

# GraviDy, a GPU modular, parallel direct-summation $N$ -body integrator: Dynamics with softening

Cristián Maureira-Fredes<sup>1,2</sup> \* & Pau Amaro-Seoane<sup>3,4,5</sup>

<sup>1</sup> *Max Planck Institute for Gravitational Physics (Albert-Einstein-Institut), D-14476 Potsdam, Germany.*

<sup>2</sup> *Universidad Técnica Federico Santa María, Avenida España 1680, Valparaíso, Chile.*

<sup>3</sup> *Institut de Ciències de l'Espai (CSIC-IEEC) at Campus UAB, Carrer de Can Magrans s/n 08193 Barcelona, Spain.*

<sup>4</sup> *Institute of Applied Mathematics, Academy of Mathematics and Systems Science, CAS, Beijing 100190, China.*

<sup>5</sup> *Kavli Institute for Astronomy and Astrophysics, Beijing 100871, China.*

draft 27 January 2018

## ABSTRACT

A wide variety of outstanding problems in astrophysics involve the motion of a large number of particles under the force of gravity. These include the global evolution of globular clusters, tidal disruptions of stars by a massive black hole, the formation of protoplanets and sources of gravitational radiation. The direct-summation of  $N$  gravitational forces is a complex problem with no analytical solution and can only be tackled with approximations and numerical methods. To this end, the Hermite scheme is a widely used integration method. With different numerical techniques and special-purpose hardware, it can be used to speed up the calculations. But these methods tend to be computationally slow and cumbersome to work with. We present a new GPU, direct-summation  $N$ -body integrator written from scratch and based on this scheme, which includes relativistic corrections for sources of gravitational radiation. GraviDy has high modularity, allowing users to readily introduce new physics, it exploits available computational resources and will be maintained by regular updates. GraviDy can be used in parallel on multiple CPUs and GPUs, with a considerable speed-up benefit. The single GPU version is between one and two orders of magnitude faster than the single CPU version. A test run using 4 GPUs in parallel shows a speed up factor of about 3 as compared to the single GPU version. The conception and design of this first release is aimed at users with access to traditional parallel CPU clusters or computational nodes with one or a few GPU cards.

**Key words:** Methods: numerical – Stars: Kinematics and dynamics – Celestial Mechanics

## 1 MOTIVATION

The dynamical evolution of a dense stellar system such as e.g. a globular cluster or a galactic nucleus has been addressed extensively by a number of authors. For Newtonian systems consisting of more than two stars we must rely on numerical approaches which provide us with solutions that are more or less accurate. In this sense, one could make the following coarse categorisation of integration schemes for pure stellar dynamics: those which are particle-based and those which are not. In the latter, the system is treated as a continuum, so that while we know the general properties of the stellar system such as the mean stellar density, of the average velocity dispersion, we do not have informa-

tion about specific orbits of stars. To this group, belongs direct integration of the Fokker-Planck equation (Inagaki & Wiyanto 1984; Kim et al. 1998) or moments of it (Amaro-Seoane et al. 2004; Schneider et al. 2011), including Monte Carlo approaches to the numerical integration of this equation (Spitzer & Hart 1971). A particle-based algorithm, however, assumes that a particle is tracing a star, or a group of them. In this group, the techniques go back to the early 40's and involved light bulbs (Holmberg 1941). The first computer simulations were performed at the Astronomisches Rechen Institut, in Heidelberg, Germany, by (von Hoerner 1960, 1963), using 16 and 25 particles. These first steps led to the modern  $N$ -body algorithms.

We can distinguish two types of  $N$ -body algorithms: the so-called collision-less, where a star just sees the background potential of the rest of the stellar system (e.g. the

\* E-mail: Cristian.Maureira.Fredes@aei.mpg.de (CM)

Barnes-Hut treecode or the fast multipole method Barnes & Hut 1986; Greendard 1987, which scale as  $O(N \log N)$  and  $O(N)$ , with  $N$  the particle number, respectively), and the more expensive collisional one, or “direct-summation”, in which one integrates all gravitational forces for all stars to take into account the graininess of the potential and individual time steps, to avoid large numerical errors. This is important in situations in which close encounters between stars play a crucial role, such as in galactic nuclei and globular clusters, because of the exchange of energy and angular momentum. The price to pay however is that they typically scale as  $O(N^2)$ .

A very well known example is the family of direct-summation NBODY integrators of Aarseth (see e.g. Aarseth 1999; Spurzem 1999; Aarseth 2003)<sup>1</sup> or also KIRA (see Portegies Zwart et al. 2001a)<sup>2</sup>. The progress in both software and hardware has reach a position in which we start to get closer and closer to simulate realistic systems.

However, the scaling  $O(N^2)$  requires supercomputers, such as traditional Beowulf clusters, which requires a parallelisation of the code, such as the version of NBODY6 developed by Spurzem and collaborators, NBODY6++<sup>3</sup> (Spurzem 1999), or special-purpose hardware, like the GRAPE (short for GRAVity PipE<sup>4</sup>) system. The principle behind GRAPE systems is to run on a special-purpose chip the most time consuming part of an  $N$ -body simulation: the calculation of the accelerations between the particles. The remainder is calculated on a normal computer which serves as host to the accelerator board(s) containing the special purpose chips. Such a system achieves similar or even higher speeds than implementations of the  $N$ -body problem on supercomputers (see e.g. Taiji et al. 1996; Makino & Taiji 1998; Makino 1998; Fukushima et al. 2005).

On the other hand, modern graphics processing units (GPUs) offer a very interesting alternative. They have been mostly used in game consoles, embedded systems and mobile phones. They were originally used to perform calculations related to 3D computer graphics. Nevertheless, due to their highly parallel structure and computational speed, they can very efficiently be used for complex algorithms. This involves dealing with the parallel computing architecture developed by NVIDIA<sup>5</sup>, the Compute Unified Device Architecture (CUDA). This is the main engine in NVIDIA GPUs, and it has been made accessible to developers via standard programming languages, such as C with NVIDIA extensions compiled thanks to a PathScale Open64 C compiler. This is what allows us to create binary modules to be run on the GPUs. Another option is Open Computing Language (OpenCL)<sup>6</sup>, which offers a framework to write parallel programmes for heterogeneous systems, including also computational nodes with field-programmable gate arrays (FPGAs), digital signal processors (DSPs), among others. CUDA, and also OpenCL are “the doors” to the native instruction set and memory of the parallel elements in the

GPUs. This means that these can be handled as open architectures like CPUs with the enormous advantage of having a parallel-cores configuration. More remarkably, each core can run *thousands* of processes at the same time. We selected CUDA over OpenCL, because our systems are equipped with NVIDIA GPUs, even though we note that OpenCL has shown similar performance to CUDA in  $N$ -body simulations (Capuzzo-Dolcetta & Spera 2013).

There has been recently an effort at porting existing codes to this architecture, like e.g. the work of Portegies Zwart et al. (2007); Hamada & Iitaka (2007); Belleman et al. (2008) on single nodes or using large GPU clusters (Berczik et al. 2011; Nitadori & Aarseth 2012; Capuzzo-Dolcetta et al. 2013) and recently, the work by Berczik et al. (2013) using up to 700 thousand GPU cores for a few million bodies simulation with the  $\phi$ -GPU<sup>7</sup> code, which reached in their work about the half of the peak of the new Nvidia Kepler K20 cards.

Large-scale (meaning number of particles) simulations have recently seen an important improvement with the work of Wang et al. (2015, 2016). In his more recent work of 2016, Wang and collaborators integrated systems of one million bodies in a globular cluster simulation, using from 2,000 to 8,600 hours of computing time.<sup>8</sup>

In this paper we present the initial version of GRAVIDY (GRAVitational DYNAMICS), a highly-modular, direct-summation  $N$ -body code written from scratch using GPU technology ready to integrate a pure dynamical gravitational system. In section 2 we present in detail the structure of the code, the most relevant and innovative parts of the algorithm, and their implementation of the scheme in the idiom of GPU computing. In section 3 we check our code with a series of well-known tests of stellar dynamics for a dense stellar system and evaluate global dynamical quantities and we also evaluate the performance of the GPU version against the CPU one. In section 5 we present the implementation of the relativistic corrections, and a set of tests. In section 6 we summarise our work and give a short description of the immediate goals that will be described in upcoming publications.

