixed neighbors, we can evolve even systems that are far from spherical symmetry (e.g., fractal clusters).

Moving forward, some complications arise when using this scheme combined with the Hermite integrator. With the AC method, during the neighbour integration steps (Δt_n), no issues are encountered, since the neighbours are constant. Though during a distant step (i.e. every Δt_f), the neighbours might have changed, thus the Hermite corrector is not self-consistent anymore because in Eqs.1.7,1.8 we are summing and subtracting accelerations and derivatives computed with potentially different neighbours, in fact, the Hermite 6th order corrector writes as:

$$\boldsymbol{v}_c = \boldsymbol{v}_{0,old} + (\boldsymbol{a}_{1,new} + \boldsymbol{a}_{0,old})\frac{\Delta t}{2} - (\dot{\boldsymbol{a}}_{1,new} - \dot{\boldsymbol{a}}_{0,old})\frac{\Delta t^2}{10} + (\ddot{\boldsymbol{a}}_{1,new} + \ddot{\boldsymbol{a}}_{0,old})\frac{\Delta t^3}{120}\,, \tag{1.31}$$

$$\boldsymbol{r}_c = \boldsymbol{r}_{0,old} + (\boldsymbol{v}_c + \boldsymbol{v}_{0,old})\frac{\Delta t}{2} - (\boldsymbol{a}_{1,new} - \boldsymbol{a}_{0,old})\frac{\Delta t^2}{10} + (\dot{\boldsymbol{a}}_{1,new} + \dot{\boldsymbol{a}}_{0,old})\frac{\Delta t^3}{120}\,, \tag{1.32}$$

where the variables labeled with *old* and *new* are computed using the neighbours of the previous and the actual step respectively. Therefore, we apply the following steps, as in [17] for the Hermite 4th order integrator, to solve the problem:

- Compute the acceleration and its derivatives due to neighbors based on the list of neighbours calculated in the previous step ($\boldsymbol{a}_{close,old}$).
- Apply the corrector calculated for the acceleration due to neighbors
- Compute the acceleration and its derivatives due to neighbors and distant stars based on the new list of neighbours calculated in this step ($\boldsymbol{a}_{close,new}$ and $\boldsymbol{a}_{far,new}$)
- Compute the acceleration and its derivatives due to distant particles based on the old neighbor list as $\boldsymbol{a}_{far,old} = \boldsymbol{a}_{far,new} + \boldsymbol{a}_{close,new} - \boldsymbol{a}_{close,old}$.
- Apply the corrector for the acceleration due to distant particles using $\boldsymbol{a}_{far,old}$ and its derivatives.

It is worth noting that $\boldsymbol{a}$, $\dot{\boldsymbol{a}}$ and $\ddot{\boldsymbol{a}}$ are computed with the new neighbours list, during the evaluation step, while $\dddot{\boldsymbol{a}}$ (far and close) is estimated during the corrector step using the old neighbours list. Since in the Hermite 6th order scheme $\dddot{\boldsymbol{a}}$ is needed in the predictor step and to calculate the correct individual time steps for stars, it is important to compute it self-consistently. To achieve this, we keep track of the incoming and outgoing neighbors with respect to r_{neigh}, and we retrieve $\dddot{\boldsymbol{a}}_{new}$ by subtracting and summing on $\dddot{\boldsymbol{a}}_{old}$ the contribution from outgoing and incoming neighbours:

$$\dddot{\boldsymbol{a}}_{new} = \dddot{\boldsymbol{a}}_{old} - \dddot{\boldsymbol{a}}_{outgoing} + \dddot{\boldsymbol{a}}_{incoming}\,. \tag{1.33}$$

We compute the outgoing and the incoming contributions using the explicit Newtonian formula for $\dddot{\boldsymbol{a}}$:

$$\begin{aligned}
\dddot{\boldsymbol{a}}_{i,1} = \sum_{k=1}^{S} \frac{m_k}{|\boldsymbol{r}_{ik,p}|^3} \Bigg\{ & \dot{\boldsymbol{a}}_{ik,p} - 9\frac{\boldsymbol{r}_{ik,p}\cdot\boldsymbol{v}_{ik,p}}{|\boldsymbol{r}_{ik,p}|^2}\boldsymbol{a}_{ik,p} + \\
& + \frac{9}{|\boldsymbol{r}_{ik,p}|^2}\left[5\frac{(\boldsymbol{r}_{ik,p}\cdot\boldsymbol{v}_{ik,p})^2}{|\boldsymbol{r}_{ik,p}|^2} - \boldsymbol{r}_{ik,p}\cdot\boldsymbol{a}_{ik,p} - |\boldsymbol{v}_{ik,p}|^2\right]\boldsymbol{v}_{ik,p} + \\
& + \frac{1}{|\boldsymbol{r}_{ik,p}|^2}\Bigg[24\frac{(\boldsymbol{r}_{ik,p}\cdot\boldsymbol{v}_{ik,p})(\boldsymbol{r}_{ik,p}\cdot\boldsymbol{a}_{ik,p} + |\boldsymbol{v}_{ik,p}|^2)}{|\boldsymbol{r}_{ik,p}|^2} - 21\frac{\boldsymbol{r}_{ik,p}\cdot\boldsymbol{v}_{ik,p}}{|\boldsymbol{r}_{ik,p}|^2}\Bigg(5\frac{(\boldsymbol{r}_{ik,p}\cdot\boldsymbol{v}_{ik,p})^2}{|\boldsymbol{r}_{ik,p}|^2} + \\
& \qquad - \boldsymbol{r}_{ik,p}\cdot\boldsymbol{a}_{ik,p} - |\boldsymbol{v}_{ik,p}|^2\Bigg) - 3(\boldsymbol{r}_{ik,p}\cdot\dot{\boldsymbol{a}}_{ik,p} + 3\boldsymbol{a}_{ik,p}\cdot\boldsymbol{v}_{ik,p})\Bigg]\boldsymbol{r}_{ik,p}\Bigg\}\,,
\end{aligned} \tag{1.34}$$

where, as in Eqs.(1.4,1.5,1.6), the quantities with subscript "ik,p" are the relative predicted values between stars i and k (e.g. $\boldsymbol{r}_{ik,p} = \boldsymbol{r}_{k,p} - \boldsymbol{r}_{i,p}$), S is the total number of incoming or outgoing neighbours, and we have assumed the gravitational constant $G = 1$.

Finally, to estimate Δt_{close} and Δt_{far} we use Eq.1.9 only if the neighbours do not change; otherwise, we use the low order, Hermite 4th order, criterion as in Eq.1.28. This is necessary because $\boldsymbol{a}^{(4)}$ and $\boldsymbol{a}^{(5)}$ would be inconsistent if the neighbor list changes. Furthermore, $\boldsymbol{a}^{(4)}$ and $\boldsymbol{a}^{(5)}$ are not needed in the predictor step, and correcting them with the corresponding Newtonian formulas, as we do for $\dddot{\boldsymbol{a}}$, would have a significant computational cost without increasing the order of the integrator.

1.4 Optimizations and parallelization

ISTEDDAS runs almost completely on GPU accelerators. Therefore we were able to minimize the amount of data communication between the CPU and the GPU.

ISTEDDAS can run on single or multiple GPUs, and on single or multiple MPI nodes, in this way, the code can exploit GPU workstations as well as GPU clusters. One problem that is important to address is related to the evaluation kernel. The calculation of the newtonian gravitational force involves the computation of the inverse square root:

$$\boldsymbol{F}_{ij} = \frac{m_i m_j G}{|\boldsymbol{r}_{i,j}|^3}\boldsymbol{r}_{i,j} = \left(\frac{1}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}}\right)^3 m_i m_j G \boldsymbol{r}_{i,j} \,, \tag{1.35}$$

that is famous to be a computationally heavy operation, in fact, in the early 1990s the fast inverse square root was invented to overcome this problem. The algorithm is, as one can guess by the name, a faster implementation of the inverse square root that approximates the exact value of $1/\sqrt{x}$. We cannot use such an algorithm since we need as much accuracy as we can. Thankfully, nowadays the algorithms to compute the inverse square root are quite fast, therefore we use the CUDA function `rsqrt()`, which is a native reciprocal square root instruction. However, to speed up the code as much as possible, we try to optimize every operation. For example, whenever we can, we use the fused multiply-addition (`fma()`), which is a floating-point operation $(a \cdot b + c)$ performed in one step with a single rounding error. It can be used to compute the square module of a vector $\boldsymbol{v}$:

$$|\boldsymbol{v}|^2 = v_x \cdot v_x + v_y \cdot v_y + v_z \cdot v_z = \texttt{fma}\left(v_x, v_x, \texttt{fma}\left(v_y, v_y, \texttt{fma}\left(v_z, v_z, 0\right)\right)\right) \,, \tag{1.36}$$

where v_x, v_y, v_z are the 3D components of the vector $\boldsymbol{v}$. In this simple example, we pass from 5 floating-point operations (3 multiplications and 2 summations) to just 3 floating-point operations. It can also be used to compute the Taylor expansions of any quantity q, for example, for a third-order expansion:

$$\begin{aligned} q_1 &\simeq q_0 + \dot{q}_0 dt + \ddot{q}_0 \frac{dt^2}{2} + \dddot{q}_0 \frac{dt^3}{6} = \\ &= \texttt{fma}\left(dt, \texttt{fma}\left(\frac{dt}{2}, \texttt{fma}\left(\frac{dt}{3}, \dddot{q}_0, \ddot{q}_0\right), \dot{q}_0\right), q_0\right) \,. \end{aligned} \tag{1.37}$$

In this simple example, we pass from 6 floating-point operations (3 multiplications and 3 summations) to just 3 floating-point operations.

1.4.1 Forces evaluation algorithm

Computing $\boldsymbol{a}_{i,1}$, $\dot{\boldsymbol{a}}_{i,1}$, $\ddot{\boldsymbol{a}}_{i,1}$, and $\dddot{\boldsymbol{a}}_{i,1}$ can be extremely computationally expensive due to the huge amount of multiplications, divisions, summations, and inverse square roots (see Eqs.1.4,1.5,1.6,1.34). However, a lot of these calculations are repeated, therefore, for each couple (i, j) of stars, we define the following quantities to avoid these repetitions:

$$S_0 = \boldsymbol{r}_{ij,p} \cdot \boldsymbol{r}_{ij,p} \,, \tag{1.38}$$

$$S_1 = \boldsymbol{r}_{ij,p} \cdot \boldsymbol{v}_{ij,p} \,, \tag{1.39}$$

$$S_2 = \boldsymbol{r}_{ij,p} \cdot \boldsymbol{a}_{ij,p} + \boldsymbol{v}_{ij,p} \cdot \boldsymbol{v}_{ij,p} \,, \tag{1.40}$$

$$S_3 = \boldsymbol{r}_{ij,p} \cdot \dot{\boldsymbol{a}}_{ij,p} + 3\boldsymbol{v}_{ij,p} \cdot \boldsymbol{a}_{ij,p} \,, \tag{1.41}$$

$$Q_1 = -3r_{inv}^2 S_1 \,, \tag{1.42}$$

$$Q_2 = -r_{inv}^2 \left(5S_1Q_1 + 3S_2\right) \,, \tag{1.43}$$

$$Q_3 = -r_{inv}^2 \left(8S_2Q_1 + 7S_1Q_2 + 3S_3\right) \,, \tag{1.44}$$

where, again, the quantities with subscript "ij, p" are the relative predicted values between stars i and j (e.g. $\boldsymbol{r}_{ij,p} = \boldsymbol{r}_{j,p} - \boldsymbol{r}_{i,p}$), and $r_{inv} = 1/\sqrt{S_0}$. Using the above quantities we can rewrite the equations for the acceleration and its derivatives as:

$$\boldsymbol{a}_{i,1} = \sum_{j \neq i}^{N} m_j r_{inv}^3 \boldsymbol{r}_{ij,p} \,, \tag{1.45}$$

$$\dot{\boldsymbol{a}}_{i,1} = \sum_{j \neq i}^{N} m_j r_{inv}^3 \left(Q_1 \boldsymbol{r}_{ij,p} + \boldsymbol{v}_{ij,p} \right) \,, \tag{1.46}$$

$$\ddot{\boldsymbol{a}}_{i,1} = \sum_{j \neq i}^{N} m_j r_{inv}^3 \left(Q_2 \boldsymbol{r}_{ij,p} + 2Q_1 \boldsymbol{v}_{ij,p} + \boldsymbol{a}_{ij,p} \right) \,, \tag{1.47}$$

$$\dddot{\boldsymbol{a}}_{i,1} = \sum_{j \neq i}^{N} m_j r_{inv}^3 \left(Q_3 \boldsymbol{r}_{ij,p} + 3Q_2 \boldsymbol{v}_{ij,p} + 3Q_1 \boldsymbol{a}_{ij,p} + \dot{\boldsymbol{a}}_{ij,p} \right) \,. \tag{1.48}$$

By computing this equation we avoid a lot of extra calculations, even the inverse square root is done just once. On top of this, we also use the tricks explained above (e.g. the `fma()` function).

1.4.2 Forces reduction and gather

When simulating star clusters with more than 10^4-10^5 stars, it is very useful to use multiple GPUs to speed up the calculation of the forces. The predictor and corrector kernels are computed equally on each GPU since they scale with the number of stars (N), and therefore are light kernels from the computational point of view. Thus they do not need any kind of communication between GPUs, however, we do transfer the results of GPU 0 to all the other GPUs every time that all the stars are synchronized. In this way we avoid the arise of discrepancies in the quantities among the GPUs.

Anyway, the force evaluation is a computationally heavy kernel, it scales at maximum as N^2 (distant step for N stars) and at minimum as $N \cdot n_n$ (neighbours step for N stars). Thus the evaluation process is split among the available GPUs to decrease the computational load on each GPU, this is done for both the far and the close accelerations. This split of the forces evaluation inevitably introduces extra communication between GPUs and CPUs at each step. The two cases, distant and neighbours steps, are handled separately with two different GPU kernels, and also different communication strategies. For the distant step, every GPU computes the interactions between all the i stars (N) with only a fraction of the j stars (N/n_{gpu}), in this way they will perform $\sim N^2/n_{gpu}$ operations, that is a perfectly balanced use of the devices. In more detail, every star has its own one-dimensional block with 128 threads (4 warps), so each thread will compute and accumulate the forces from $(N/n_{gpu})/128$ j stars. Since we have divided the j stars, at the end a reduction will be necessary to build the total forces between stars. Instead, for the neighbours step, since n_n is much smaller than N or N/n_{gpu} would not make sense to divide the neighbours among the GPUs (n_n/n_{gpu}), because each i star would have to interact with a very small amount of j star on each device, that is not optimal, especially if that number is smaller then 32. Thus, every GPU computes the interactions between a fraction of the i stars (N/n_{gpu}) with all their neighbours, or j stars, (n_n), in this way they will perform $\sim N \cdot n_n/n_{gpu}$ operations. In more detail, every block is, again, one-dimensional and has 128 threads, but this time it will handle 4 i stars (one for each warp). So, each thread of the warp will compute and accumulate the forces from $n_n/32$ j stars, and this is the reason why we choose $n_n \propto 32$ for the value of the number of neighbours. Since this time we have divided the i stars, at the end a gather will be necessary to communicate the total forces between stars to all the GPUs.