*We have decided to focus on single-node clusters (meaning one or more GPU cards embedded in a host PC) and traditional multi-CPU clusters (e.g. Beowulf clusters), since this setup is more common to most users who aim to run middle-scale simulations. In the appendices we give a succinct description on how to download the code, how to compile it, and the structure of the data. We also include a set of python tools to analyse the results. Moreover, we also introduce a simple visualisation tool based on OpenGL, which can provide us with information sometimes difficult to obtain with two-dimensional graphics. In particular, we have made a significant effort in documentation and modularity, since it is our wish that the code is used, shaped and modified at will.*

<sup>1</sup> All versions of the code are publicly available at the URL <http://www.ast.cam.ac.uk/~sverre/web/pages/nbody.htm>

<sup>2</sup> <http://www.sns.ias.edu/~starlab/>

<sup>3</sup> Available at this URL <http://silkkroad.bao.ac.cn/nb6mpi>

<sup>4</sup> <http://grape.c.u-tokyo.ac.jp/grape>

<sup>5</sup> <http://www.nvidia.com>

<sup>6</sup> <https://www.khronos.org/opencl/>

<sup>7</sup> <ftp://ftp.mao.kiev.ua/pub/berczik/phi-GPU/>

<sup>8</sup> This impressive achievement was rewarded with a bottle of Scotch whisky (not whiskey), kindly and generously offered to him by Douglas Heggie during the excellent MODEST 15-S in Kobe.

## 2 THE CURRENT ALGORITHM

### 2.1 The integration scheme

In this section we give a very brief introduction to the numerical  $N$ -body problem. We refer the reader to e.g. Aarseth (2003); Heggie & Hut (2003) or the excellent on-line course “The art of computational science”<sup>9</sup>. The evolution of an  $N$ -body system is described by the second order ordinary differential equation

$$\ddot{\mathbf{r}}_i = -G \sum_{\substack{j=1 \\ j \neq i}}^N m_j \frac{(\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^3}, \quad (1)$$

where  $G$  is the gravitational constant,  $m_j$  is the mass of the  $j$ th particle and  $\mathbf{r}_j$  the position. We denote vectors with bold fonts. The basis of the problem is purely dynamical, because the orbital evolution is determined exclusive by the gravitational interaction.

The total energy of the system is a useful quantity to keep track of every time step in the integration. It is given by the expression

$$E = \frac{1}{2} \sum_{i=1}^N m_i \mathbf{v}_i^2 - \sum_{i=1}^N \sum_{j>i}^N \frac{Gm_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|}, \quad (2)$$

where  $\mathbf{v}_i$  is the velocity of the particle  $i$ .

To numerically integrate the system of equations we adopt the 4th-order Hermite integrator (H4 from now onwards) presented in Makino (1991); Makino & Aarseth (1992) (and see also Aarseth 1999, 2003). H4 is a scheme based on a predictor-corrector scenario, which means that we use an extrapolation of the equations of motion to get a predicted position and velocity at some specific time. We then use this information to get the new accelerations of the particles, later we correct for the predicted values using interpolation based on finite differences terms. One can use polynomial adjustment in the gravitational forces evolution among the time because the force acting over each particle changes smoothly (which is the reason why adding a very massive particle representing e.g. a super massive black hole will give you sometimes a headache). To advance the system to the following integration time we approximate the equations of motion with an explicit polynomial. This prediction is less accurate, but it is improved in the corrector phase, which consist of an implicit polynomial that will require good initial values to scale to a good convergence.

This is a fourth-order algorithm in the sense that the predictor includes the contributions of the third-order polynomial, and after deriving the accelerations, adds a fourth-order corrector term. In the remaining of this paper we focus on the implementation of the scheme into our GPU (and CPU) code and how to maximise all of the computational resources available. For a detailed description of the idea behind H4, we refer the reader to the article in which it was presented for the first time, (Makino & Aarseth 1992).

An advantage of the choice for H4 is that we can use the family of Aarseth’s codes (among others) as a test-bed for our implementation. These codes –some of which adopt

H4, but not all of them– have been in development for more than 50 years. The codes are public and have been widely used and validated, improved and checked a number of times by different people, they have been compared to other codes and even observational data. In this regard, to test our implementation and parallelisation of H4, the access to the sources of the codes is an asset.

### 2.2 Numerical strategy

A main goal in the development of GRAVIDY is its *legibility*. We have focused in making it easy to read and modify by other users or potential future developers without compromising the computational performance of the algorithm. This means that we have made a significant effort in keeping a clear structure in the source code so that, in principle, it can be well understood by somebody who has not previously worked with it with relatively little effort. The modularity of the code should allow new users to easily implement new physics or features into it or adapt it to the purposes they seek. It is unfortunately easy –at least to a certain extent– to miss either clarity in coding or performance, when trying to have both in a code. For instance, if we want to obtain the best performance possible, one has to use low-level instructions that for an outside user might result into something difficult to understand when reading or trying to modify the source code. On the other hand, name conventions for files, functions and variables might become a burden to certain applications.

While most existing  $N$ -body codes have achieved certain balance between the two to some degree, it is difficult to adapt them to new architectures and technology to boost their performance. For the development of GRAVIDY, we have followed the next steps:

- Serial Implementation of the initial version,
- Profiling and assessment of the integrator,
- Algorithm classification and finding the hot-spots,
- Optimisation of the bottlenecks.

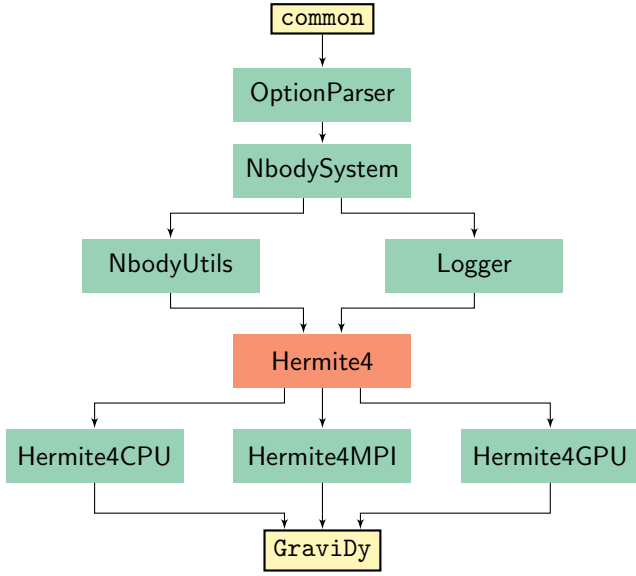
### 2.3 Particular choices

**Object oriented programming:** Object oriented programming (OOP) is a powerful paradigm that allows us to program an algorithm as objects interactions. In GRAVIDY, we use OOP. The reason beneath it is related to our parallelisation scheme, which is described below, more concretely with the data structure we have chosen.

We have mainly two possible choices for data structures: classes with arrays, or Arrays of Objects, which follows the basic idea of Struct of Arrays (SoA) and Array of Structs (AoS). For GRAVIDY we have chosen *classes* with arrays for the main units of the program structure. It is a good strategy to minimise the data transfer between Host and Device, so as to avoid having large communication times.

It is not required to update the forces of all the particles, so that we encapsulate the information of the active particles, and then we transfer the AoS to the GPU. All the remaining attributes of the bodies (i.e. those not transferred to the GPU) are just class-members (arrays), and need to be in the host CPU. An example of this could be large linear arrays, such as the time steps of the particle.

<sup>9</sup> <http://www.artcompsci.org/>



**Figure 1.** Class diagram of the code that shows the hierarchy of the application structure (GRAVIDY).

**Class distribution:** Since our code is using OOP, we describe a brief interaction between the classes in Fig. 1. The main header, `common.hpp`, contains the definition of the constants, structures, macros, etc. The idea behind this model is to easily be able to add more features in upcoming versions of our code, from new utilities functions to new integration schemes.

Every class is in charge of a different mechanism, from getting the integration options from command-line, to the different integration methods using parallelism or not<sup>10</sup>.

**Double-precision (DP) over Single-precision (SP):** Using DP or SP in  $N$ -body codes has been already addressed by different authors in the related literature (see e.g. Hamada & Itaka 2007; Nitadori 2009; Gaburov et al. 2009). Using DP is not the best scenario for GPU computing, because there is a decrease factor in the maximum performance that a code can reach. We can reach only half of the theoretical maximum performance peak, which depends on each individual card: for example, the NVIDIA Tesla C2050/M2050 has a peak of the processing power in GFLOPs 1030.46 with SP, but only 515.2 with DP.

We choose DP for a more accurate numerical representation, because it provides us a simple way of getting better energy conservation, at the expenses of performance. There are different approaches, like the mixed-precision, (Aarseth 1985), and pseudo DP (Nitadori 2009, currently used in the code  $\phi$ -GPU, Berczik et al. 2011). These offer a relatively more accurate representation (compared to SP) without a big impact in performance.

## 2.4 The implementation scheme

These are the steps that GRAVIDY follows when running a simulation:

- (i) Memory allocation of the CPU and GPU arrays.
- (ii) Initialisation of the variables related to the integration.
- (iii) Copy the initial values of positions, velocities and masses of the particles to the GPU to calculate the initial system energy, and calculate the initial acceleration and its first time derivative, the so-called “jerk”. The cost of this force calculation is  $O(N^2)$ .
- (iv) Copy the initial forces from the GPU to CPU.
- (v) Find the particles to move in the current integration time,  $N_{\text{act}}$ , with a cost  $O(N)$ .
- (vi) Save the current values of the forces, to use them in the correction step, with a cost  $O(N)$ .
- (vii) Integration step:

- (a) Predict the particle’s positions and velocity up to the current integration time, with cost  $O(N)$ .
- (b) Copy of the predicted positions and velocities of all the particles from the CPU to the GPU.
- (c) Update the  $N_{\text{act}}$  particles on the GPU, which is explained in detail in section 2.5.

(1) Copy the  $N_{\text{act}}$  particles to a temporary array on the GPU.

(2) Calculate the forces between the particles on the GPU, with a cost  $O(N_{\text{act}} \cdot N)$ .

(3) Reduce forces on the GPU.

(4) Copy the new forces from the GPU to the CPU.

(d) Correct the position and velocity of the  $N_{\text{act}}$  updated particles on the CPU,  $O(N_{\text{act}})$ .

(e) Copy the positions and velocities of the corrected  $N_{\text{act}}$  particles from the CPU to the GPU.

GRAVIDY adheres to the usual good practises of the beginning of the development of every direct-summation  $N$ -body code:

- *Direct-summation*, also known as particle-particle strategy, This approach is the simplest way to address the task of calculating the exerted force by all the  $N-1$  bodies on a single body that we need to update at certain time step. This brute-force procedure has an order  $O(N^2)$ , which represents the bottleneck of the algorithm.

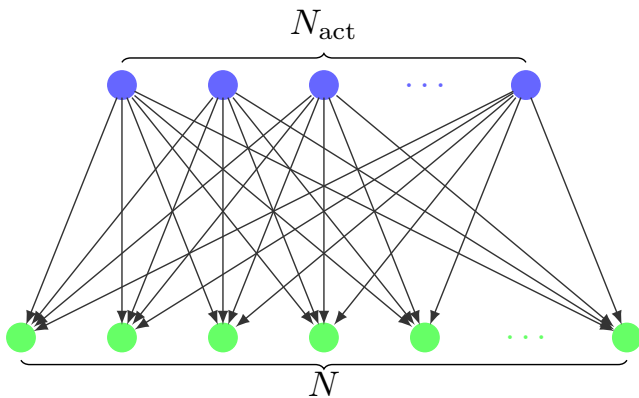
- *Softened point-mass potential*, as an alternative in this version of the code to a proper close encounter regularisation. All particles are represented by a dimensionless point mass. We introduce a softening parameter ( $\epsilon$ ) in the distance calculation between two bodies while we get the new forces,

$$\ddot{\mathbf{r}}_i = -G \sum_{\substack{j=1 \\ j \neq i}}^N \frac{m_j}{(r_{ij}^2 + \epsilon^2)^{3/2}} \mathbf{r}_{ij}, \quad (3)$$

so as to handle the situation in which two bodies get closer.

- *Block time steps*, It is not straightforward to have an  $N$ -body code using individual time steps in parallel computing, because the idea behind massive parallelism is to perform the same task on different data chunks. We use the block time steps algorithm (Press 1986), to update group particles simultaneously. This scheme has been adopted by a number of authors (Portegies Zwart et al. 2001b; Hut 2003;

<sup>10</sup> For more information, please refer to the code documentation



**Figure 2.** Relation between the particles which will be updated in a certain integration time ( $N_{\text{act}}$ ) and the whole set of particles ( $N$ ). The relation between the active particles and the others is  $N_{\text{act}} \ll N$  in non-synchronisation times.

Aarseth 1999, 2003; Harfst et al. 2008; Nitadori & Aarseth 2012).

The algorithm uses a H4 to integrate the evolution. The description of all the equations for each step is presented in Makino & Aarseth (1992)<sup>11</sup>

## 2.5 The parallelisation scheme

As we have already mentioned, the bottleneck of any  $N$ -body code is the force calculation. In this respect, GRAVIDY is not different and a quick performance test to get the profile of our serial code yields almost 95% of the execution time in this calculation. We hence introduce a parallelisation scheme, which we discuss in detail now.

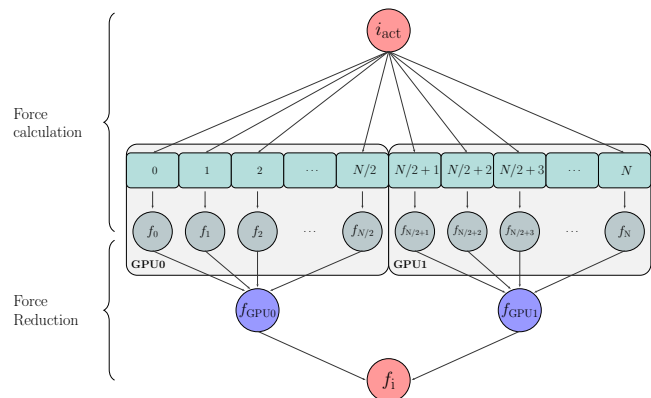
GRAVIDY is based on a direct-summation H4 integrator and uses block time steps, so that in the force update process we have a nested loop for every  $i$ -active particle (which we will refer to from now with the subscript “act”). This means that for every particle which needs to be updated we have a loop run on the whole set of particles of our system to check whether  $j$ -particle is interacting with the  $i$ -particle.

The whole process scales with the amount of  $i$ -particles, as we can see in Figure 2.

We then need to parallelise the loop corresponding to each of the  $i$ -particles. For each of them we circulate through all of the  $j$ -particles, and this is the process which needs to be parallelised. Although this is in principle a straightforward scheme, since we focus on GPUs, we run into the following issues:

(i) A GPU can launch a large number of threads, easily up to thousands of them. In our scenario, however, the number of active particles  $N_{\text{act}}$  is very small compared to the total amount of particles ( $N$ ). This has an impact on the performance: we do not use all available threads, we are integrating a grid of  $N_{\text{act}} \times N$  forces. When the number of active particles is very low our *occupancy* will be bad.

<sup>11</sup> Eq. (2) has a typo in the sign of the second term in the sum of  $\dot{\mathbf{a}}_{i,1}$ .



**Figure 3.** parallelisation scheme to split the  $j$ -loop instead of the  $i$ -loop. Two GPUs are depicted to represent how the code works with multiple devices. In this case, we have two sections, the first is to calculate the force interactions of the  $i$ -particle with the whole system but by different threads (upper part). Then a reduction, per device, is necessary to get the new value for the  $i$ -particle force ( $f_i$ ).

(ii) Contrary, in the case in which we have to move all particles, we will have an  $O(N^2)$  parallelism, which maximises the GPU power. In this case, however, the *memory bandwidth* is the limitation factor, since every particle requires all information about all other  $N - 1$  particles.

It is better to have all particles handled by the GPU, and not only the active ones, because even though this subgroup is smaller, or even much smaller, it is more efficient from the point of view of the GPU, since the *occupancy* is improved. The parallelisation happens at  $j$ -level (i.e. when calculating the forces between active particles with the rest of the system). This idea was first implemented by Nitadori (2009), and has proven to yield very good performance.

The main ideas behind the  $j$ -parallelisation is how force calculation is done and the summation of the forces (“reduction”):

- *Force calculation:* The interaction between the  $i$ -particle and the rest of the system is distributed among the GPU threads, which means that we launch  $N$  threads, and each of them calculates its contribution with the  $i$ -particle. After this calculation, we have an array where each element contains the contributions all the particles. $j$  This corresponds to the upper part of Fig.(3), which illustrates a set-up of two GPUs. After the force calculation we end up with an array containing the information about the forces for all particles.

- *Force reduction:* In the lower part of the same Fig. we depict the summation of all of these forces, which is also performed in parallel, so that we use the blocks distribution of the GPU for this task.

## 3 THE THREE FLAVOURS OF GRAVIDY: TESTS

Thanks to the fact that there is a number of codes implementing similar approaches to ours, we are in the position of running exhaustive tests on GRAVIDY. Indeed, the global

dynamics of a dense stellar system (typically an open cluster, because of the limitation in the number of particles we can integrate) has been addressed numerically by a large number of authors in the field of stellar dynamics. Therefore, we have decided to focus on the dynamical evolution of a globular cluster with a single stellar population. We present in this section a number of tests to measure the performance and the accuracy of the three versions of GRAVIDY which we present using different amount of particles. Our goal is to be able to offer an Open Source code that fits different needs and requirements. This is why this first release of GRAVIDY offers three different choices, which are general enough for different users with different hardware configurations. These are:

(i) **The CPU version** consists in the more basic implementation in this work, a CPU version. I.e. This version uses OpenMP and is intended for a system without graphic processing units, but with many cores. This flavour can be used for debugging purposes by disabling the OpenMP directives (`#pragma omp`). This is the basis for our further development of the code.

(ii) **The MPI version** is virtually the same serial implementation, but with OpenMPI directives added to improve the performance of the hot-spots of the algorithm, in particular the force and energy calculation. In this case we use the MPI library, and hence it can be run on a single machine using a certain amount of cores as “slave” processes or on a large cluster with separated machines as slaves.