For the reduction we apply the following strategy: first of all, there is the single GPU reduction, which is done at warp level, to be as efficient as possible, using the CUDA function `__shfl_down_sync()`, see Fig.1.2[2]. Then, since in the evaluation

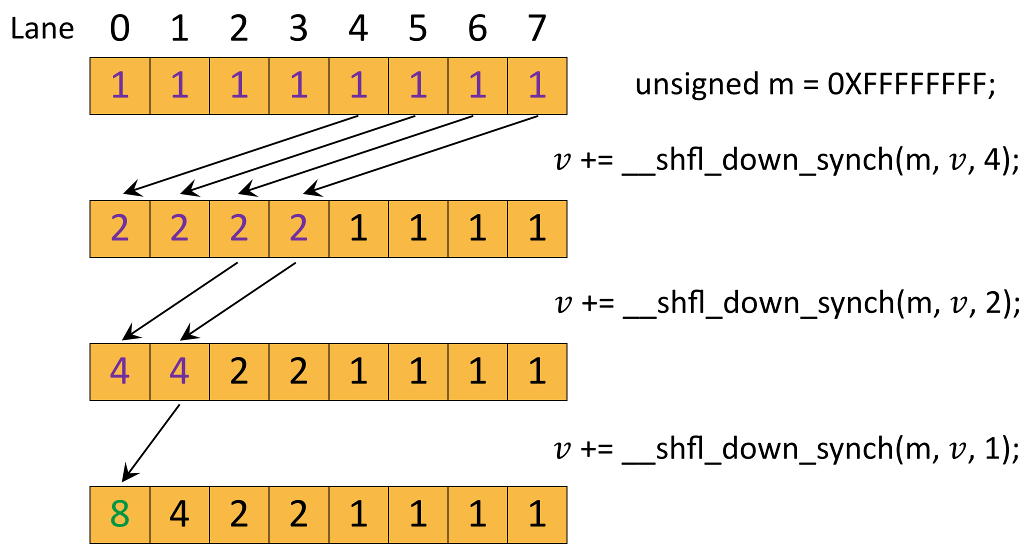

Figure 1.2: The cartoon shows an example of using warp-level primitives. It uses `__shfl_down_sync()` to perform a tree-reduction to compute the sum of the v variable held by each thread in a warp. At the end of the loop, v of the first thread in the warp contains the desired sum (8 in this example). $m =$ `0XFFFFFFFF` is the standard mask used to perform this operation, and it is an hexadecimal integer constant where the prefix `0X` means the next number is written in the hexadecimal, and `FFFFFFFF` is $2^{32} - 1$ in decimal notation, or 32 consecutive "1" in binary notation. A warp comprises 32 lanes, with each thread occupying one lane, so it needs 5 uses of the function to be reduced. The data exchange is performed between registers, and it is more efficient than going through shared memory, which requires a load, a store, and an extra register to hold the address.

kernel each block handles an i star, we reduce all the values of the first threads of each warp of each block. After these passages, every GPU has the reduced forces on

[2]See the link https://developer.nvidia.com/blog/using-cuda-warp-level-primitives/ for extra details about warp level reductions.

each i star from N/n_{gpu} stars. To obtain the total force from all the N stars of the simulation there are two final steps: the computed forces are copied to the CPUs that perform an intra-node reduction between the GPUs within each node and in the end an MPI reduction (`MPI_Allreduce()`) is performed between the nodes to finally have the complete far and close forces for each star. For the gather we use a similar strategy as above: first of all, there is the single GPU reduction, which is done at warp level exactly as before. Since every warp handles the interaction of each star with its neighbours, this is the only reduction required. In fact, after this step, we already pass to the intra-node gather between the GPUs within each node, and in the end, we perform an MPI gather (`MPI_Allgatherv()`) between the nodes to finally have the complete neighbours forces for each star on all the GPUs.

1.4.3 Ahmad-Cohen tree building optimization

In order to identify neighbours, we needed a nearest-neighbour searching tree library, therefore we test 5 different codes to decide which one best match our needs ISTEDDAS:

- ANN[3] (Approximate Nearest Neighbor searching): a library written in C++, running on CPU, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrarily high dimensions.
- FLANN[4] (Fast Library for Approximate Nearest Neighbors): a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It is written in C++ and runs on GPU [22].
- FAISS[5] (Facebook AI Similarity Search): a library for efficient similarity search and clustering of dense vectors. It contains algorithms that search in sets of vectors of any size. It is written in C++ and runs on GPU [14].
- hnswlib[6] (Hierarchical Navigable Small World graphs library): a library written in C++ that runs on CPU, for the approximate K-nearest neighbor search it uses navigable small world graphs with controllable hierarchy [19].
- GGNN[7] (Graph-based GPU Nearest Neighbor search): its search structure is based on nearest neighbor graphs and information propagation on graphs. the code is designed to take advantage of GPU architectures to accelerate the hierarchical building of the index structure and for performing the query [12].

These libraries have two main operations: the construction of the tree, and the search for neighbours. The first operation is the creation of the tree structure from which the search for neighbours is done. We benchmark these codes based on the usage

[3]The code is public and available here: http://www.cs.umd.edu/∼mount/ANN/
[4]The code is public and available here: https://github.com/flann-lib/flann
[5]The code is public and available here: https://github.com/facebookresearch/faiss
[6]The code is public and available here: https://github.com/nmslib/hnswlib
[7]The code is public and available here: https://github.com/chingyaoc/ggnn.pytorch

that they would have in ISTEDDAS, therefore we use a three-dimensional space and we test a number of points that could resemble the number of stars in star clusters $N = [2^{11}, 2^{13}, 2^{14}, 2^{15}, 2^{16}, 2^{17}]$. For each code, and for each number of stars, I save both the time needed to build the tree structure and the searches (or queries) of the first 64 neighbours on N_q stars. For the benchmarks, N_q is chosen to be an integer fraction of N: $[1, N/128, N/32, N/8, N/2, N]$. The benchmarks in Fig.(1.3,1.4) were done on a laptop with the specifics in Table 1.1.

CPU	:	Intel(R) Core(TM) i7-10750H 2.60GHz
RAM	:	16GB DDR4
GPU	:	NVIDIA GeForce GTX 1650 Ti
GRAM	:	4GB GDDR6

Table 1.1: Laptop specifics.

Fig.1.3 shows the scalability of each library with respect to the increasing number of points and the increasing number of queries, moreover, it is also shown the time comparison for the creation of the tree. Instead, in Fig.1.4, the same data are rearranged to show the time comparison for the neighbours searches among the 5 libraries. From the point of view of building the tree, the two best options are ANN and FLANN, ANN performs much better with smaller N, and FLANN scales better with increasing N. In contrast, from the point of view of the searches, hnswlib perform better than anyone else with $N_q = 1$ for every N, however increasing N and N_q, the two best options for ISTEDDAS are, again, ANN and FLANN. Probably, the reason behind these results is that the other libraries are optimized for high-dimensional spaces, while we just need a three-dimensional tree. For ISTEDDAS, on a single GPU, the best possible combination would be to use ANN with a small number of stars and FLANN with a large number of stars, with the separation around 2^{14}-2^{15} stars. However, because ANN does not scale on multiple nodes and can use only one CPU (see Section 4.1), while FLANN can scale on multiple GPUs, it will be crucial to take advantage of FLANN inside ISTEDDAS in order to achieve good scaling over tens or hundreds of GPUs on the next-generation supercomputers. Since ANN was already implemented inside the code and the difference between ANN and FLANN is not significantly high on a single GPU, we stick with it, leaving the implementation of FLANN in ISTEDDAS for future work. The library ANN is implemented in ISTEDDAS, and it was modified to run the search for neighbours in parallel with OpenMP. This optimization speeds up the ANN library on a single node, however it still cannot scale with multiple nodes.

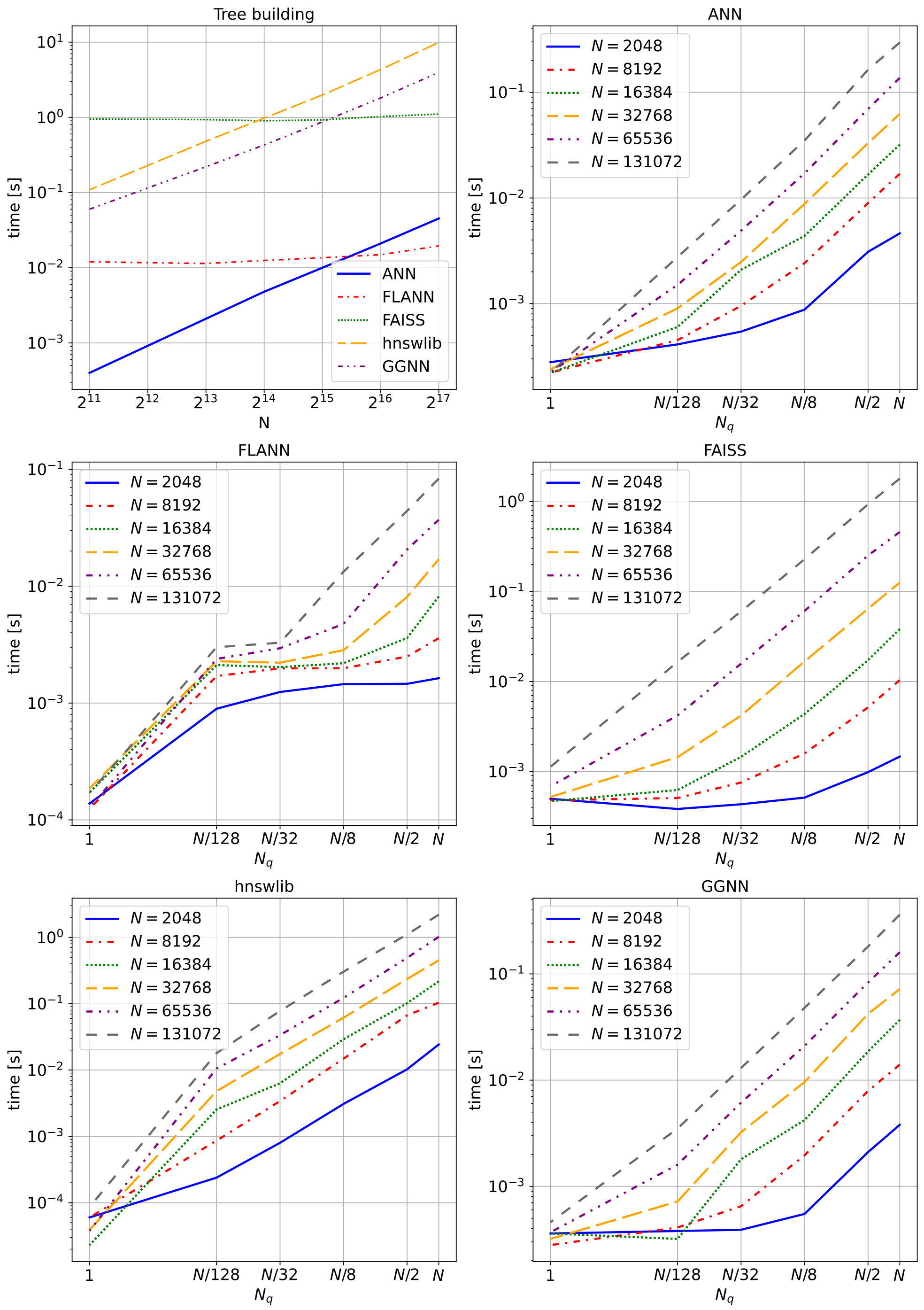

Figure 1.3: Benchmarks of the 5 tested libraries. In the top-left panel it is shown the time taken by each library to build the 3D tree structure in function of the total number of stars. In the other 5 panels it is shown, for each library, the time taken to perform a search (or query) of the first 64 neighbours on N_q points. N_q is a fraction of the total number of stars N: $[1, N/128, N/32, N/8, N/2, N]$. N is chosen as a power of 2: $[2^{11}, 2^{13}, 2^{14}, 2^{15}, 2^{16}, 2^{17}]$.

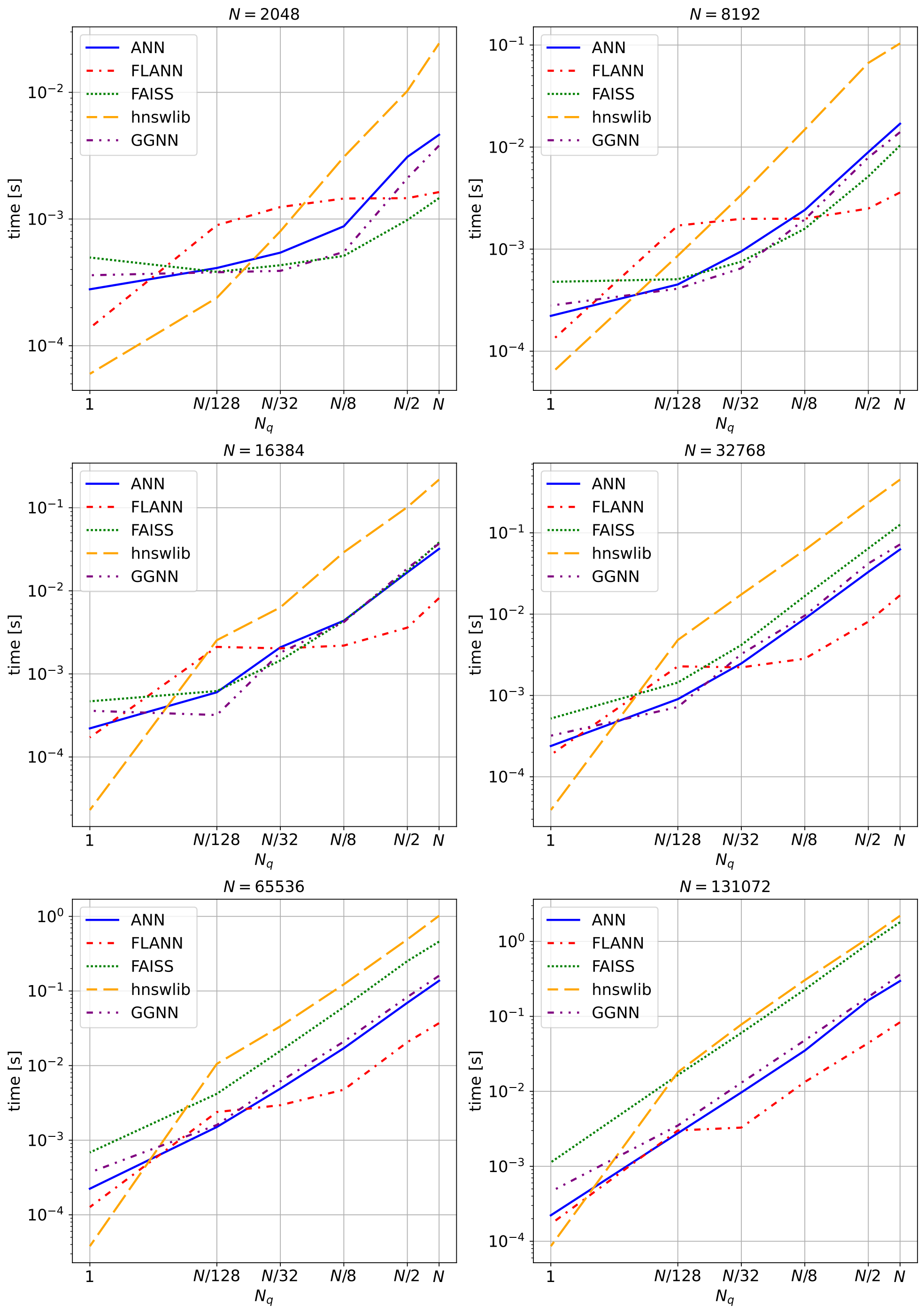

Figure 1.4: Benchmarks of the 5 tested libraries. In each panel it is shown, for a certain total number of stars N, the time taken by each library to perform a search (or query) of the first 64 neighbours on N_q points (stars). N_q is a fraction of the total number of stars N: $[1, N/128, N/32, N/8, N/2, N]$. N is chosen as a power of 2: $[2^{11}, 2^{13}, 2^{14}, 2^{15}, 2^{16}, 2^{17}]$.

Chapter 2

The few body code TSUNAMI

While the 6th order Hermite integrator is a powerful main integrator in ISTEDDAS, it might not be suitable for the long-term integration of stable, very tight star systems (e.g. binaries, triples). To integrate a tight binary, the Hermite integrator would significantly reduce the time step time, making the numerical integration of the whole system stall, and accumulating numerical error very fast. Thus, implementing a fast, high-accuracy algorithm to take care of tight systems in ISTEDDAS is crucial, especially if our aim is to investigate the evolution of merging compact-object binaries and their GW emission. Therefore, we have coupled ISTEDDAS with TSUNAMI, a code for the high-accuracy numerical integration of binaries and hierarchical systems [27–29]. TSUNAMI is based on the following techniques: regularization of the equations of motion, chain coordinates to reduce round-off errors, and Bulirsch–Stoer extrapolation. The first technique (regularization) takes care of the singularity of the gravitational potential (when the distance between the two bodies is near zero). The second technique (chain coordinates) helps reduce the round-off errors in hierarchical systems, which arise when moving to center-of-mass coordinates, without introducing numerically expensive techniques of compensated summation. The third method (Bulirsch–Stoer extrapolation) increases the accuracy of the integration and makes it adaptable over a wide dynamical range. Furthermore, TSUNAMI includes additional features such as perturbative forces, general relativity corrections through post-Newtonian terms up to 3.5 order, tidal interactions, and a collision-detection algorithm. In this chapter, I will go into further detail on these techniques.

2.1 Coupling ISTEDDAS and TSUNAMI

TSUNAMI is completely re-implemented inside ISTEDDAS as a module of it and it runs completely on the CPU. The crucial part of the coupling is the "decision making", that is the totality of the effective criteria to decide whether a group of stars must be regularized with TSUNAMI, or it is better to leave it within ISTEDDAS. This is important because TSUNAMI runs entirely on CPU code while ISTEDDAS is designed to run entirely on GPU. Thus, switching-on regularization would imply significant CPU-GPU data transfer, which might become a bottleneck if the decision-making is not carefully implemented.

2.1.1 Decision Making

In ISTEDDAS the decision making is similar to that described in [3, 15], but with some crucial differences. Finding the binary systems in an N-body system is a slow procedure with time complexity $O(N^2)$, but using the already built k-d tree from the Ahmad-Cohen scheme (see Section 1.3) we have it for free since we already know the nearest neighbours of every star in the system. There are two possible systems for which the code decides to integrate a group of stars using TSUNAMI: binary systems or close encounters. Before entering into details, let us define the important length scale to check the proximity of two stars: the close encounter radius, defined as:

$$R_{cl} = 2\frac{\bar{m}}{\sigma^2}\eta_r \,, \tag{2.1}$$

where $\bar{m}$ and σ^2 are the average mass and the velocity dispersion, and η_r is a free parameter used to conveniently adjust R_{cl}. We do not consider the whole N-body system to compute these quantities, but only the nearest $n_{n,max}$ stars (see eq. 1.29), in this way, we are computing the local R_{cl} for each star. In addition, To understand whether two stars are a bound (soft or hard binary) or unbound (close encounter) system we use the total energy, that is:

$$E_{tot} = U + K = -\frac{m_i m_j}{|\boldsymbol{r}_{ij}|} + \frac{1}{2}m_i|\boldsymbol{v}_i - \boldsymbol{v}_{com}|^2 + \frac{1}{2}m_j|\boldsymbol{v}_j - \boldsymbol{v}_{com}|^2 \,, \tag{2.2}$$

where U and K are the gravitational potentials and kinetic energy of the two stars, $m_{i,j}$ and $v_{i,j}$ are the masses and the velocities, $v_{com} = (m_i v_i + m_j v_j)/(m_i + m_j)$ is the velocity of the two stars' center of mass, and $|\boldsymbol{r}_{ij}|$ is the distance between the two stars. The sign of the total energy carries an important physical meaning, a negative E_{tot} is associated with a bound gravitational system, and a positive E_{tot} is associated with an unbound gravitational system. therefore, comparing E_{tot} with the average kinetic energy of the N-body system ($\bar{K} > 0$) we understand the type of bound between the two considered stars:

$$\begin{aligned}
E_{tot}/\bar{K} &\leq -1 && \text{bound system (hard binary)}\,,\\
-1 < E_{tot}/\bar{K} &< 0 && \text{bound system (soft binary)}\,,\\
0 \leq E_{tot}/\bar{K} & && \text{unbound system (close encounter)}\,,
\end{aligned}$$

Putting everything together, we have a binary system if two stars are separated by a distance lesser than R_{cl} and they are a hard binary from the energy point of view, while we have a close encounter if two stars are separated by a distance lesser than R_{cl}, they are an unbound system and they are pointing to each other (the dot product between their relative position and velocity is negative). I can finally summarize the conditions as:

$$\begin{aligned}
&\text{Binary system:} && |\boldsymbol{r}_{ij}| < R_{cl} \quad \& \quad E_b/\bar{K} \leq 1\,,\\
&\text{Close encounter:} && |\boldsymbol{r}_{ij}| < R_{cl} \quad \& \quad E_b/\bar{K} \geq 0 \quad \& \quad \boldsymbol{r}_{ij}\cdot\boldsymbol{v}_{ij} < 0\,,
\end{aligned}$$

where the vectors $\boldsymbol{r}_{ij}$ and $\boldsymbol{v}_{ij}$ are the relative position and velocity of the two stars, and the symbol & is the logic "and".

Once the code has found two stars for TSUNAMI it also checks if the next nearest star wants to be part of the system. For close encounters, the code simply checks the distance between the extra star and the center of mass of the two stars already in TSUNAMI:

$$|\boldsymbol{r}_{ij}| < \frac{2}{3} R_{cl} \,.$$

In this case, we would have a three-body close encounter, however, to add other stars the check is done considering the center of mass of the most bound couple of stars in the system (called inner binary, the one with minimum E_b). For a binary system, we have the exact same check as above, however, there are other two conditions, based on orbital parameters, for stars to enter TSUNAMI and form a multiple-body system. First of all, the orbital eccentricity between the entering body and the harder binary, and the semi-major axis of the inner binary in the TSUNAMI system are defined as:

$$e = \frac{1}{\mu} \left| \left(|\boldsymbol{v}_{bk}| - \frac{\mu}{|\boldsymbol{r}_{bk}|} \right) \boldsymbol{r}_{bk} - (\boldsymbol{r}_{bk} \cdot \boldsymbol{v}_{bk})\, \boldsymbol{v}_{bk} \right| \,, \tag{2.3}$$

$$a = \frac{m_b}{2} \left(\frac{|\boldsymbol{v}_{ij}|^2}{2} - \frac{m_b}{|\boldsymbol{r}_{ij}|} \right)^{-1} \,, \tag{2.4}$$

where $m_b = m_i + m_j$, $\mu = m_b + m_k$, the quantities labeled with bj are relative to the center of mass of the inner binary (b) and the entering stars (k), and the ones labeled with ij are relative to the stars in the inner binary. The other two exclusive conditions to let a star enter the TSUNAMI integration are relative to the orbit configurations:

$$\begin{aligned} e > 1 (\text{open orbit case}) \quad & \& \quad |\boldsymbol{r}_{ij}| < R_{cl} \sqrt{0.5\, N \mu} \,, \\ e \leq 1 (\text{close orbit case}) \quad & \& \quad \gamma \geq \gamma_{crit} \,, \end{aligned}$$

where N is total number of stars in the N-body system, the symbol & is the logic "and", and γ is the perturbation parameter defined as:

$$\gamma = \frac{2\bar{m}}{m_b} \left(\frac{a}{|\boldsymbol{r}_{ij}|} \right)^3 \,, \tag{2.5}$$

$\gamma_{crit} = 2^{-6} = 0.015625$ is a fixed parameter, $\bar{m}$ is the average mass of the N-body system and m_b is the mass of the binary system.

At the end of this selection, we could have one or multiple systems of stars that need to be integrated with the ARChain algorithm, those systems will be called regularized systems in the rest of this thesis. Inside TSUNAMI the stars will be shifted in the center of mass reference frame, while in ISTEDDAS the center of mass of the regularized system takes the place of one of those stars and it is evolved with the Hermite integrator, the other stars become "ghost" particles with $m = 0$.

It is crucial to identify, and carefully take care of, another set of stars, that is the perturbers. They will have two roles: they will enter the TSUNAMI integration as differential external accelerations during the Leapfrog and will be checked to see if new stars need to be part of the regularized system. The perturbers are selected among the nearest bodies for each regularized system using the perturbation parameter defined above (eq. 2.5):

$$\gamma > \gamma_{pert} \,,$$

where $\gamma_{pert} = 10^{-6}$ is a fixed parameter.

The final part of the decision-making is the termination conditions, that is the checks that establish whether a star will continue to be integrated using the ARChain or will go back to the Hermite integrator. For binary systems there is a single condition to break the regularization:

$$|\boldsymbol{r}_{ij}| > 2\, R_{cl} \,.$$

In contrast, for multiple stars systems, a particle can exit the regularization if the following 3 conditions are satisfied:

$$!\,[\, e < 1 \quad \& \quad |\boldsymbol{r}_{bj}| < 10\, R_{cl}\,] \,, \tag{2.6}$$

$$\boldsymbol{r}_{bj} \cdot \boldsymbol{v}_{bj} > 0 \,, \tag{2.7}$$

$$\gamma < \gamma_{crit} \,, \tag{2.8}$$

where the symbol & is the logic "*and*", and the symbol ! is the logic negation. The first condition is important for the hierarchical system in which a third body has a very eccentric and elongated orbit, we want to avoid the body entering and exiting the regularization at each orbit (see Fig.4.7). The second condition is to ensure that the body is moving away from the harder binary, while the third condition is, again, a threshold on the perturbation parameter. After the termination, the exiting stars will regain their mass in ISTEDDAS and, if the regularized system survives, the center of mass coordinate is updated accordingly.

2.1.2 Time synchronization with the block time-step method

The block time-steps method (see Section 1.2) is a powerful addition to ISTEDDAS, as it significantly increase the performance of the code without losing accuracy. The centers of mass of the regularized systems are always considered without neighbours in ISTEDDAS ($r_{neigh} = 0$), therefore they have always single time-step (Δ_{far}). When the center of mass does a step in ISTEDDAS, the regularized system relative to the center of mass is integrated simultaneously in TSUNAMI on the same time-step. However, the TSUNAMI works with the regularized time coordinate, which is not quantized as the time step of the centre of mass and is related to physical time through the integral equation $ds = U dt$, where ds and dt are the regularized and physical time steps and U is the gravitational potential energy. Thus, synchronizing the integration time of the regularized system and that of the associated centre of mass results in potential time-synchronization errors. To fix this issue, I implemented the following strategy: the ARChain integrator stops right after the physical time step that overpasses the centre of mass step, and the initial quantities of this last step are used to re-initialize the integrator that will run again for the remaining amount of time using a normal Leapfrog ($ds = dt$). In this way, the last piece of time before the synchronization with

ISTEDDAS is done without regularization (the algorithm can not change the time-step), and the ARChain ends the integration exactly at the desired time. However, using this method results, sometimes, in a slightly wrong computation of the orbits in the final step, especially when the regularized system is very tight or very eccentric, and this, after a long integration, accumulates a lot of errors on the energy of the system. Therefore, I implement another strategy to solve the synchronization problem without losing precision: as before, the ARChain integrator stops right after the step that overpasses the desired time, and the initial quantities of this last step are used to re-initialize the integrator that will run again for the remaining amount of time but forcing the time-step to be roughly an order on magnitude smaller than the one computed by the integrator. If the desired final time is surpassed without reaching the desired precision, the previous step is retrieved and the timestep is decreased again. I keep adjusting these final time-steps to reach the convergence at the ISTEDDAS time-step within a certain small threshold. This strategy does not take more time with respect to the other described strategy and completely solves the synchronization problem without losing the accuracy of the ARChain integrator.

2.1.3 Parallelization

TSUNAMI runs on CPU and, in order to speed it up, I parallelize it using OpenMP, however, it is important to stress that also the ANN library is running in parallel using OpenMP. Thus, I use the OpenMP tasks: the entire ISTEDDAS loop is run by the master thread of OpenMP on each node, then, when the simulation needs a tree-search, or to integrate a system in TSUNAMI, a task is open by the master thread, that spawns the needed amount of threads to parallelize the operation. Using an OpenMP task is a good strategy since the master thread can go on in the simulation while the spawned threads keep working in parallel. Thus, even if both the two operations are needed, the two different tasks will spawn different threads to run them in parallel with each other. While the ANN library must do its search before the new computation of forces, and so it needs a barrier before the forces-evaluation kernel on GPU, TSUNAMI is free to run independently from ISTEDDAS. Because of this, the TSUNAMI threads do not need any barrier and can run asynchronously with the GPU for multiple ISTEDDAS steps until their next synchronization with the ISTEDDAS time-step, which can happen, borderline, after a single step of ISTEDDAS. Each thread of TSUNAMI will run one regularized systems, or multiple if they are too many to evolve. Since each system will have its own time-step in ISTEDDAS, the entire OpenMP task will synchronize with ISTEDDAS when the regularized system with the faster time-step will be synchronized with the ISTEDDAS time-step.

Chapter 3

The population-synthesis code SEVN

In an N-body code, the evolution of single and binary stars can be implemented following different approaches. A possible strategy is to use a stellar evolution code (e.g., PARSEC [7, 9–11, 23, 26]) to evolve stars every time that is needed, but this is a very slow approach because stellar evolution codes are very sophisticated and need to integrate the equations for the internal structure of stars, at each step. Another strategy is to use fitting formulas for all the stellar parameters. This approach is very fast but the generation of such formulas is very complicated and time-consuming since, each time a stellar prescription has to be updated, all the formulas must be re-derived from scratch. Therefore, it would be hard to keep up with the state-of-the-art stellar evolution prescriptions. In contrast, a look-up table-based population-synthesis code uses pre-evolved stellar evolution tracks that are provided in the form of look-up tables, for a grid of masses and metallicities, which are read and interpolated on the fly. This strategy has two crucial advantages: it is computationally very fast, and users can easily switch to a different set of look-up tables, thus investigating the impact of different stellar evolution prescriptions is possible without changing any lines in the code. ISTEDDAS is designed to be interfaced with the SEVN population-synthesis code. In this chapter, I will give an overview of the strategies adopted in our code to implement single and binary stellar evolutionary processes. Moreover, I will describe SEVN following [13].