(iii) **The GPU version** discards all CPU usage and only relies on the GPU to integrate all gravitational interactions. As we mention later, we tried to use CPU combined with GPU, but we did not see any benefit in it, and the approach was hence neglected. We use CUDA to be able to interact with NVIDIA graphics processing units. The code is designed to detect the amount of present GPUs and use all of them, unless otherwise required by the user. This means that this version can use in a parallel way as many GPU cards as the host computer can harbour in a very simple and efficient way. The communication between the different GPU cards in the host computer is internal and run through Peripheral Component Interconnect Express (PCIe), a high-speed serial computer expansion bus standard, so that the data flows rapidly because of the low overhead.

The specifications of the hardware (CPU, GPU and available RAM) and operating systems we used are summarised in table 1.

### 3.1 Initial conditions and $N$ -body units

For all of our tests we choose an equal-mass Plummer sphere (Plummer 1911) for the sake of comparison with other codes. We choose standard  $N$ -body units (NBU, hereon) for the calculations and in the resulting output (Hénon 1971; Heggie & Mathieu 1986). This means that

- The total mass of the system is 1:  $\sum_{i=0}^N m_i = 1$ .
- The gravitational constant ( $G$ ) is set to 1:  $G = 1$ .
- The total energy of the system is equal to  $-0.25$ :  $E_{\text{tot}} = K + U = -0.25$ , with  $K$  and  $U$  the total kinetic and potential energy of the system, respectively.
- The virial radius is set to  $\approx 1$ .

<b>System A</b>		<b>datura</b> (165 nodes)
CPU	Intel(R) Xeon(R) CPU X5650 @ 2.67GHz (24 cores)	
GPU		<i>none</i>
RAM		24 GB
OS		Scientific Linux 6.0
<b>System B</b>		<b>gpu-01</b> (1 node)
CPU	Intel(R) Xeon(R) CPU E5504 @ 2.00GHz (4 cores)	
GPU	4 x Tesla M2050 @ 575 Mhz (448 cores)	
RAM		24 GB
OS		Scientific Linux 6.0
<b>System C</b>		<b>krakatoa</b> (1 node)
CPU	AMD Opteron 6386SE @ 2.8 GHz (32 cores)	
GPU	2 x Tesla K20c @ 706 MHz (2496 cores)	
RAM		256 GB
OS		Debian GNU/Linux 8
<b>System D</b>		<b>sthelens</b> (1 node)
CPU	Intel(R) Xeon(R) CPU E5-2697v2 (IvyBridge) @ 2.7GHz (24 cores)	
GPU	2 x Tesla C2050 / C2070 @ 1.15 Ghz (448 cores)	
RAM		256 GB
OS		Debian GNU/Linux 8

**Table 1.** Specification of the different systems of the Albert Einstein Institute used for the tests.

The Plummer spheres have a fixed half-mass radius of 0.8 and a Plummer radius of 0.6.

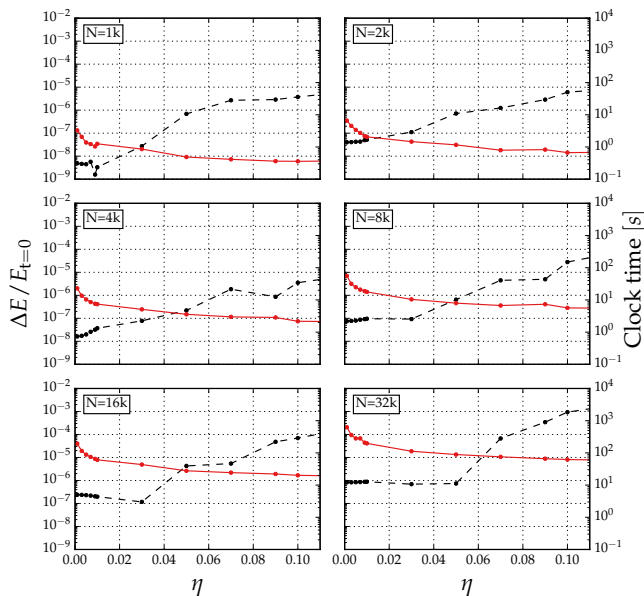
We used the code by Küpper et al. (2011) (McLuster) to generate all the initial conditions for the test we performed on the current work.

### 3.2 Accuracy, performance and speed

For GRAVIDY, as we have seen, we have chosen a H4 integrator. The numerical error introduced scales hence as  $O(\Delta t^4)$  assuming a shared time step, which means that the previous is true *only* if all particles are updated at every integration step. Since we use a block time step scheme, certain groups of particles share a time step value, but not all of them. Thanks to this approach, the numerical error which we reach in our integrations is slightly less than the value mentioned previously.

A free parameter,  $\eta$ , was introduced in Makino & Aarseth (1992) responsible for determining the calculation of every time step of the system, from the initial calculation to the update after every iteration. Hence, so as to assess an optimal value for it, we perform different tests to find a balance between a good energy conservation and a minimum wall clock time. We explore values between 0.001 and 0.1 integrating a series of systems with  $N$  ranging between 1024 to 32768, for convenience<sup>12</sup>, and up to 2 NBU. We show the results in Fig.(4) performed on System B of Tab. (1). For small values of  $\eta$ , the cumulative energy error approximately stays constant, because the error is small enough to leave accuracy in hands of the integrator scheme and the hardware. Increasing  $\eta$  leads to larger errors. This is particularly evident when we use systems with a larger number of particles. The system with  $N = 32768$  particles, and a  $\epsilon = 10^{-4}$ , achieves  $\Delta E/E_0 \approx 10^{-3}$  for  $\eta = 0.1$ , while it is as low as  $\Delta E/E_0 \approx 10^{-6}$  for the same value and 1024 particles.

<sup>12</sup> Any number of particles can be also handle properly, not necessarily powers of 2.

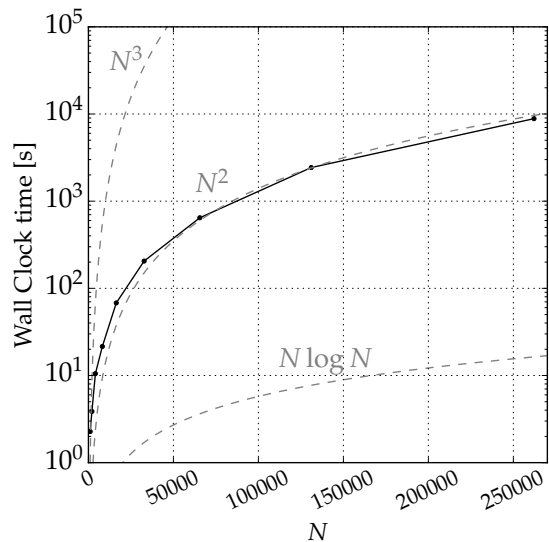


**Figure 4.** Cumulative energy error (dashed black line) and wall clock time (solid red line) in function of  $\eta$  for six different systems consisting of a Plummer sphere with  $N = 1, 2, 4, 8, 16, 32$  k particles, with  $k := 1000$ , from the top to the bottom, left to right. The integration corresponds to one time unit, namely from  $t = 1$  to  $t = 2$  in the wall clock time analysis, and for  $t = 2$  in the energy error calculation. The reason for choosing the elapse between 1 and 2 is to get rid of any initial numerical error at the simulation startup, from 0 to 1. All tests have been performed on System B of Tab. (1).

In the same figure we describe the performance in function of  $\eta$  by using the wall clock time in seconds for the code to reach one NBU for the same values of the parameter. We can see that the value of  $\eta$  is inversely proportional to the time, since increasing its value results in decreasing the execution time. When we increase  $\eta$  we implicitly increase the time step of every particle, so that one unit of time is reached sooner. We find that a value of about 0.01 is the best compromise for most of our purposes, yielding an accuracy of about  $\Delta E/E_0 = 10^{-7}$  in most of the cases.

To measure the execution speed of our code we perform a set of tests by integrating the evolution for one NBU of a Plummer sphere with different particle numbers, ranging from  $N = 1024$  to  $N = 262144$ . For the analysis, we choose the time starting at  $t = 2$  and finishing at  $t = 3$ , since the first time unit is not representative because the system can have some spurious numerical behaviour resulting from the fact that it is not *slightly* relaxed. When testing the parameters  $\eta$  and  $\epsilon$ , we picked the time starting at  $t = 1$  and finishing at  $t = 2$  because we wanted to understand their impact not right at the beginning of the simulation. Now we allow the system to further relax so as to obtain a more realistic system. In particular, the distribution time steps drifts away from the initial setup.

We display the wall clock time of each integration in Fig. (5). We also display reference curves for the powers of  $N^3$ ,  $N^2$  and  $N \log N$ , multiplied by different factors to adapt them to the figure. We see that GRAVIDY scales very closely



**Figure 5.** Wall clock time of integration from  $t = 1$  NBU up to  $t = 2$  NBU, using  $\eta = 0.01$  and  $\epsilon = 10^{-4}$  using different amount of particles on System C of Tab.(1).

as a power of 2. The deviations arise from the fact that not all particles are being updated at every time step.

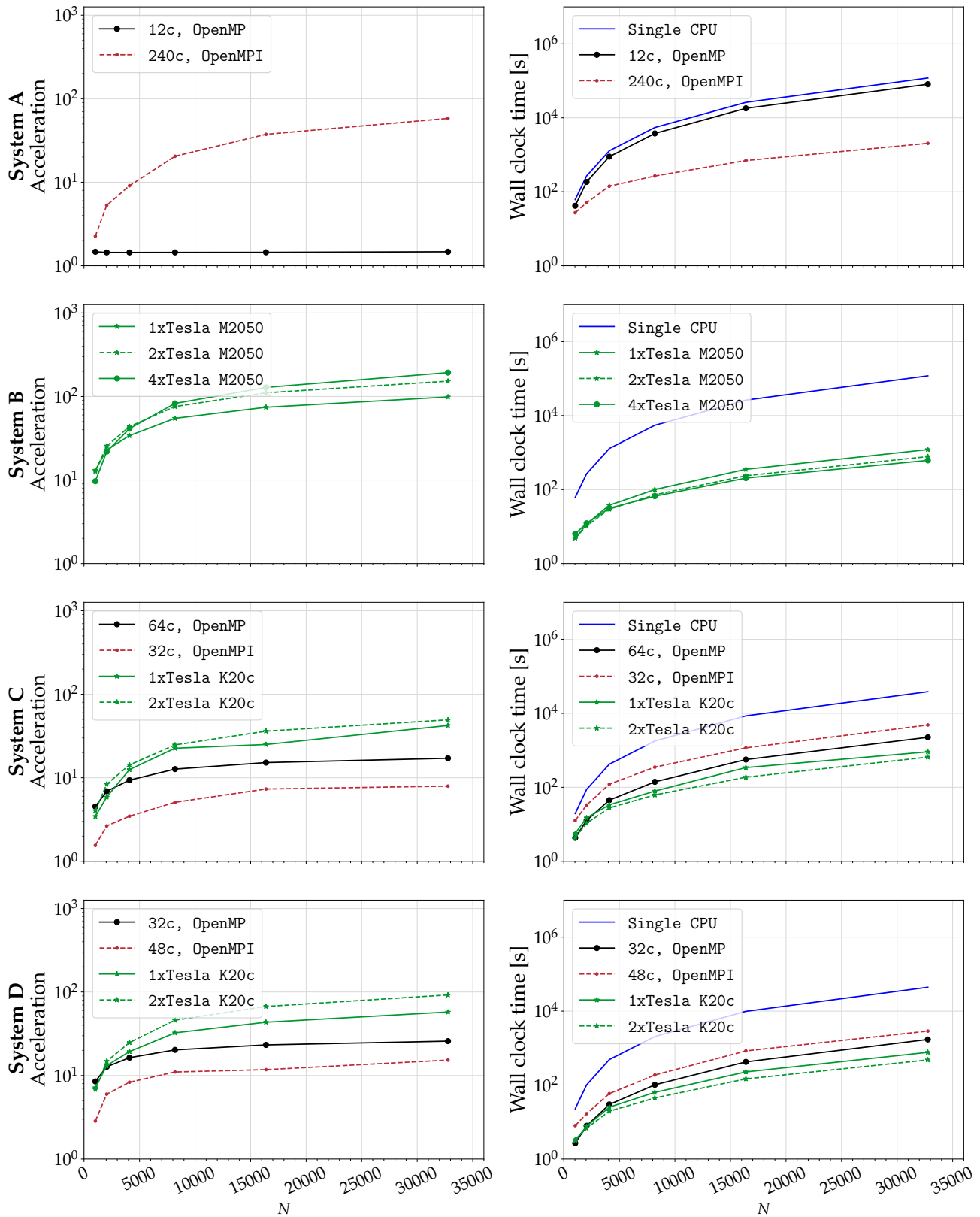
In Figure 6 we show the acceleration factor for all parallel scenarios as compared to the single-thread CPU case, which we use as a reference point. Due to the design of the code, the maximum performance is achieved with the larger particle number. The most favourable scenario for GRAVIDY is, as we can see in the figure, System B. The 4 GPUs available boost the performance up to a factor of 193 as compared with the single-thread CPU case. A similar speed up is achieved on System D, which reaches a factor of 92 for the 2 GPUs. The CPU-parallel version lies behind this performance: only reaching a factor of 58 for System A, using up to 240 cores.

### 3.3 Scaling of the three different flavours of the code

An obvious question to any user of a numerical tool is that of scaling. In this subsection we present our results for the three different versions of GRAVIDY of how wall clock time scales as a function of threads or cores, or what is the acceleration of the multiple-GPU version of the code in function of the particle number as compared with a single GPU run, which we use as reference point.

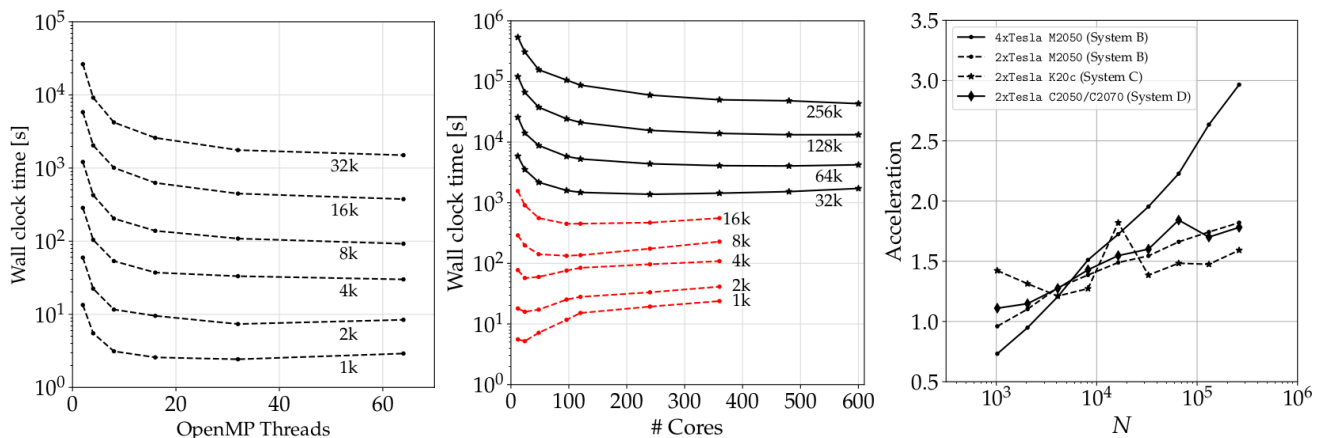
In Fig. (7) we depict this information for the CPU, MPI and GPU versions. We can see in the CPU version that for small amounts of particles, in particular for 2k and 1k, we have an increase in the execution time with more threads, contrary to what we would expect. This is so because the amount of parallelism is not enough and the code spends more time splitting data and synchronising threads than performing the task itself, a usual situation in tasks with a low level of computation.

The MPI version uses the same j-parallelisation idea from the GPU one. In this case the code splits the whole system to the amount of available slaves (be it cores or nodes), performs the force calculation and finally sums up



**Figure 6.** Acceleration factor and wall clock time for the different parallel versions of the integrator (see Tab. 1). The acceleration factor is normalised to the single CPU version (1 thread), up to  $T = 3[NBU]$ . For the CPU parallel version of the code, we give information about the number of cores with the letter “c”. The GPU-parallel cases display the information on the number of cards with multiplying numbers.





**Figure 7.** Performance of the three different flavours of GRAVIDY, as a function of the number of OpenMP threads, and number of cores and GPUs, for the CPU, MPI and GPU versions, respectively and from left to right. The integration corresponds to up to  $t = 2$  NBU. *Left panel:* The CPU version runs on a single node with different numbers of threads. The experiments were performed on system C of Tab.(1). *Mid panel:* The MPI version running on different numbers of cores, using up to 600 of them and particles, up to 262144. In this case we use system A of the same table. *Right panel:* The GPU flavour using different amount of devices in parallel and particles. We show the acceleration factor as compared to a single GPU run for three different setups with different GPU specifications, corresponding to systems B, C and D of the same table.

(“reduces”) the final forces for all active particles. This procedure was performed developing our own forces datatype operations and reduction, based on structures. This means that we define our own operations to be able to “sum” two forces (which are two three-dimensional arrays per particle). The simulations with small amount of particles (1k, 2k, 4k, 8k and 16k) are a clear example of a parallelisation “overkill”: using more resources than what is actually needed. Additionally, the communication process plays a role in scaling, which can be seen in the curves corresponding to these simulations for a number larger than 200 cores - the execution time increases instead of decreasing. On the other hand, large amount of particles (cases with 32k, 64k, 128k and 256k) show the expected behaviour, a better execution time with more nodes or cores. Surely this is not a solution for all simulations, since at some point the curves flatten.