3.1 Coupling ISTEDDAS and SEVN

SEVN can work as a stand-alone code (for fast population synthesis studies in the field) or can be linked to an N-body code, without having performance penalties. Our dynamical evolution code ISTEDDAS is coupled with the new population-synthesis code SEVN.

There are only three quantities that have to be communicated between the two codes, for each star: mass, radius, and the stellar evolution time-step. The radius is important to check whether two stars collide/merge or they are only passing close to each other. The new masses are needed by ISTEDDAS to correctly compute the new accelerations. The evolution time-step is needed to synchronize the evolution of the stars with the block time-steps method in ISTEDDAS.

An aspect that is worth mentioning is that SEVN lacks the evolutionary tables for low-mass stars (mass $\lesssim 2\mathrm{M}_\odot$). Such low-mass tracks will be included in an upcoming version of the SEVN code (Spera, M., private communication). Therefore, for now, in ISTEDDAS we select and evolve only the stars with a mass larger than a certain threshold (default at $3\mathrm{M}_\odot$, for safety). The other stars will be considered as non-evolving point-mass stars. This is an acceptable approximation considering that the ISTEDDAS code is still under development and for preliminary tests, we do not evolve star clusters for more than a few hundred of Myrs.

3.1.1 Single star evolution

The selected single stars are evolved by SEVN, which will adjust the evolutionary time-step according to its own algorithms, for multiple time steps until the mass has changed beyond a certain percentage threshold (by default we choose 2%). For each time step, the mass of the stars in ISTEDDAS is evolved through a linear interpolation between the initial and the final mass of the stars. This is done during the predictor step. To perform such interpolation on GPU we communicate two quantities: the angular coefficient (a_m) and the intercept (b_m) of the line ($m(t) = a_m t + b_m$), that is:

$$
\begin{aligned}
a_m &= \dot{m} = \frac{dm}{dt} = \frac{m(t_1) - m(t_0)}{t_1 - t_0}\,, \\
b_m &= m(t_0) - a_m t_0\,,
\end{aligned}
$$

where for t_0 and t_1 we consider the initial and final time of the total step performed by SEVN. In Fig.3.1 I show the evolution of an example star born with $\sim 61\mathrm{M}_\odot$ and $Z = 0.002\mathrm{Z}_\odot$ on the ZAMS. The comparison between SEVN and the SEVN module in ISTEDDAS are perfectly superposed, since they correctly use the same algorithms to evolve stars, while the interpolation made on GPU slightly differs from them, as expected. I also plotted the evolution of the core masses, the radius, and the luminosity, to show a complete picture of the star's evolution.

Using SEVN, the mass of stars is no longer a constant, it changes because of the various mechanisms involved in the stellar evolution, therefore we modified the equations for the derivatives of the acceleration used in ISTEDDAS (Eqs.1.5,1.6,1.34) to consider the variation of the star's masses. In this way, all the stars can feel the mass changes of other stars in the cluster by adjusting their dynamical time-step in ISTEDDAS, see Eq.1.9. Moreover, it is crucial to modify the derivatives of the acceleration to stabilize the dynamical integrator, in fact, without this correction, the high derivatives tend to accumulate errors until the simulation becomes unstable. The new acceleration of a single star (see Eq.1.4) is:

$$
\boldsymbol{a}_i = \sum_{j \neq i}^{N} \frac{m_j}{|\boldsymbol{r}_{ij}|^3}\boldsymbol{r}_{ij} = \sum_{j \neq i}^{N} m_j \boldsymbol{\mathcal{Q}}_{ij}\,, \tag{3.1}
$$

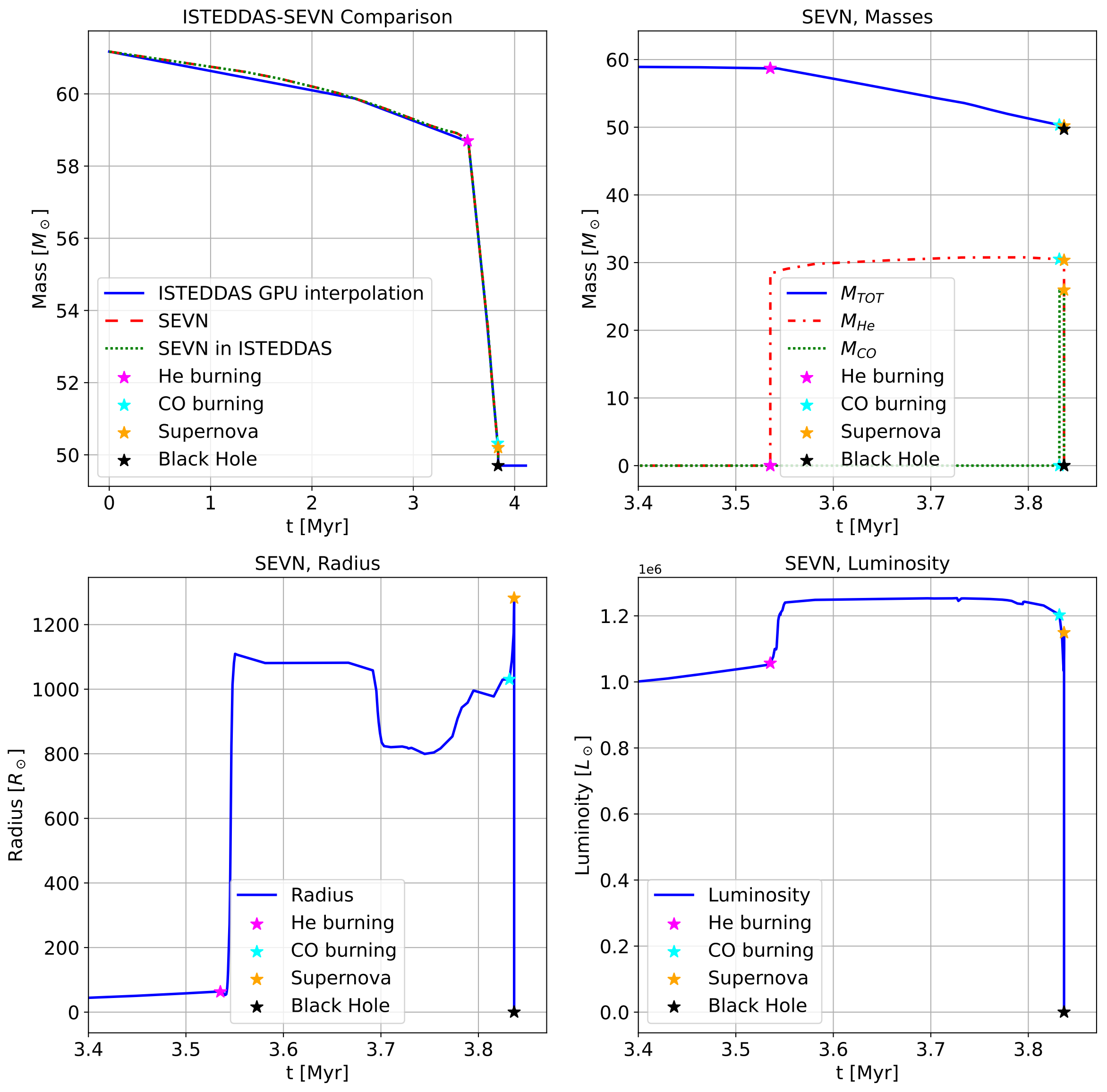

Figure 3.1: The plots in this figure are referred to the evolution of a star born with $\sim 61\mathrm{M}_\odot$ and $Z = 0.002\mathrm{Z}_\odot$. In the top-left panel, I show the comparison between the mass evolution over time simulated in SEVN (red), in the SEVN module in ISTEDDAS (green), and its GPU interpolation in the ISTEDDAS integrator (blue). In the top-right panel, I show the total mass evolution (blue), and the mass evolution of the Helium core (red) and the Carbon-Oxygen core (green). In the bottom-left and bottom-right panels, I show the evolution of the radius and the luminosity of the star respectively. In the last three plots, I zoomed on the last moment of the life of the star. In all the plots, the four colored stars represent the important moments in the star's evolution: the starting of the Helium burning phase (fuchsia), the starting of the Carbon-Oxygen burning phase (cyan), the supernova (orange), and the formation of the black hole (black).

where $\boldsymbol{\mathcal{Q}}_{ij}$ contains all the terms but the mass, and we have assumed the gravitational constant $G = 1$. In this way, the derivatives become:

$$\begin{aligned}
\dot{\boldsymbol{a}}_i &= \sum_{j\neq i}^{N} m_j \dot{\boldsymbol{\mathcal{Q}}}_{ij} &\longrightarrow&& \dot{\boldsymbol{a}}_i &= \sum_{j\neq i}^{N} m_j \dot{\boldsymbol{\mathcal{Q}}}_{ij} + \dot{m}_j \boldsymbol{\mathcal{Q}}_{ij}\,,\\
\ddot{\boldsymbol{a}}_i &= \sum_{j\neq i}^{N} m_j \ddot{\boldsymbol{\mathcal{Q}}}_{ij} &\longrightarrow&& \ddot{\boldsymbol{a}}_i &= \sum_{j\neq i}^{N} m_j \ddot{\boldsymbol{\mathcal{Q}}}_{ij} + 2\dot{m}_j \dot{\boldsymbol{\mathcal{Q}}}_{ij}\,,\\
\dddot{\boldsymbol{a}}_i &= \sum_{j\neq i}^{N} m_j \dddot{\boldsymbol{\mathcal{Q}}}_{ij} &\longrightarrow&& \dddot{\boldsymbol{a}}_i &= \sum_{j\neq i}^{N} m_j \dddot{\boldsymbol{\mathcal{Q}}}_{ij} + 3\dot{m}_j \ddot{\boldsymbol{\mathcal{Q}}}_{ij}\,,
\end{aligned}$$

where $\dot{m}$ corresponds to the angular coefficient a_m, and all the successive derivatives of the mass are zero since we interpolate it linearly. The extra terms on the right are not computationally expensive since we already compute them: in $\dot{\boldsymbol{a}}_i$ we need $\dot{m}_j \boldsymbol{\mathcal{Q}}_{ij}$, but $\boldsymbol{\mathcal{Q}}_{ij}$ it has already been computed for $\boldsymbol{a}_i$, in $\ddot{\boldsymbol{a}}_i$ we need $2\dot{m}_j \dot{\boldsymbol{\mathcal{Q}}}_{ij}$, but $\dot{\boldsymbol{\mathcal{Q}}}_{ij}$ it has already been computed for $\dot{\boldsymbol{a}}_i$, and in $\dddot{\boldsymbol{a}}_i$ we need $3\dot{m}_j \ddot{\boldsymbol{\mathcal{Q}}}_{ij}$, but $\ddot{\boldsymbol{\mathcal{Q}}}_{ij}$ it has already been computed for $\ddot{\boldsymbol{a}}_i$.

[MS#1: Quello che segue non si capisce davvero.. per favore, rifrasa tutto in modo che si capisca qualcosa.. da qui fino a “Then the neighbours of this star follow its time-step with the algorithm explained before.”][MM#1: L’ho rimesso, anche aiutandomii con chatgpt, mi sembra più chiaro adesso.] These adjustments aim to anticipate mass changes among stars in the cluster, yet they don’t entirely resolve the issue. When a star, labeled ISTEDDAS, progresses to evolve within SEVN ($t_{ist} \geq t_{sevn,i}$), inactive stars at that point won’t immediately sense the $\dot{m}$ change until they become active. Consequently, their time-steps will be updated too late. To tackle this, synchronizing the entire cluster each time a single star desires to evolve in SEVN was considered. However, this approach would impose a substantial computational burden and negate the advantages of the block time-step method. Hence, we opted to synchronize only the neighboring stars, identified through the Ahmad-Cohen scheme, of the evolving star.

To achieve this, we verify in advance whether star i intends to evolve with SEVN based on the condition:

$$t_{ist} + \Delta t_c \geq t_{sevn,i}\,, \tag{3.2}$$

where Δt_c represents the time-step relative to the star’s neighbors. If this condition holds true, we proceed to evolve the star in SEVN and, in the subsequent step, designate all its neighbors as active particles artificially. This adjustment allows them to adapt their time-steps according to the newly computed accelerations and derivatives based on the updated $\dot{m}$. Importantly, this artificial activation doesn’t disrupt the synchronization of the block time-step method. As the star is placed within the same active block as the evolving star, the method remains reliable even if the actual time-step executed by the star doesn’t align as a power of 2. In instances where a star is among the neighbors of multiple evolving stars, its block in the subsequent step will be set to the largest (or smallest time-step) among the blocks of the involved evolving stars.

Additionally, there's another issue concerning the time-step that must be addressed. In cases where the SEVN evolution involves one or multiple exceedingly small steps, the dynamical time-step of a star might be large enough to bypass these steps, resulting in the loss of information regarding the changing $\dot{m}$. Consequently, if the dynamical steps of a star meet this condition, we constrain it to not skip over more than a single SEVN step. Subsequently, the neighboring stars follow its time-step using the previously outlined algorithm.

At the end of star evolution, massive stars can have different fates: become a neutron star or a black hole after a supernova explosion, or they disintegrate because of the PISN mechanism. In the first case, we stop the evolution of the mass, assigning the final mass of the remnant, and we add to the corrected velocity of the body the kick velocity due to the supernova explosion. In the second case, the star is simply removed from the N-body system. Since the kick velocity can be much higher then the escape velocity of the simulated star cluster, those stars who receive a strong kick will become "super-fast stars". Since these stars have not reached such high speeds gradually because of accelerations, but instead, their speeds were instantaneously increased because of the supernova kick, their neighbours will not be able to adapt their time steps accordingly. Therefore, after a supernova kick, we checked the velocity of the newborn remnant, if it is higher than the escape velocity of the cluster we force its neighbours to have an equal or smaller time-step with respect to the considered star. This is achieved in the exact same way as was already explained for the synchronization of the SEVN steps and the ISTEDDAS steps.

3.1.2 Binary star evolution

As for the single star evolution, binary stars are also evolved in SEVN only if both stars have greater masses than the chosen threshold. If, instead, only one of the two stars has a greater mass with respect to the threshold, that star will evolve as a single star in SEVN.

When ISTEDDAS detects a binary that is tight enough to be integrated with TSUNAMI (see the explanation on the decision making in Subsec.2.1.1), if both the stars have masses greater with respect to the threshold and the system is gravitationally bound, a binary is created in SEVN. It has already been explained that the time steps in SEVN are adaptive, and that we do not evolve stars one step at a time: the stars keep evolving until their mass has changed beyond a certain percentage threshold. Thus, because of the nature of the time steps, creating a binary in SEVN is not an easy task. First of all, it is necessary to synchronize the two stars, to do so, for each evolving star, we create two of them to track both the actual and the last state of the stars. In this way, when a binary is created, we can bring back the two involved stars to their last state, and evolve them again until the exact moment of the binary creation. In particular, we evolve the stars until they overpass that moment for one step, then we restore their state and redo the very last step with the needed time step (using a specific function of SEVN to impose a time step). Therefore, at the creation of the binary, the two stars have the same age in SEVN, ISTEDDAS, and TSUNAMI, as it is supposed to be.

The selected binaries are evolved by SEVN for multiple time steps until either one of the two masses, the eccentricity, or the semi-major axis of the system has changed beyond a certain percentage threshold (by default we choose 2%). Then, the inner dynamics of the binary is evolved with TSUNAMI while the dynamics of its center of mass is evolved, along with the rest of the star cluster, by ISTEDDAS. Because of the binary evolution, the masses of the two stars change over time, so ISTEDDAS interpolates the total mass of the system in the same way as it was for the single stellar evolution. However, we can not use the same approach with TSUNAMI due to the way its algorithm works: it uses the relative error of the energy of the system to adjust its internal time step, however, by interpolating the masses, we cannot preserve the conservation of energy and the algorithm will not be able to compute the regularized time steps. In the worst-case scenario, the Bulirsch-Stoer algorithm will not be able to converge, blocking the whole simulation. In order to avoid this problem, we wait for the binary system to be synchronized in SEVN and TSUNAMI, then we communicate the changed masses from SEVN to TSUNAMI. The time synchronization between SEVN binary evolution and TSUNAMI dynamical evolution is achieved in the way of the synchronization of two single stars before the creation of the binary. Thus, for each binary, we have two copies of it to track both the actual and the last state of the binary stars. When the binary feels that, on the next step, the dynamical evolution will overpass its evolution time, the binary is reset to its last state and re-evolved to the exact time when TSUNAMI will arrive with its own time step. We also implement another check in TSUNAMI: whether the eccentricity or the semi-major axis of the system has changed beyond a certain percentage threshold (on default we choose 10%), TSUNAMI will ask for a time synchronization with SEVN in advance with respect to the time set by SEVN; this also happens if TSUNAMI decides to terminate the binary. Somebody could notice that using this method we evolve every binary star two times, however, this will never become a computational bottleneck since SEVN is orders of magnitudes faster than the dynamical integrators of ISTEDDAS and TSUNAMI.

When, in the next step, the binary is perfectly synchronized in SEVN and TSUNAMI, it will be possible to coherently communicate between the two codes. SEVN has to communicate to TSUNAMI the new masses of the two stars, and the new orbital parameters, that are the semi-major axis and the eccentricity (the evolution of binaries can change these parameters because of the exchange of mass in the system), and simultaneously, TSUNAMI has to communicate to SEVN its own new orbital parameters. The fact that the orbital parameters are influenced simultaneously by the dynamics and the binary stellar evolution, complicates this otherwise trivial communication. What would physically happen in reality is that the orbital parameters would continuously vary because of both the contributions happening simultaneously, however, in our case, we will approximate this behavior by combining the two contributions every time that a synchronization happens. This is the very reason why we use those thresholds to stop the codes and synchronize them, to combine the orbital parameters in the smoothest way possible. The orbital parameter combination is:

$$a_1 = a_0 + da_{\text{SEVN}} + da_{\text{TSUNAMI}} \,,$$
$$e_1 = e_0 + de_{\text{SEVN}} + de_{\text{TSUNAMI}} \,,$$

where a_1 and e_1 are the new combined semi-major axis and eccentricity respectively, a_0 and e_0 are the previous ones, which is the same for both the codes because of the previous synchronization, and finally da and de are the contributions from SEVN and TSUNAMI for the actual step and are computed as the difference between the values at the end of the step minus the values at the end :

$$da_{\text{SEVN}} = a_{1,\text{SEVN}} - a_{0,\text{SEVN}} \,,$$
$$de_{\text{SEVN}} = e_{1,\text{SEVN}} - e_{0,\text{SEVN}} \,,$$
$$da_{\text{TSUNAMI}} = a_{1,\text{TSUNAMI}} - a_{0,\text{TSUNAMI}} \,,$$
$$de_{\text{TSUNAMI}} = a_{1,\text{TSUNAMI}} - a_{0,\text{TSUNAMI}} \,.$$

At this point TSUNAMI has the new orbital parameters and communicates them to SEVN that updates the old ones. However, since TSUNAMI use Cartesian coordinates to integrate systems, we have to translate these new orbital parameters into new positions and velocities for the two bodies. The starting point is for TSUNAMI to compute da_{TSUNAMI} and de_{TSUNAMI}, to do it we switch from Cartesian coordinates, that are positions and velocities (x, y, z, v_x, v_y, v_z), to Keplerian coordinates, that are 6 orbital parameters: eccentricity (e), semi-major axis (a), inclination (i), longitude of the ascending node (Ω), argument of periapsis (ω) and true anomaly (ν). Then, using the new eccentricity and semi-major axis but maintaining fixed the old i, Ω, ω and ν, we pass again to Cartesian coordinates finding the relative positions of the two stars. Since SEVN does not give any information about the phase of the two bodies in their orbits, we do not have any way of retrieving those four angles, thus keeping the old ones from TSUNAMI is the only way to return to Cartesian coordinates.

Moreover, during this operation, we fix the position of the center of mass of the binary, that is like hypothesizing that the binary system loses mass isotopically, and we recompute positions and velocities of the two bodies with respect to the center of mass. To do so we use the following equations:

$$\boldsymbol{r}_{COM} = \frac{\boldsymbol{r}_1 m_1 + \boldsymbol{r}_2 m_2}{m_1 + m_2} \,, \tag{3.3}$$
$$\boldsymbol{v}_{COM} = \frac{\boldsymbol{v}_1 m_1 + \boldsymbol{v}_2 m_2}{m_1 + m_2} \,, \tag{3.4}$$
$$\boldsymbol{r}_{12} = \boldsymbol{r}_2 - \boldsymbol{r}_1 \,, \tag{3.5}$$
$$\boldsymbol{v}_{12} = \boldsymbol{v}_2 - \boldsymbol{v}_1 \,, \tag{3.6}$$

to obtain the final result:

$$\boldsymbol{r}_1 = \boldsymbol{r}_{COM} - \boldsymbol{r}_{12}\frac{m_2}{m_1 + m_2}\,, \tag{3.7}$$

$$\boldsymbol{r}_2 = \boldsymbol{r}_{COM} + \boldsymbol{r}_{12}\frac{m_1}{m_1 + m_2}\,, \tag{3.8}$$

$$\boldsymbol{v}_1 = \boldsymbol{v}_{COM} - \boldsymbol{v}_{12}\frac{m_2}{m_1 + m_2}\,, \tag{3.9}$$

$$\boldsymbol{v}_2 = \boldsymbol{v}_{COM} + \boldsymbol{v}_{12}\frac{m_1}{m_1 + m_2}\,. \tag{3.10}$$

Finally, if the regularized system has more than 2 bodies, TSUNAMI shifts the total center of mass of the system to zero to be consistent.

3.1.3 Energy conserving scheme

ISTEDDAS implements the Hermite 6th order integrator, which is an energy-conserving scheme, therefore we expect the total energy of the simulation to be conserved over time, except for the accumulation of numerical errors. On default, the total energy of the star cluster energy is computed considering point-mass stars with constant masses:

$$E = K + U = \frac{1}{2}\sum_{i=0}^{N}\left(m_i|\boldsymbol{v}_i|^2 - \sum_{j\neq i}^{N}\frac{m_i m_j}{|\boldsymbol{r}_{ij}|}\right)\,, \tag{3.11}$$

here the potential energy U is divided by 2 because we are computing twice all the couples (i,j) of stars, and we have assumed the gravitational constant $G = 1$. It is crucial for an N-body code to conserve the total energy because the check on the relative error of the energy:

$$\frac{\Delta E}{E(t_0)} = \frac{E(t) - E(t_0)}{E(t_0)} = 0\,, \tag{3.12}$$

is a powerful tool to check whether the simulation has any numerical or algorithmic errors.

The addition of stellar evolution with SEVN breaks the conservation of energy because of the multiple stellar evolution processes simulated that continuously decrease the mass of the evolving stars. Because of the mass losses, it is impossible to conserve the total energy of the system computing it as above. Therefore, to be able to conserve the energy, we re-inject the energy of the lost mass using $\dot{m}$. Theoretically, if the mass is constant the time derivative of the total energy would be always zero, in this case, with a varying mass, we would have instead:

$$\dot{E} = \frac{1}{2}\sum_{i=0}^{N}\left(\dot{m}_i|\boldsymbol{v}_i|^2 - \sum_{j\neq i}^{N}\frac{\dot{m}_i m_j + m_i\dot{m}_j}{|\boldsymbol{r}_{ij}|}\right)\,, \tag{3.13}$$

or, more simply:

$$\dot{E} = \sum_{i=0}^{N}\left(\frac{1}{2}\dot{m}_i|\boldsymbol{v}_i|^2 - \sum_{j\neq i}^{N}\frac{\dot{m}_i m_j}{|\boldsymbol{r}_{ij}|}\right)\,. \tag{3.14}$$

All the derivatives which do not involve $\dot{m}$ cancel each other out because, as mentioned before, if the mass is constant $\dot{E} = 0$. Thus, for each i star of the system, we integrate in time the lost energy:

$$E_{lost,i} = \int_0^t \dot{E}_i dt \,. \tag{3.15}$$

To numerically compute it, at each distant step and for each star, we accumulate $\dot{E}_i \Delta t_f$. At this point, we have retrieved the conservation of the energy as:

$$\frac{\Delta E}{E(t_0)} = \frac{E(t) - E(t_0) - E_{lost}}{E(t_0)} = 0 \,, \tag{3.16}$$

where E_{lost} is the summation of $E_{lost,i}$ on all the evolving stars of the system.

Chapter 4

Numerical tests

In this chapter, I am going to show the preliminary results from the code. In particular, I will show the benchmark of the N-body code ISTEDDAS,

4.1 ISTEDDAS tests

The benchmarks in this section are done on the new CINECA supercomputer Leonardo[1], that today is the fourth most powerful supercomputer in the world accordingly to the TOP500 list[2]. In Table 4.1 there are the specifics of the GPU nodes of Leonardo.

CPU	:	Intel(R) Xeon(R) Platinum 8358 2.60GHz (32 cores, 1 thread per core)
RAM	:	512 (8×64) GB DDR4 3200 MHz
GPU	:	4 NVIDIA Ampere 100
GRAM	:	256 (4×64) GB HBM2
Network	:	NVIDIA Mellanox HDR DragonFly++ 200Gb/s

Table 4.1: Leonardo's nodes specifics. See [30] for further details about Leonardo's architecture.

I will use a maximum of 2 nodes with 4 parallel GPUs each for the scaling tests, therefore 8 GPUs at maximum. In particular I will run scaling tests on 1, 2, 4 and 8 GPUs for star clusters with 2048 (2^{11}), 4096 (2^{12}), 8192 (2^{13}), 16384 (2^{14}), 32768 (2^{15}), 65536 (2^{16}), 131072 (2^{17}), and 262144 (2^{18}) stars (as shown in Table 4.2). The star clusters are generated using the McLuster[3] code [16]. All these clusters have a King density profile with $W_0 = 5$ (dimensionless parameter which specifies the model concentration), a half-mass radius of 2 pc, a velocity dispersion of $\sigma = 1.3127$ km/s, they are at the virial equilibrium, and their initial mass function follows the Kroupa

[1]Check the link https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.4%3A+Leonardo+UserGuide for extra details about Leonardo.

[2]TOP500 June 2023 link: https://www.top500.org/lists/top500/2023/06/

[3]The code is public and available here: https://github.com/ahwkuepper/mcluster.

IMF within the range of star's masses $0.1\text{M}_\odot < M_* < 150\text{M}_\odot$. Moreover, these clusters do not have binary stars, tidal field, mass segregation, or fractality. All the simulations reach a total time of 10 in code units, check Table 4.2 to see how many Myr this corresponds for each simulation.

N_{stars}	M_{tot} [M$_\odot$]	ρ [M$_\odot$/pc^3]	t_{relax} [Myr]	T_{tot} [Myr]	$\tau_\%$
2048 (2^{11})	1322.4	57.37	109.71	4.11	3.75%
4096 (2^{12})	2597.9	77.53	82.65	2.93	3.54%
8192 (2^{13})	5165.0	154.13	41.82	2.08	4.97%
16384 (2^{14})	10398.5	310.31	20.63	1.46	7.07%
32768 (2^{15})	20429.9	609.66	10.69	1.04	9.73%
65536 (2^{16})	41664.7	1243.34	5.14	0.73	14.20%
131072 (2^{17})	83983.8	2506.21	2.53	0.52	20.55%
262144 (2^{18})	169716.0	5064.59	1.24	0.36	29.04%

Table 4.2: List of simulations made for the scaling tests of ISTEDDAS. It contains: the total number of stars, the total mass (in M$_\odot$), the two-body relaxation timescale (in Myr), the total simulated time (in Myr), and the total simulated time as a percentage of the relaxation time: $\tau_\% = T_{tot}/t_{relax} \cdot 100$.

First of all, in Fig.4.1 I show the relative error on the total energy of the cluster (see Eq.3.12): note how well the energy conservation is maintained by the Hermite 6th order integrator for all the simulations made for the scaling tests. From the plot, one could think that the bigger the number of stars, the faster the error grows. In reality, the rate at which the error grows depends on how many t_relax were simulated. So, the total simulated time as a percentage of the relaxation time ($\tau_\% = T_{tot}/t_{relax} \cdot 100$, see Table 4.2) tells us that higher the number of particles higher $\tau_\%$, and so faster the error grows (for the specifically chosen settings of these simulations). In theory, in a set of simulations with settings chosen ad hoc to have $\tau_\%$ equal in all the simulations, we would see a similar error plot for all the cases. [MS#2: Perche dici che questa cosa e' expected? Secondo me non e' expected a parita' di tempi di crossing... per esempio, su 10 tempi di crossing mi aspetto conservazioni di energia simili, solo che 10 tempi di crossing per un sistema piccolo sono pochi Myr, mentre su un sistema grande sono olti Myr.. quindi, paradossalmente, a parita' di Myr integrati, mi aspetto che l'errore sia migliore per i clusters con piu' particelle... per favore spiega meglio qui cosa sta succedendo...]. However, the error is always maintained quite low (maximum $\sim 10^{-5}$), and it is consistent among the simulations with different numbers of GPUs, proving the correctness of the multi-GPU implementation.