The GPU version is a different scenario, since every device has its own capability, limitations and features that makes it difficult to compare their performances. For this reason we have decided to present the acceleration factor of every case normalised to a single-GPU run in the same system. This flavour of GRAVIDY should always have a better performance when increasing the particle number. Although having a good occupancy is in principle the ideal scenario in this case, it is not necessarily the best reference point to assess the efficiency of the CUDA kernels, because it is related to register uses, but also to the amount of redundant calculations and the arithmetic intensity. We show the acceleration factor of two and four **Tesla M2050** devices as compared to a single-GPU run which have hardware and performance differences<sup>13</sup> but they nonetheless reach a similar acceleration factor. We have access to two **Tesla K20c**, which have

more than the double peak performance in double precision floating point compared to the other mentioned models. The scaling between using one and two devices has a factor of 1.6.

Every GPU is a different device, so that in order to obtain a proper optimisation we need to first do a study in terms of kernel calls configuration. The current work present a fixed configuration of 32 threads per block, using a number of blocks corresponding to  $N/32$ . A deeper study on each GPU-device configuration is planned for future publication, where speeding up the first GPU implementation will be one of the main concerns.

#### 4 THE ROLE OF SOFTENING ON DYNAMICS

For the current version of GRAVIDY, and quoting Sverre Aarseth on a comment he got some years ago during a talk, “we have denied ourselves the pleasure of regularisation” (Kustaanheimo & Stiefel 1965; Aarseth & Zare 1974; Aarseth 1999, 2003). This means that the code resorts to softening, via the parameter  $\epsilon$ . This quantity can be envisaged as a critical distance within which gravity is, for all matters, nonexistent. This obviously solves the problem of running into large numerical errors when the distance between two particles in the simulation become smaller and smaller, because since they are 0-dimensional, this induces an error which grows larger and larger as they approach. This comes at a price, however. The relaxation time of the system is, approximately (see e.g. section on Two-body relaxation by Amaro-Seoane 2012),

without active cooling, while model C includes the active cooling and can be installed on any standard computer.

<sup>13</sup> The primary difference is that model M is designed for Original Equipment Manufacturer (OEM) for an integrated system,

$$t_{\text{rlx}} \sim N_* \frac{t_{\text{dyn}}}{\ln(d_{\text{max}}/d_{\text{min}})}. \quad (4)$$

In this equation  $d_{\text{min}}$  and  $d_{\text{max}}$  are the minimum and maximum impact parameters. In an unsoftened  $N$ -body problem they are of the order of  $d_{\text{min}} \approx Gm/\sigma^2$ , and the size of the cluster, respectively. In other words,  $d_{\text{min}} \cong R_{\text{cl}}/N$ , with  $R_{\text{cl}}$  the radius of the self-gravitating cluster, if the system is virialised, and  $d_{\text{max}}$  is of the the half-mass radius order. Now suppose the code uses a softening parameter  $\epsilon$ . If the value of  $\epsilon$  is smaller than  $d_{\text{min}}$ , then softening should play only a minor role in two-body relaxation, and the global dynamical evolution of the cluster must be similar to that of another cluster using regularisation. In the contrary case in which  $\epsilon > d_{\text{min}}$ , the relaxation time is artificially modified, as we can read from the last equation. The larger the quantity  $\ln(d_{\text{max}}/d_{\text{min}})$ , the more efficient is relaxation, and hence the shorter the relaxation time.

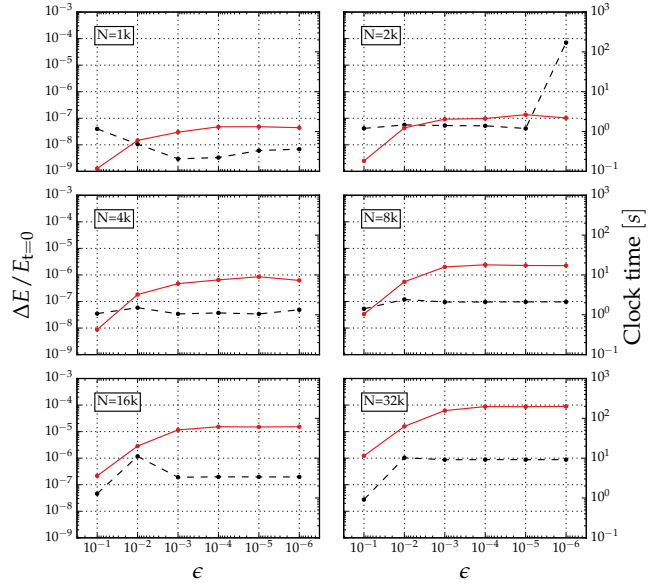
#### 4.1 “Best” value for the softening?

We perform a series of simulations to assess the relevance of  $\epsilon$  in the global dynamical evolution of an autogravitating stellar system. In Figure 8 we depict the energy error and wall clock time for six different particle numbers as a function of the softening. The lower its value, the faster the simulation. However, by using larger values of the softening, we must understand that we are evolving a system in which two-body deflections are not being taking into account. This is the most important aspect of two-body relaxation, and therefore a critical factor in the general evolution. Thus, the fundamental feature which drives the global evolution of the system is non-existing below larger and larger distances. In particular, the larger values correspond to about 10% of the virial radius of the system. From these panels it seems that a value of  $\epsilon \approx 10^{-4}$  is a good compromise *for this particular test that we are running in this example*. A good practice would be that the user tests different softening values for the case which is being addressed before making a decision for the softening. This choice is left for the user of the code, because we deem it difficult, if not impossible, to implement a self-regulating scheme in which the best value for the softening is calculated a priori.

#### 4.2 Core collapse

##### 4.2.1 Single-mass calculations

A good reference point to assess the global dynamical evolution of a dense stellar system is the core collapse of the system (see e.g. Spitzer 1987; Aarseth et al. 1974; Giersz & Spurzem 1994). We present here the evolution of the so-called “Lagrange radii” (the radii of spheres containing a certain mass fraction of the system) in Figure 9, for three representative values of the softening, the three upper panels, as calculated with GRAVIDY, and depict also the results of one calculation performed with NBODY6GPU (Nitadori & Aarseth 2012), the lower panel, which uses KS regularisation (Kustaanheimo & Stiefel 1965; Aarseth 2003). This can be envisaged as the “best answer”, which provides the reference point with which the other calculations should be



**Figure 8.** Cumulative energy error (dashed black line) and wall clock time (solid red line) using different values of the softening ( $\epsilon$ ). We integrate different amounts of particles  $N$  up to  $t = 2$  NBU. The wall clock time corresponds to the execution time between  $t = 1$  and  $t = 2$  NBU while the energy error is the one at  $t = 2$  NBU.

compared. In the figures we use the half-mass relaxation time, which we introduce as

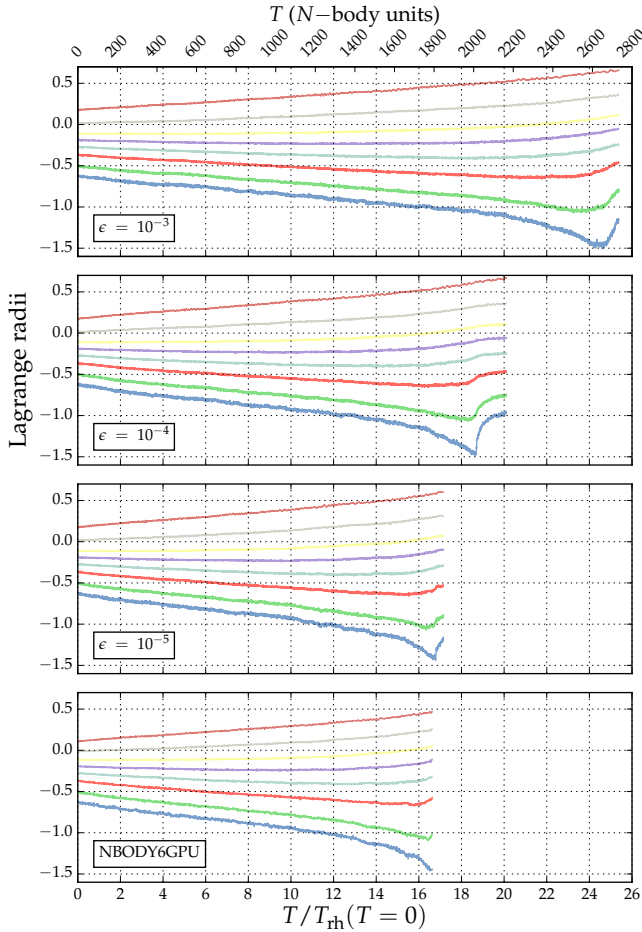
$$t_{\text{rh}} = 0.138 \left( \frac{N r_h^3}{Gm} \right)^{\frac{1}{2}} \frac{1}{\ln(\Lambda)}, \quad (5)$$

where  $N$  is the number of particles of the system,  $m$  the average mass of a star,  $r_h$  the half-mass radius, and  $\Lambda := \gamma N$ , with  $\gamma = 0.1$  the argument of the Coulomb logarithm.