In Fig.4.2 and Fig.4.3 I show the results of the scaling tests on a single node. In particular, in Fig.4.2 I show the larger contributions to the total running time of the simulations: the ANN tree contribution (see the subsection 1.4.3), the MPI contribution, the computation of the forces due to far stars ($\boldsymbol{a}_{far,new}$ in Section 1.3), and the contribution from the searching algorithm that determines which stars need to be integrated for each step. Since the ANN tree contribution is the larger one, I also show its two main internal contributions: The tree construction, and the search

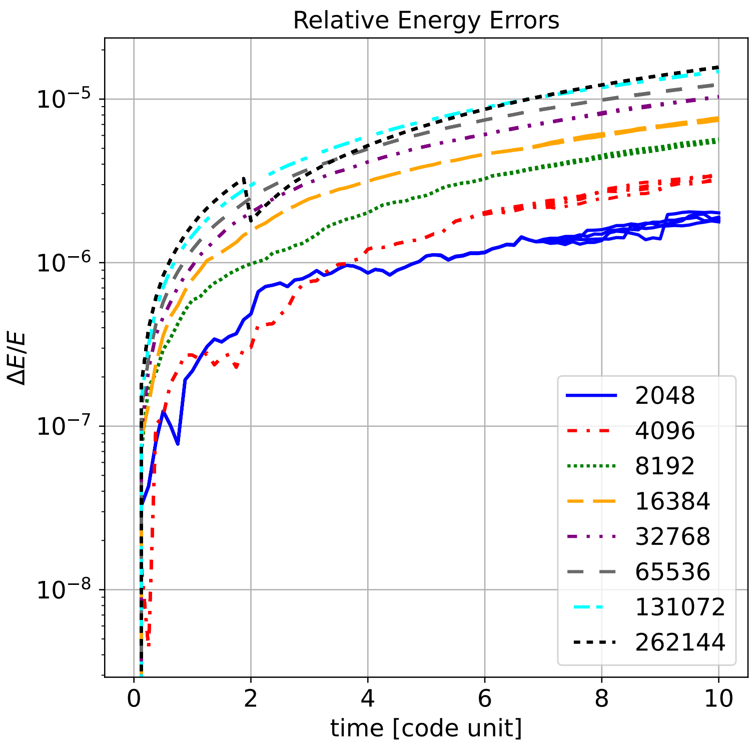

Figure 4.1: Relative error on the total energy of the system (see Eq.3.12) versus the time in code units for each scaling test simulation. For each simulation multiple lines are shown, one for each number of GPUs used (1,2,4, and 8), they have the same style and color since they superpose almost perfectly, as expected.

for neighbours, which is the most computationally expensive operation in the code. Instead, in Fig.4.3 I show less important contributions together with the total running time for reference: the computation of the forces due to far stars, close stars, and the contribution from the old neighbours (respectively $\boldsymbol{a}_{far,new}$, $\boldsymbol{a}_{close,new}$, and $\boldsymbol{a}_{close,old}$ in Section 1.3), the contributions from the predictor and the corrector, the contribution from updating the time-steps of all the stars at each step, and the contribution from the memory transfers between CPUs and GPUs. In Fig.4.4 I show the scaling of the far forces kernel alone. In order to highlight how it performs, I plot the running time with a single GPU with the running time for other cases. The computation of the forces was the bottleneck in N-body codes for decades, therefore we optimize it as much as possible, obtaining a remarkable result. The cases with few stars do not scale well, as expected, since a single GPU can already handle that number of stars, and dividing the work just introduces extra operations: memory transfers, reductions, and gatherings. However, from 32768, stars the scaling on a single node becomes perfect, while on multiple nodes at least 65536 stars are needed to improve the performance. In Fig.4.5 I show the results of the scaling tests on two nodes: simulation characteristics are the same as in Fig.4.2, except for the fact I show just the two cases with more stars (131072 and 262144). From the plots, it is clear that the MPI implementation slows down ISTEDDAS too much, making it even worse than running the code on a single node.

The reason behind this behavior is that the far forces kernel is the only part of the code that can scale with an increasing number of GPUs. ISTEDDAS was designed in this way because that was the main bottleneck for N-body codes. Clearly from all the scaling tests that I have shown, the actual bottleneck of the code is the ANN tree and the MPI implementation, and both of these bottlenecks can be solved. For the tree, as I extensively discussed in Section 1.4.3, the solution would be to switch to another library, in particular, we could use FLANN, which already runs on GPU and can scale on multiple GPUs.Another possible solution would be to think about a new algorithm to further reduce the number of tree constructions within each macro step, but this is beyond the scope of this thesis. For MPI, the solution would be to exploit the new technology in the new GPU supercomputers. For example, each node of the CINECA's supercomputer Leonardo is provided with a new NVIDIA Mellanox HDR Dragonfly++ network card that enables direct communication between GPUs from different nodes, and provides some preliminary calculations when needed, in order to speed up communications even more (for example if a reduction operation is called in the code). Moreover, the communication between GPUs on the same nodes can avoid passing through the CPU because of the InfiniBand connections between GPUs. Nowadays, almost all the most powerful supercomputers in the world are provided with GPUs and use a similar technology to speed up communications among accelerators. Among them, Frontier[4] and LUMI[5] (that use AMD GPUs) or Leonardo[6], Summit[7], and Sierra[8] (that use NVIDIA GPUs). Clearly, this technology represents the future of GPU computation. In order to use such technology one could use the CUDA-Aware MPI (for NVIDIA hardware), which is an implementation of MPI that can exploit the new connections between GPUs, or one could use NCCL, an NVIDIA library for communication that underneath uses CUDA-Aware MPI. These optimizations will be done in future works related to the development of ISTEDDAS.

[4]First position in the TOP500, website: https://www.olcf.ornl.gov/frontier/

[5]Third position in the TOP500, website: https://www.lumi-supercomputer.eu/

[6]Fourth position in the TOP500, website: https://leonardo-supercomputer.cineca.eu/it/home-it/

[7]Fifth position in the TOP500, website: https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/

[8]Sixth position in the TOP500, website: https://hpc.llnl.gov/hardware/compute-platforms/sierra

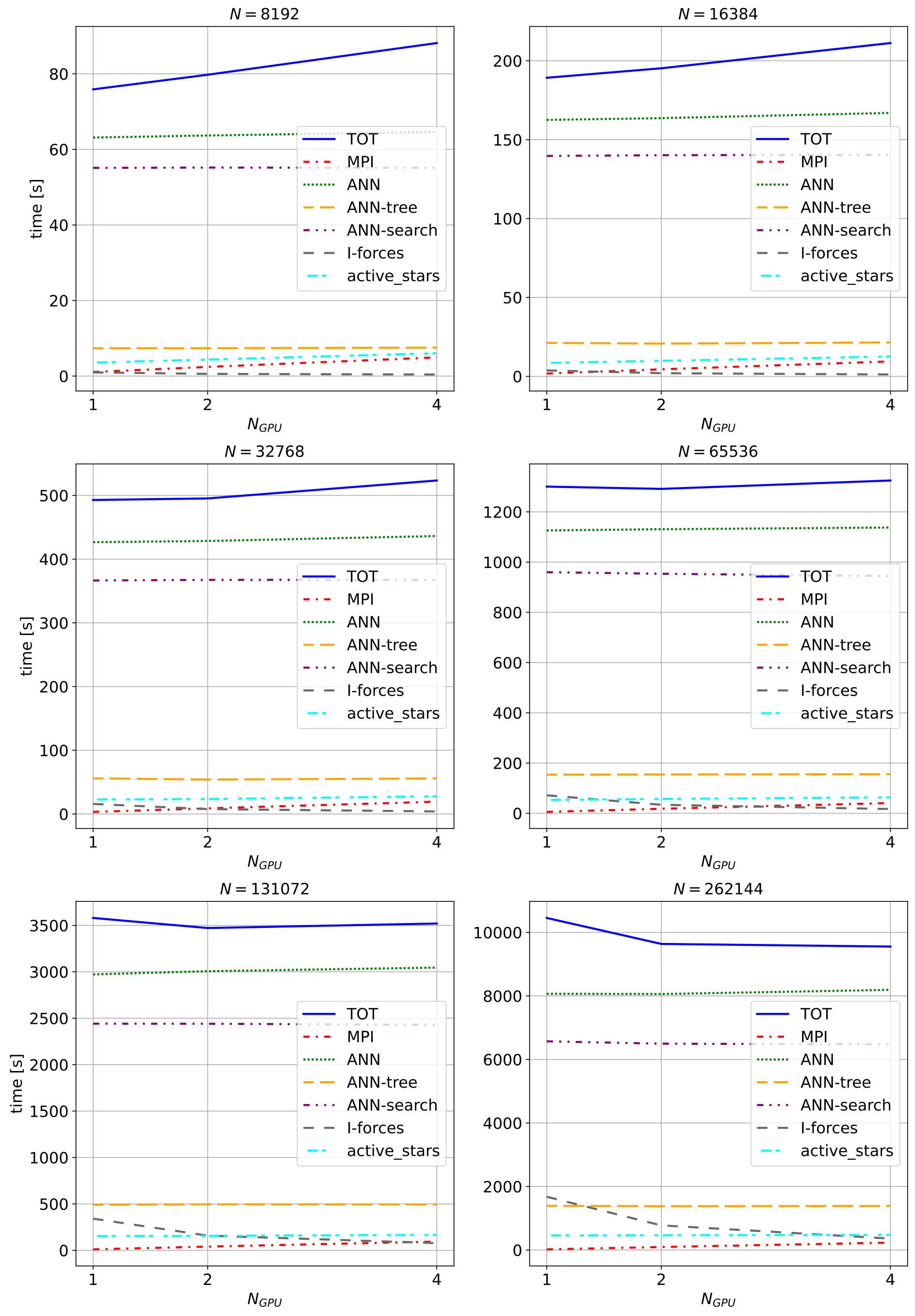

Figure 4.2: Scaling tests of ISTEDDAS. In each panel, it is shown, for a certain total number of stars N, the measured times versus the number of used GPUs (N_{GPU}). The six most important timings are shown together with the total time ("TOT" in blue): the MPI contribution ("MPI" in red), the ANN tree contribution ("ANN" in green), the tree construction and tree search contributions to ANN ("ANN-tree" in yellow, and "ANN-search" in purple), the contribution of the far forces computation ("I-forces" in grey), and the search for new active stars contribution ("active_stars" in cyan).

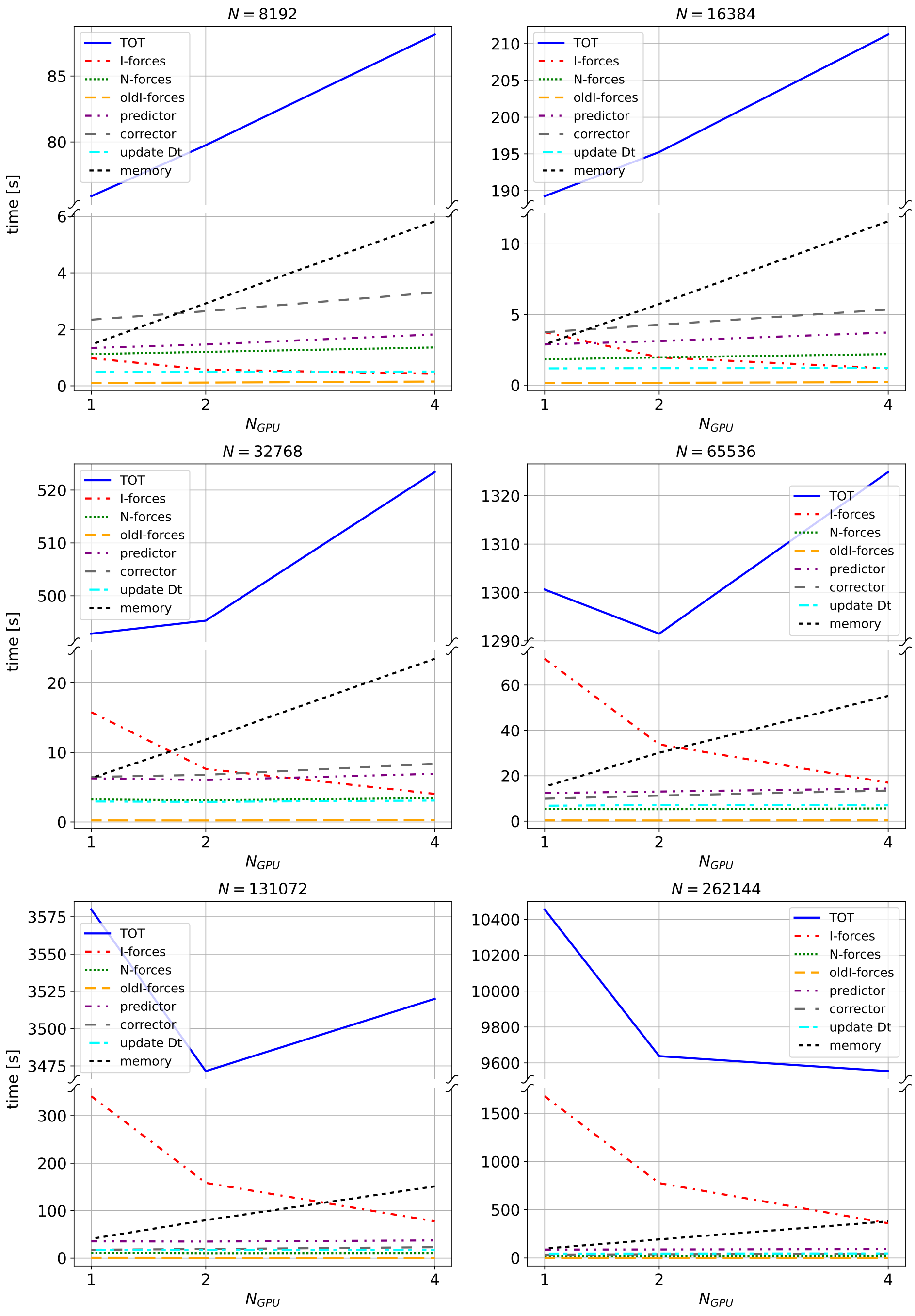

Figure 4.3: Scaling tests of ISTEDDAS. In each panel, it is shown, for a certain total number of stars N, the measured times versus the number of used GPUs (N_{GPU}). The seven important but less costly timings are shown together with the total time ("TOT" in blue): the contribution of the far, near, and previous near forces computations ("I-forces" in red, "N-forces" in green, and "oldI-forces" in yellow), the predictor contribution ("predictor" in purple), the corrector contribution ("corrector" in grey), the contribution for updating the time-steps ("update_Dt" in cyan), and the memory transfer between CPUs and GPUs contribution ("memory" in black).

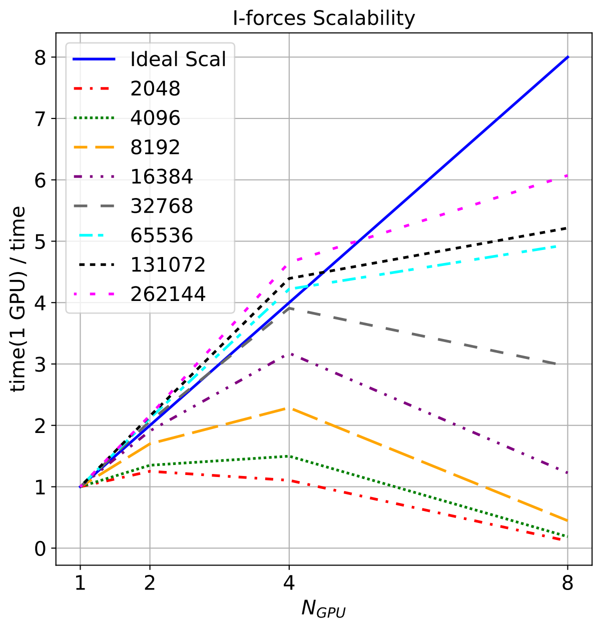

Figure 4.4: Scaling tests of ISTEDDAS. The plot shows the ratio between the measured times for a single GPU and the total number of used GPUs (t_{1GPU}/t_{NGPU}) versus the number of used GPUs (N_{GPU}). The blue line shows the ideal scaling, the other lines show the scaling of the far forces computation kernel for each simulation.

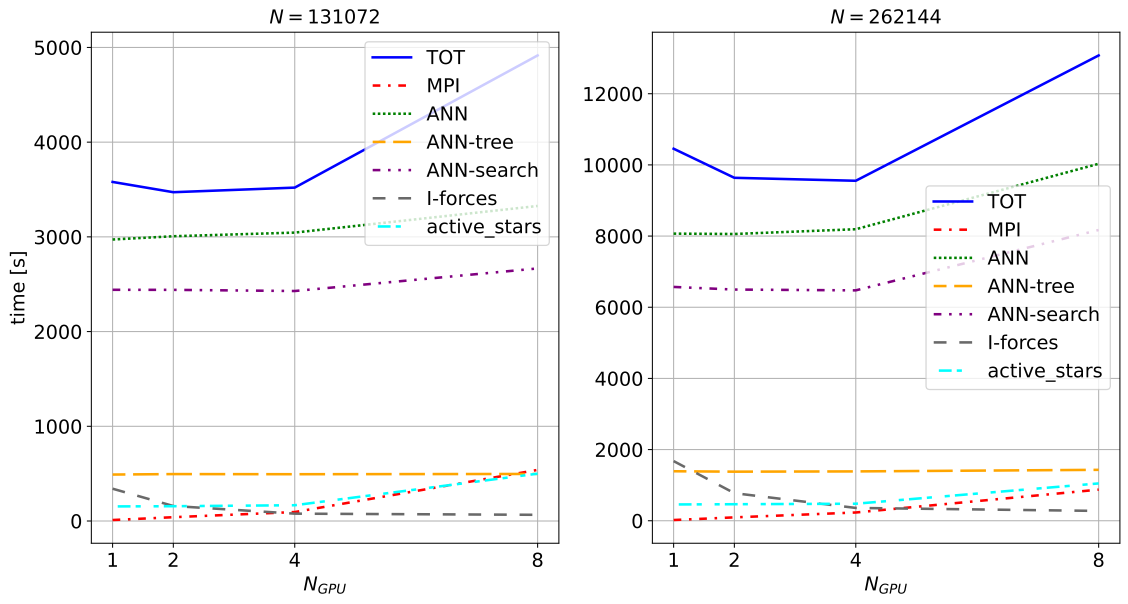

Figure 4.5: Scaling tests of ISTEDDAS. Same plot type of Fig.4.2. In this case with the “8 GPUS” entry on the x-axis and just for the two more expensive simulations (N equal to 2^{17} and 2^{18}).

4.2 TSUNAMI tests

The following tests are made on a laptop, its specifications are in Table 1.1. For Fig.4.6 two simulations were done, both of them on the same star cluster with 2048 stars and one tight binary system. The two simulations run for the same amount of time on my laptop (5 minutes). The one with ISTEDDAS alone was able to evolve the star cluster for just ~ 1400 years. During the simulation, the error keeps increasing almost linearly (red and red dashed lines), this is due to the vicinity of the two stars in the binary: the Hermite 6^{th} order integrator is a powerful tool but is not symplectic and cannot integrate correctly such a tight binary system. Moreover, the block time-step method (see Section 1.2) assigns the smallest possible time-step to the system because of the enormous acceleration and its derivatives involved (see Eq.1.9). As a consequence, the simulation takes so many steps to progress that the numerical error accumulation becomes an important contribution to the total energy error of the simulation. Thus, ISTEDDAS alone gets "stuck" accumulating numerical errors in simulation with tight binaries, in this case even a tiny star cluster, as the one in this example, could become challenging to simulate. Instead, the simulation with TSUNAMI was able to evolve the star cluster for ~ 40000 years keeping the relative error on the energy of the star cluster stable under 10^{-7} (blue dashed line), since the binary was correctly integrated with the ARC integrator with a minimal error accumulation (green dotted line). This plot demonstrates the importance of using two distinct integrators for direct N-body simulation of star clusters: one to take care of long-range interactions and one to take care of tight systems such as binary stars, hierarchical systems, and close encounters.

In Fig.4.7 and Fig.4.8 two examples of hierarchical systems are shown, an unstable one and a stable one respectively. Both of them have three levels of zoom, in order to show both the motion of the overall system and the specific motion of the tight binaries. In both the plot the real movement of stars in the combination ISTEDDAS+TSUNAMI came out: TSUNAMI integrates the internal motion of the system (binary stars or hierarchical system), while ISTEDDAS integrates the motion of its center of mass in the cluster. Therefore, when plotting the orbits, a "twitchy" movement came out, where the orbits do not evolve smoothly together with the center of mass, they evolve for the amount of time of the next ISTEDDAS step but fixed in the previous center of mass coordinates. Moreover, in Fig.4.7, one can also see how the decision-making (in particulate the condition in Eq.2.6) keeps this system in TSUNAMI even when the third body and the inner binary are very far at their apoastron. This is necessary until the three bodies are in a bound configuration with a very tight periastron in order to avoid the system going in and out from TSUNAMI at each orbit.

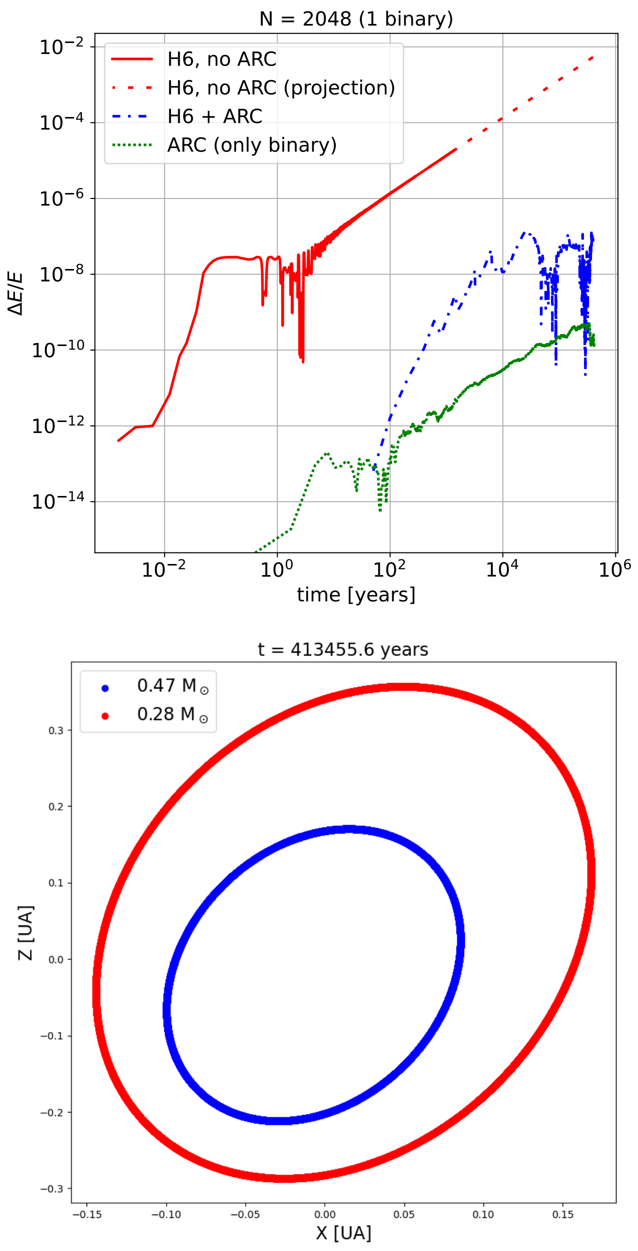

Figure 4.6: Simulation of a star cluster with 2048 stars and a single binary made with ISTEDDAS alone using just thee Hermite 6th order integrator, and then made with ISTEDDAS+TSUNAMI, using the ARC integrator for the binary. The top panel shows the relative error of the energy of the star cluster versus the time in years. In red is shown the error for the simulation with ISTEDDAS alone, and the red dashed line shows the projection of the error. In blue is shown the error for the simulation where the binary is integrated by TSUNAMI, and in green is shown the error of the ARC integrator (TSUNAMI) on the binary alone. The bottom panel shows, in the x-z plane (in astronomic units), the evolution of the orbit of the binary. The coordinates are re-centered in the center of mass of the binary in order to show its stability in 413455 years of evolution.

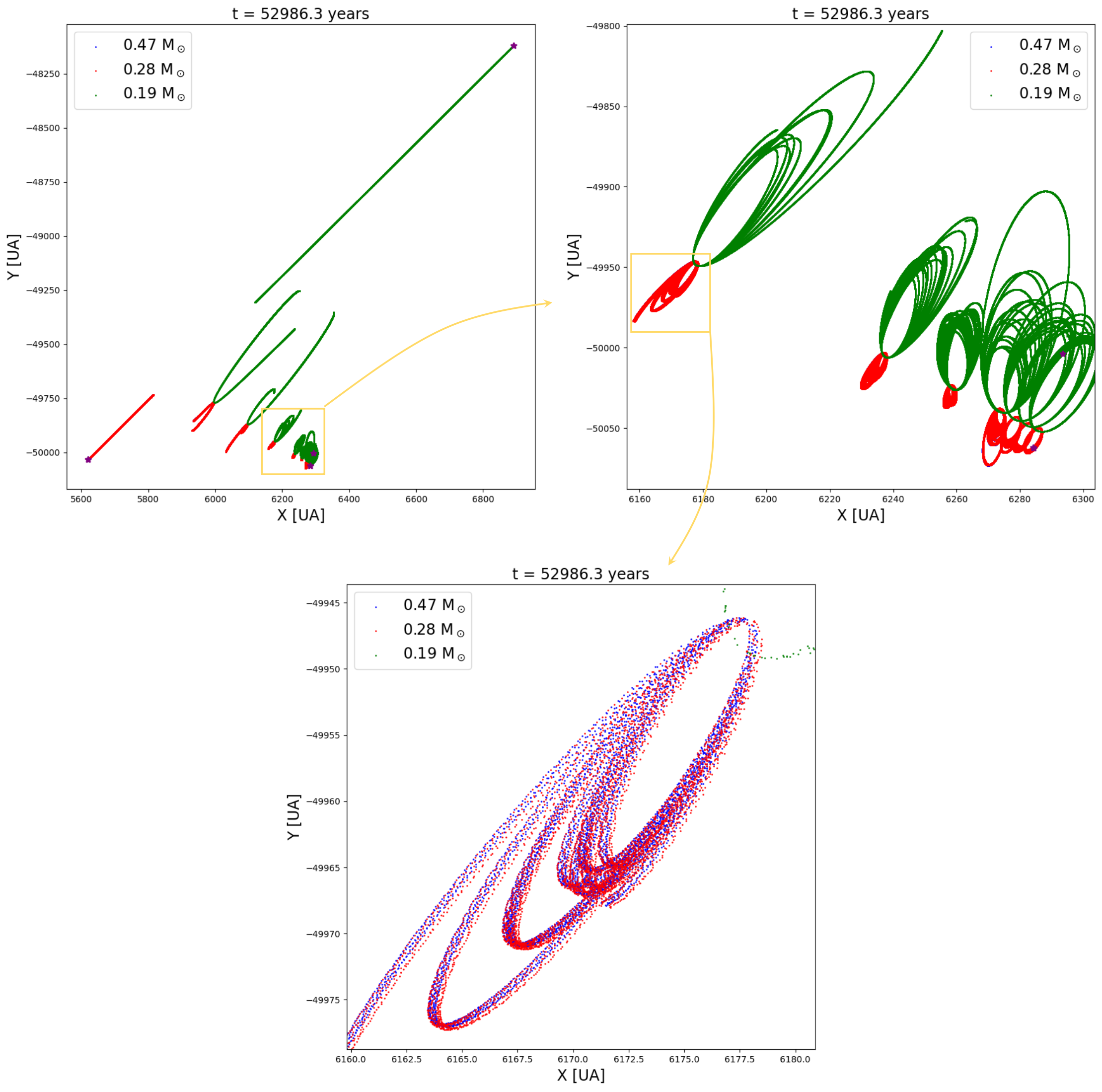

Figure 4.7: Example of the evolution of a hierarchical system in an unstable configuration. The following plots are in the x-y plane (in astronomic units) where the position (0,0) is the center of mass of the star clusters. The inner binary is formed by two stars with masses 0.47M$_\odot$ (blue) and 0.28M$_\odot$ (red), and the third body has a mass of 0.19M$_\odot$ (green). The top-left panel shows the entire evolution of the system for 52986 years, from its beginning to its disruption. The top-right panel is a zoom of the yellow square in the first panel, it shows the orbits in the initial part of the evolution. The bottom panel is a zoom of the yellow square in the second panel, it shows the orbit of the inner binary, which is so tight is not visible in the other two plots.

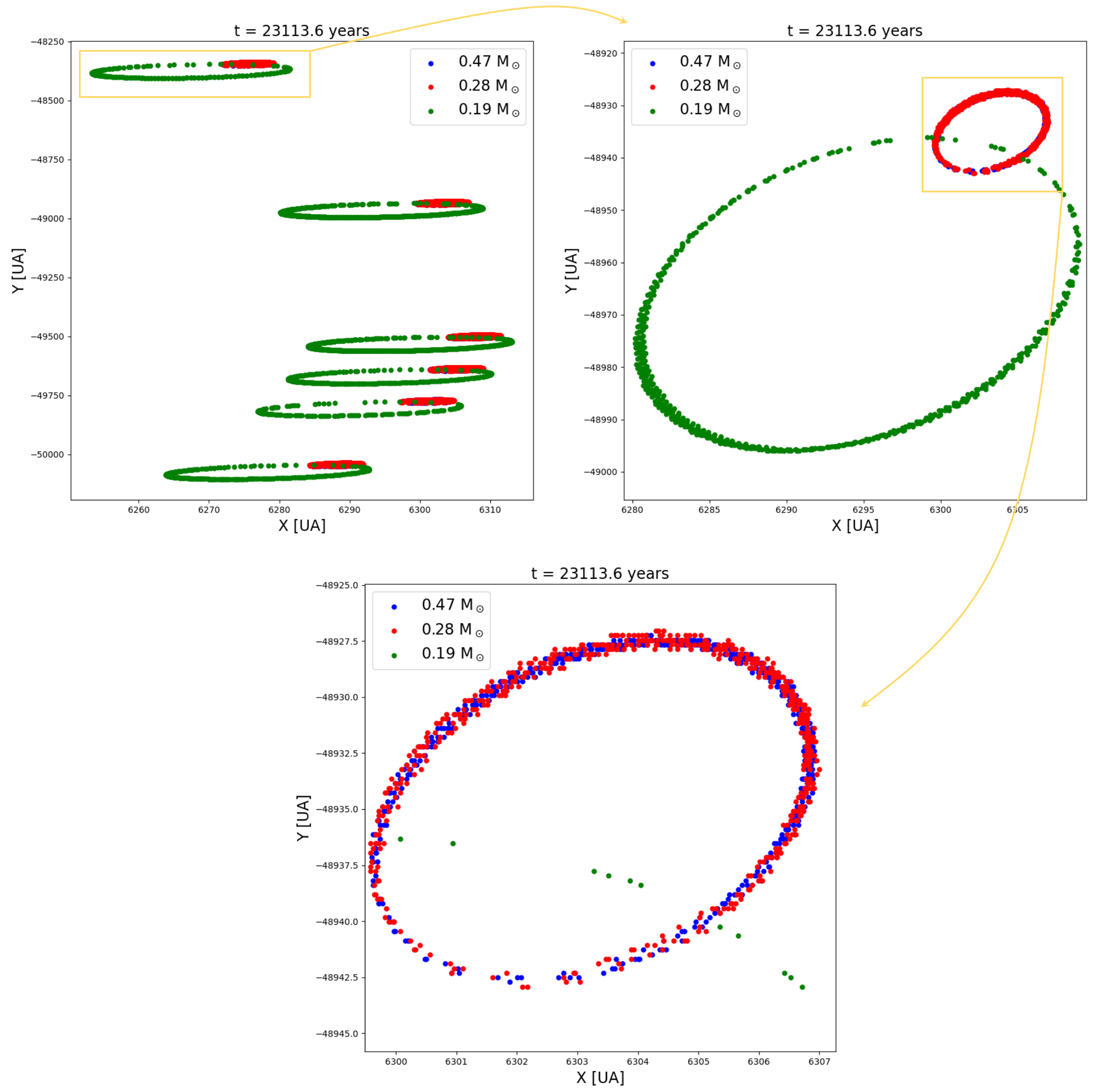

Figure 4.8: Example of the evolution of a hierarchical system in a stable configuration. The following plots are in the x-y plane (in astronomic units) where the position (0,0) is the center of mass of the star clusters. The inner binary is formed by two stars with masses 0.47M$_\odot$ (blue) and 0.28M$_\odot$ (red), and the third body has a mass of 0.19M$_\odot$ (green). The top-left panel shows the entire evolution of the system for 23113 years. The top-right panel is a zoom of the yellow square in the first panel, and the bottom panel is a zoom of the yellow square in the second panel, it shows the orbit of the inner binary, which is so tight is not visible in the other two plots.