From the panels we can easily see the impact of the softening parameter in the calculations: the collapse of the core is retarded for larger values. Our default choice for the softening,  $10^{-4}$  is just 2  $T_{\text{rh}}$  earlier than a NBODY6GPU calculation that we performed to compare with our code.

Another way of looking at the core collapse is in terms of energy. In Figure 10 we display the evolution of the energy for the same systems of Figure 9. As the collapse develops, the average distance between particles becomes smaller and smaller. There is an obvious correlation between the conservation of energy and the value of the softening. The transition between a fairly good energy conservation and a bad one happens more smoothly for larger and larger values of the softening, since the error has been distributed since the beginning of the integration. This means that, the smaller the value of the softening, the more abrupt the transition between the good and bad energy conservation, which leads to a big jump for the lowest value,  $10^{-5}$ . We stop the simulations at this point because of the impossibility of GRAVIDY to form binaries, the main way to stop the core collapse.

As discussed previously, and as we can see in Figures (10, 9), the introduction of softening in the calculations has an impact on the global dynamical behaviour of the system. We find factors of 1.001, 1.08 and 1.55 of delay to reach



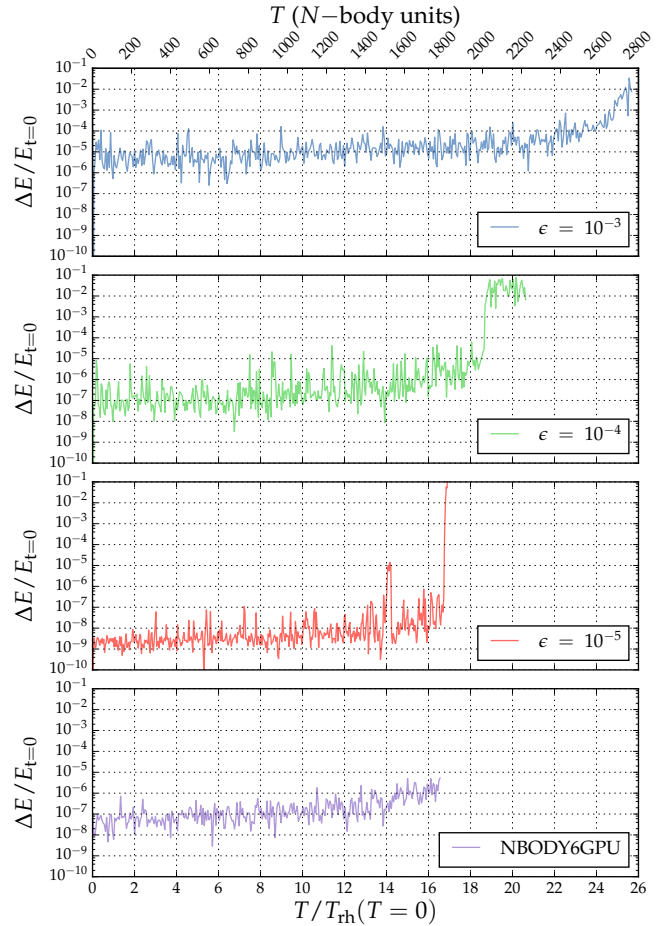
**Figure 9.** Comparison of the Lagrange radii of a Plummer Sphere with  $N = 8192$  particles, using different values of  $\epsilon$  (softening) for GRAVIDY and the NBODY6GPU code, from upper to bottom. The mass percentages are 0.5, 1, 2, 3, 4, 5, 6, 75 and 90% of the total mass, from the bottom to the upper part of each plot. The core collapse is reached at  $\approx 24, 18$  and  $16 T_{\text{rh}}$  for  $\epsilon = 10^{-3}, 10^{-4}$  and  $10^{-5}$  respectively. The half-mass relaxation time for this system is  $T_{\text{rh}} = 112.186[NBU]$  The NBODY6GPU code does not include a softening parameter, and treat binary evolution with a KS-regularisation.

the core collapse for the softening values  $\epsilon = 10^{-5}$ ,  $\epsilon = 10^{-4}$  and  $\epsilon = 10^{-3}$ , respectively.

The NBODY6GPU simulation was run on a different system, using a GeForce GTX 750 (Tesla M10) GPU, which is why we compared with the overall system evolution instead of the wall clock time.

#### 4.2.2 Calculations with a spectrum of masses

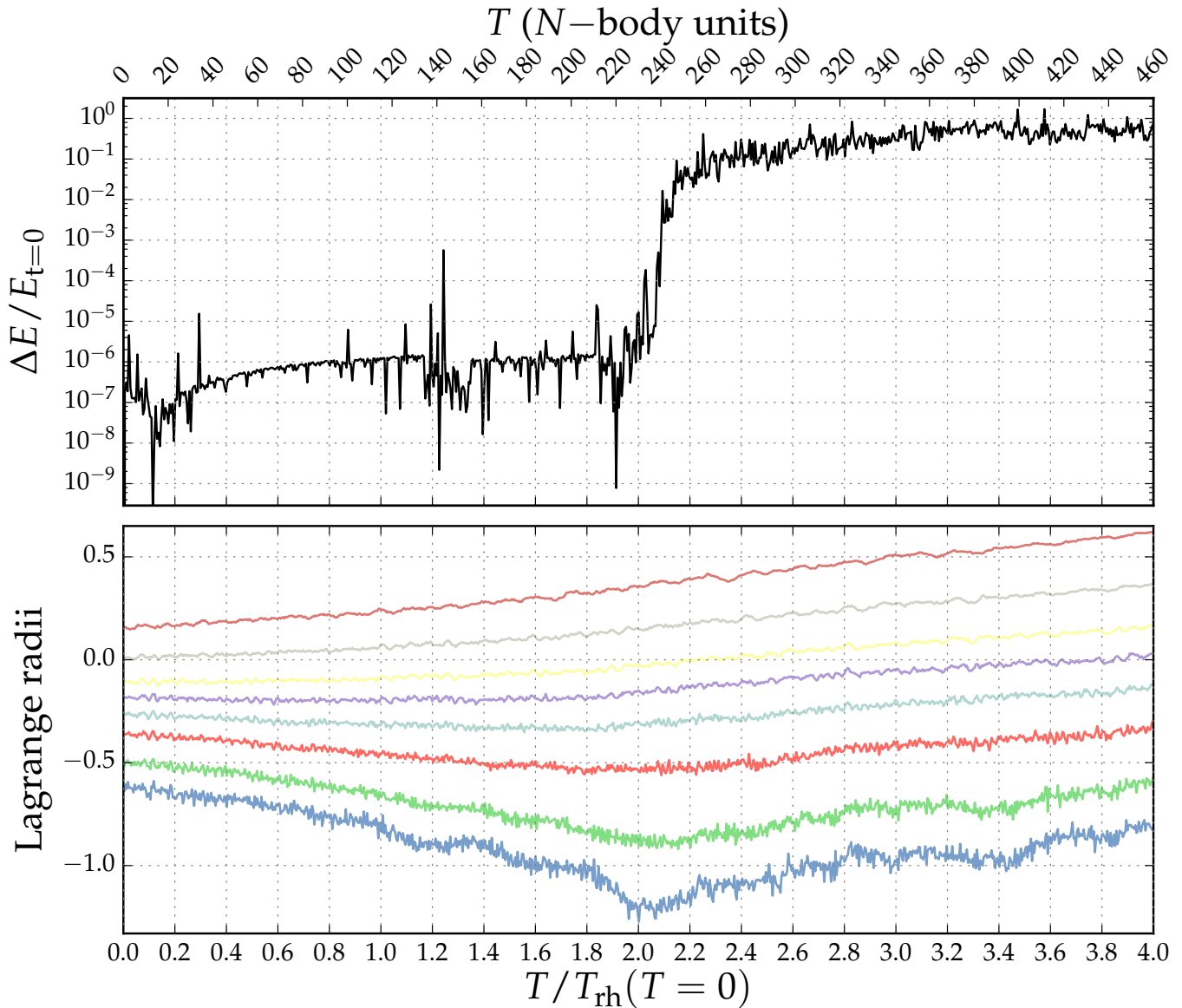
Additionally to the single-mass calculations, we have also addressed multi-mass systems. The fact of having an Initial Mass Function (IMF) accelerates the core collapse of the system, as shown by many different authors (Inagaki & Wiyanto 1984; Spitzer 1987; Kim & Lee 1997; Kim et al. 1998). In our calculations, we use a Plummer sphere with a Kroupa IMF (Kroupa 2001) and 8192 particles. In Figure (11) we present the evolution of the Lagrange radii and the energy conservation of the system. We can see that the



**Figure 10.** Energy conservation in a long time integration of a system with  $N = 8102$  Comparison of the Energy conservation of a Plummer Sphere with  $N = 8192$  particles, using different values of  $\epsilon$  (softening) for GRAVIDY and the NBODY6GPU code, from upper to bottom. The core collapse is reached at  $\approx 24, 18$  and  $16 T_{\text{rh}}$  for  $\epsilon = 10^{-3}, 10^{-4}$  and  $10^{-5}$  respectively. The half-mass relaxation time for this system is  $T_{\text{rh}} = 112.186[NBU]$  The NBODY6GPU code does not include a softening parameter, and treat binary evolution with a KS-regularisation. All the runs were stopped after the core collapse.

core collapse happens around  $2T_{\text{rh}}$ , which is the point from which the energy conservation becomes worse and worse, to achieve a value of about 6 orders of magnitude worse than in phases before the collapse. Another way of depicting the collapse is by identifying the heaviest 10% of the stellar population and how it distributes in the core radius as calculated at  $T = 0$ . We can see this in Figure (12).