Conclusions

In this thesis work, I presented the strategies in place to develop ISTEDDAS, a new direct N-body code written in C++ and CUDA, that is designed to run natively on GPUs (proven to be much more proficient than CPUs in handling highly parallelizable problems) and that uses a combination of a high-order integrator (Hermite 6^{th} order) with block time-step and the Ahmad-Cohen neighbours scheme to significantly speed up high-accuracy N-body simulations.

Direct N-body simulations of star clusters still struggle to resolve small-scale binary interactions while simulating large systems simultaneously. Achieving high resolution in areas where binary interactions occur is computationally demanding and often limits the size of the simulated cluster or the number of particles that can be realistically tracked. Some codes avoid a direct N-body approach to solve this computational problem (e.g. using tree-based codes) and, although the statistical properties of star clusters are reproduced correctly, they do not compute precisely the trajectories of stars, overall losing precision. On top of this, most N-body codes are coupled with outdated population-synthesis codes to evolve stars. These population-synthesis codes are difficult to update with the recent prescriptions for single and binary stellar evolution.

ISTEDDAS is expected to overcome the shortcomings of current stat-of-the-art N-body codes.

ISTEDDAS can accurately integrate very tight binaries, strong gravitational interactions, and hierarchical few-body systems thanks to the coupling with the state-of-the-art code TSUNAMI, which implements the Algorithmic Regularization Chain combined with the Bulirsch-Stoer method. This allows us to resolve all the spans of spatial and temporal scales of star clusters and to investigate the formation and evolution of compact-object binaries (i.e., the progenitors of the gravitational-wave events observed by the LIGO-Virgo-KAGRA collaboration) with high accuracy.

ISTEDDAS has also been coupled with the population synthesis code SEVN, a new, up-to-date population-synthesis code based on look-up tables, that takes care of the evolution of both single and binary stars. Because of the use of look-up tables, updating SEVN does not require much effort and this makes it the perfect tool in order to keep the simulations always updated with the new stellar evolution prescriptions.

At the moment, a preliminary version of ISTEDDAS coupled with TSUNAMI and SEVN is already functional, and some results have been presented in this thesis. The interface ISTEDDAS – TSUNAMI supports the dynamical evolution of tight systems and binaries and the interface ISTEDDAS – SEVN supports the evolution of single stars. Binary stellar evolution, which represents the final coupling of all three codes, is currently under testing.

From the point of view of the performance, ISTEDDAS demonstrates to be very effective in computing the forces within the star cluster. In fact, while the raw complete computation of force scales as N^2, the kernel that computes forces in ISTEDDAS scales roughly linearly on multiple GPUs (see Fig.4.4). However, the overall performance of the code is still sub-optimal since all the extra operations made to optimize the forces computation are still to be fully optimized, especially the ANN-tree part. Optimization strategies are possible and they are currently under discussion.

TSUNAMI already demonstrates its efficiency in handling tight systems with respect to the ISTEDDAS integrator (see Fig.4.6). In this thesis, we presented some preliminary results. However, the interface is still under intensive testing: there are still some peculiar cases in which the decision-making algorithm is sub-optimal.

SEVN was the last addition and its interface and it seems to have almost no impact on the performance of the code since both ISTEDDAS and TSUNAMI are significantly more computationally demanding.

Regarding the future of the code, the plan is to continue developing and optimizing the interfaces, also implementing the one between TSUNAMI and SEVN. We plan to further optimize the code by implementing a new GPU library for the approximate nearest-neighbors search algorithm to overcome the current bottleneck of the N-body integrator, and by implementing the GPU version of TSUNAMI to speed up the parallel integration of many regularized systems, in order to optimize the code for star clusters with a high fraction of binary systems. We also plan to implement the possibility of adding an external potential to the force calculation in order to simulate the interaction of dense stellar systems with an external tidal field (e.g., host galaxy).

The combination ISTEDDAS – TSUNAMI – SEVN, once 100% operational, will be able to help investigate a plethora of different problems related, in general, to star clusters and galaxies, not only gravitational waves from compact binary coalescences. The code will be able to accurately simulate all the trajectories in the system, comprehensive of binary systems and tight hierarchical systems, and to handle the stellar evolution of single and binary stars with up-to-date stellar prescriptions.

In essence, the code's ability to efficiently simulate dense stellar systems while considering both stellar evolution and dynamics in a comprehensive manner is pivotal. ISTEDDAS fills a critical gap, offering a more coherent and precise approach to understanding the multifaceted interactions within star clusters, their relation to compact object binaries and gravitational waves, their galactic context, and their impact on the evolution of galaxies as a whole.

Bibliography

[1] S. J. Aarseth. From NBODY1 to NBODY6: The Growth of an Industry. *Publications of the Astronomical Society of the Pacific*, 111(765):1333–1346, Nov. 1999. doi:10.1086/316455.

[2] S. J. Aarseth. Star Cluster Simulations: the State of the Art. In J. Henrard and S. Ferraz-Mello, editors, *Impact of Modern Dynamics in Astronomy*, page 127, Jan. 1999. URL: https://ui.adsabs.harvard.edu/abs/1999imda.coll..127A.

[3] S. J. Aarseth. *Gravitational N-Body Simulations: Tools and Algorithms*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, 2003. doi:10.1017/CBO9780511535246.

[4] S. J. Aarseth. Regularization tools for binary interactions. In J. Makino and P. Hut, editors, *Astrophysical Supercomputing using Particle Simulations*, volume 208, page 295, jan 2003. arXiv:astro-ph/0110148.

[5] A. Ahmad and L. Cohen. A numerical integration scheme for the n-body gravitational problem. *Journal of Computational Physics*, 12(3):389–402, 1973. URL: https://www.sciencedirect.com/science/article/pii/0021999173901605, doi:10.1016/0021-9991(73)90160-5.

[6] P. Berczik, R. Spurzem, L. Wang, S. Zhong, and S. Huang. Up to 700k GPU cores, Kepler, and the Exascale future for simulations of star clusters around black holes. In *Third International Conference "High Performance Computing*, pages 52–59, Oct. 2013. arXiv:1312.1789.

[7] A. Bressan, P. Marigo, L. Girardi, B. Salasnich, C. Dal Cero, S. Rubele, and A. Nanni. PARSEC: stellar tracks and isochrones with the PAdova and TRieste Stellar Evolution Code. MNRAS, 427(1):127–145, nov 2012. arXiv:1208.4498, doi:10.1111/j.1365-2966.2012.21948.x.

[8] R. Capuzzo-Dolcetta, M. Spera, and D. Punzo. A fully parallel, high precision, N-body code running on hybrid computing platforms. *Journal of Computational Physics*, 236:580–593, Mar. 2013. arXiv:1207.2367, doi:10.1016/j.jcp.2012.11.013.

[9] Y. Chen, A. Bressan, L. Girardi, P. Marigo, X. Kong, and A. Lanza. PARSEC evolutionary tracks of massive stars up to 350 $M_\odot$ at metallicities $0.0001 \leq Z \leq$

0.04. MNRAS, 452(1):1068–1080, Sept. 2015. arXiv:1506.01681, doi:10.1093/mnras/stv1281.

[10] Y. Chen, L. Girardi, A. Bressan, P. Marigo, M. Barbieri, and X. Kong. Improving PARSEC models for very low mass stars. MNRAS, 444(3):2525–2543, Nov. 2014. arXiv:1409.0322, doi:10.1093/mnras/stu1605.

[11] G. Costa, A. Bressan, M. Mapelli, P. Marigo, G. Iorio, and M. Spera. Formation of GW190521 from stellar evolution: the impact of the hydrogen-rich envelope, dredge-up, and $^{12}C(\alpha, \gamma)^{16}O$ rate on the pair-instability black hole mass gap. MNRAS, 501(3):4514–4533, Mar. 2021. arXiv:2010.02242, doi:10.1093/mnras/staa3916.

[12] F. Groh, L. Ruppert, P. Wieschollek, and H. P. A. Lensch. GGNN: Graph-based GPU Nearest Neighbor Search. *arXiv e-prints*, page arXiv:1912.01059, Dec. 2019. arXiv:1912.01059, doi:10.48550/arXiv.1912.01059.

[13] G. Iorio, G. Costa, M. Mapelli, M. Spera, G. J. Escobar, C. Sgalletta, A. A. Trani, E. Korb, F. Santoliquido, M. Dall'Amico, N. Gaspari, and A. Bressan. Compact object mergers: exploring uncertainties from stellar and binary evolution with SEVN. *arXiv e-prints*, page arXiv:2211.11774, Nov. 2022. arXiv:2211.11774, doi:10.48550/arXiv.2211.11774.

[14] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *arXiv e-prints*, page arXiv:1702.08734, Feb. 2017. arXiv:1702.08734, doi:10.48550/arXiv.1702.08734.

[15] S. Konstantinidis and K. D. Kokkotas. MYRIAD: a new N-body code for simulations of star clusters. A&A, 522:A70, Nov. 2010. arXiv:1006.3326, doi:10.1051/0004-6361/200913890.

[16] A. H. W. Küpper, T. Maschberger, P. Kroupa, and H. Baumgardt. Mass segregation and fractal substructure in young massive clusters – I. The McLuster code and method calibration. *Monthly Notices of the Royal Astronomical Society*, 417(3):2300–2317, 10 2011. arXiv:https://academic.oup.com/mnras/article-pdf/417/3/2300/3826875/mnras0417-2300.pdf, doi:10.1111/j.1365-2966.2011.19412.x.

[17] J. Makino and S. J. Aarseth. On a Hermite Integrator with Ahmad-Cohen Scheme for Gravitational Many-Body Problems. *Publications of the Astronomical Society of Japan*, 44:141–151, Apr. 1992. URL: https://ui.adsabs.harvard.edu/abs/1992PASJ...44..141M.

[18] J. Makino and P. Hut. Performance Analysis of Direct N-Body Calculations. ApJS, 68:833, Dec. 1988. doi:10.1086/191306.

[19] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *arXiv e-prints*,

page arXiv:1603.09320, Mar. 2016. arXiv:1603.09320, doi:10.48550/arXiv.1603.09320.

[20] C. Maureira-Fredes and P. Amaro-Seoane. GraviDy, a GPU modular, parallel direct-summation N-body integrator: dynamics with softening. *Monthly Notices of the Royal Astronomical Society*, 473(3):3113–3127, 09 2017. arXiv:https://academic.oup.com/mnras/article-pdf/473/3/3113/21736460/stx2468.pdf, doi:10.1093/mnras/stx2468.

[21] M. Mencagli, N. Nazarova, and M. Spera. ISTEDDAS: a new direct n-body code to study merging compact-object binaries. *Journal of Physics: Conference Series*, 2207(1):012051, mar 2022. doi:10.1088/1742-6596/2207/1/012051.

[22] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*, 2009. URL: http://www.image.ntua.gr/iva/files/MujaLowe_ICCVTA2009%20-%20Fast%20Approximate%20Nearest%20Neighbors%20with%20Automatic%20Algorithm%20Configuration.pdf.

[23] C. T. Nguyen, G. Costa, L. Girardi, G. Volpato, A. Bressan, Y. Chen, P. Marigo, X. Fu, and P. Goudfrooij. PARSEC V2.0: Stellar tracks and isochrones of low- and intermediate-mass stars with rotation. A&A, 665:A126, Sept. 2022. arXiv:2207.08642, doi:10.1051/0004-6361/202244166.

[24] K. Nitadori, M. Iwasawa, and J. Makino. 6th and 8th Order Hermite Integrator Using Snap and Crackle. In E. Vesperini, M. Giersz, and A. Sills, editors, *Dynamical Evolution of Dense Stellar Systems*, volume 246, pages 473–474, May 2008. doi:10.1017/S1743921308016207.

[25] S. F. Portegies Zwart, S. L. W. McMillan, P. Hut, and J. Makino. Star cluster ecology — IV. Dissection of an open star cluster: photometry. *Monthly Notices of the Royal Astronomical Society*, 321(2):199–226, 02 2001. arXiv:https://academic.oup.com/mnras/article-pdf/321/2/199/3023740/321-2-199.pdf, doi:10.1046/j.1365-8711.2001.03976.x.

[26] J. Tang, A. Bressan, P. Rosenfield, A. Slemer, P. Marigo, L. Girardi, and L. Bianchi. New PARSEC evolutionary tracks of massive stars at low metallicity: testing canonical stellar evolution in nearby star-forming dwarf galaxies. *Monthly Notices of the Royal Astronomical Society*, 445(4):4287–4305, 11 2014. arXiv:https://academic.oup.com/mnras/article-pdf/445/4/4287/6097104/stu2029.pdf, doi:10.1093/mnras/stu2029.

[27] A. Trani, M. Spera, M. Mencagli, et al. The TSUNAMI code, 2024. In preparation.

[28] A. A. Trani, M. S. Fujii, and M. Spera. The Keplerian Three-body Encounter. I. Insights on the Origin of the S-stars and the G-objects in the Galactic Center. *The Astrophysical Journal*, 875(1):42, Apr. 2019. arXiv:1809.07339, doi:10.3847/1538-4357/ab0e70.

[29] A. A. Trani, M. Spera, N. W. C. Leigh, and M. S. Fujii. The keplerian three-body encounter. II. comparisons with isolated encounters and impact on gravitational wave merger timescales. *The Astrophysical Journal*, 885(2):135, nov 2019. doi:10.3847/1538-4357/ab480a.

[30] M. Turisini, G. Amati, and M. Cestari. LEONARDO: A Pan-European Pre-Exascale Supercomputer for HPC and AI Applications. *arXiv e-prints*, page arXiv:2307.16885, July 2023. arXiv:2307.16885, doi:10.48550/arXiv.2307.16885.

[31] L. Wang, M. Iwasawa, K. Nitadori, and J. Makino. PETAR: a high-performance N-body code for modelling massive collisional stellar systems. *Monthly Notices of the Royal Astronomical Society*, 497(1):536–555, sep 2020. arXiv:2006.16560, doi:10.1093/mnras/staa1915.

[32] L. Wang, R. Spurzem, S. Aarseth, K. Nitadori, P. Berczik, M. B. N. Kouwenhoven, and T. Naab. nbody6++gpu: ready for the gravitational million-body problem. *Monthly Notices of the Royal Astronomical Society*, 450(4):4070–4080, 05 2015. arXiv:https://academic.oup.com/mnras/article-pdf/450/4/4070/5782448/stv817.pdf, doi:10.1093/mnras/stv817.