The equilibrium of the system can be evaluated by analysing the distribution of the time steps. As we have mentioned previously, in Section (2.4), the initial distribution of time steps in the system has a log-normal distribution, which in a balanced system must remain similar, or close. In Figure (13) we show the step distribution after the core collapse for the single-mass system with  $\epsilon = 10^{-4}$



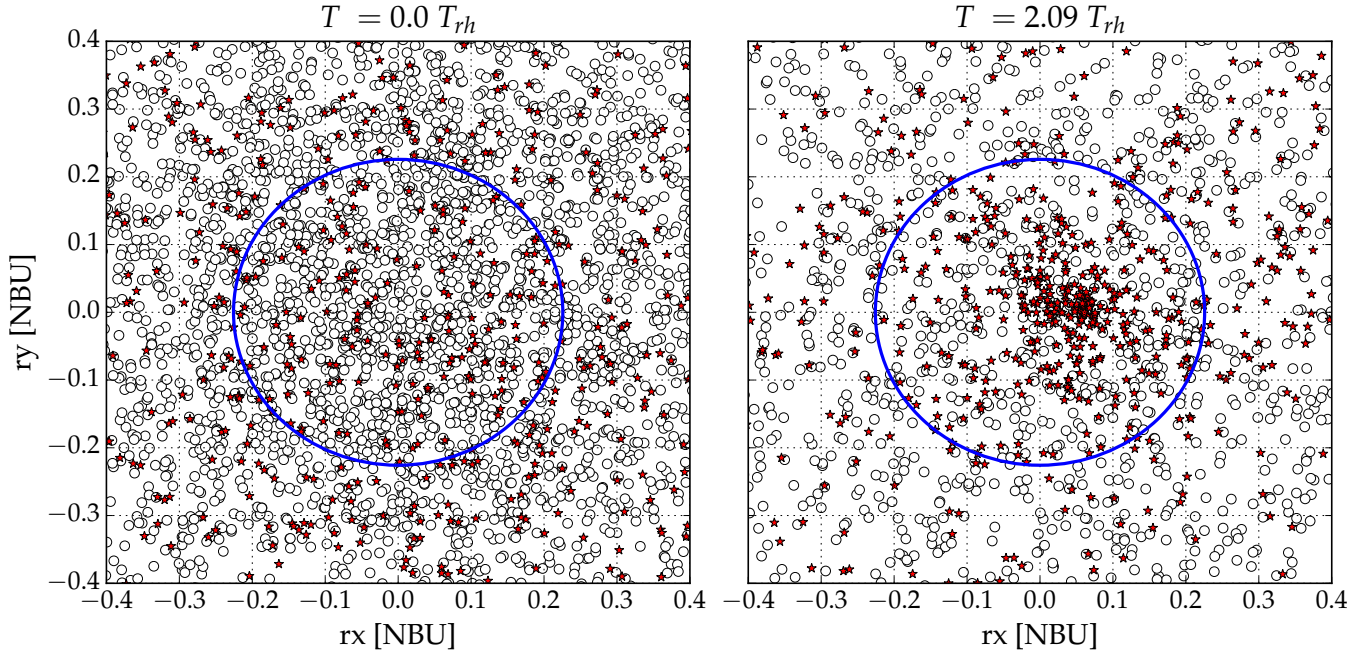
**Figure 11.** Plummer sphere using 8192 particles and following a Kroupa IMF. *Top Panel:* Cumulative energy of the system. *Bottom Panel:* Lagrange radii distribution for 0.5, 1, 2, 3, 4, 5, 6, 75 and 90% of the total mass.

## 5 RELATIVISTIC CORRECTIONS

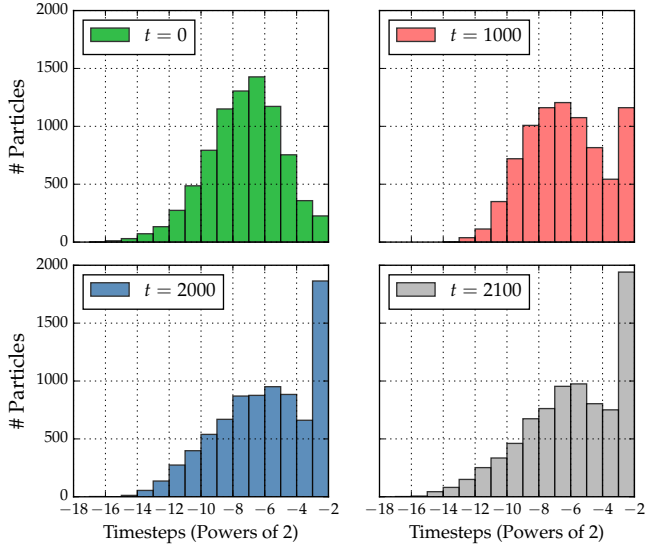
GRAVIDY includes a treatment of relativistic orbits. This has been implemented for the code to be able to study sources of gravitational waves. The approach we have used is the post-Newtonian one, presented for the first time in an  $N$ -body code in the work of Kupi et al. (2006) (and see also Amaro-Seoane & Chen 2016) and later expanded to higher orders in Brem et al. (2013). The idea is to modify the accelerations in the code to include relativistic corrections at 1PN, 2PN (periastris shifts) and 2.5PN (energy loss in the form of gravitational wave emission). Contrary to the scheme of Kupi et al. (2006), which implements the modification in the regularised binaries, in the case of GRAVIDY, the corrections are active for a pair of two particles for which we set the softening to zero. The expressions for the accelerations,

as well as their time derivatives can be found in the updated review of 2017 Amaro-Seoane (2012).

We run a series of different tests for binaries with different mass ratios and initial semi-major axis. In Fig.(14) we display the evolution of a binary of two super massive black holes of total mass  $1.33^6 M_{\odot}$  and mass ratios of 1 and 2. In Fig.(15) we show mass ratios of 5 and 100, and the latter starts with a smaller initial semi-major axis. For each of these cases we plot the geometric distance, the relative velocity and the eccentricity. Higher mass ratios lead to a more complex structure in the evolution. We can see how the relative velocity increases up to a significant fraction of the speed of light  $c$  as the separation grows smaller. We however note that the post-Newtonian approach should not be trusted for velocities larger than about 20%  $c$ .



**Figure 12.** Inner section of a Plummer sphere with 8192 particles Plummer sphere following a Kroupa IMF before (left) and after the core collapse (right) at  $T = 2.09 T_{rh}$ . The blue circle depict the core radius at  $T = 0$ . The top 10% of the heaviest particles in the system are marked as red stars, while all other particles as empty circles.



**Figure 13.** Time step distribution of a Plummer sphere with  $N = 8192$  particles. Four different times are shown, (1)  $t = 0$  NBU, for an initial distribution (upper left panel) (2)  $t = 1000$  NBU, a few half-mass relaxation times  $\sim 9 T_{rh}$  (upper right panel), (3)  $t = 2000$  NBU, a pre core-collapse stage with many particles leaving the core (lower left panel), (4)  $t = 2100$  NBU, a post core-collapse stage with a few particles (mostly binaries) reaching smaller time steps (lower right panel).

## 6 CONCLUSIONS AND FUTURE WORK

In this work we have presented the first version of our new  $N$ -body code, written purely in C/C++, using OpenMPI and CUDA, which we call GRAVIDY. The current version of

our code provides an environment to evolve a self-gravitating stellar system, and uses a H4 integration scheme, using block time steps and softening, and features relativistic corrections (perapsis shift and energy loss) for sources of gravitational radiation. This first release of GRAVIDY has been mainly focused on users who can have access to a machine hosting few GPUs, or usual parallel CPU systems.

We summarise here the main features of GRAVIDY:

(i) The code is written using an iterative and incremental development, which is methodology similar to the Assess, Parallelise, Optimise, Deploy (APOD) development cycle presented by NVIDIA.

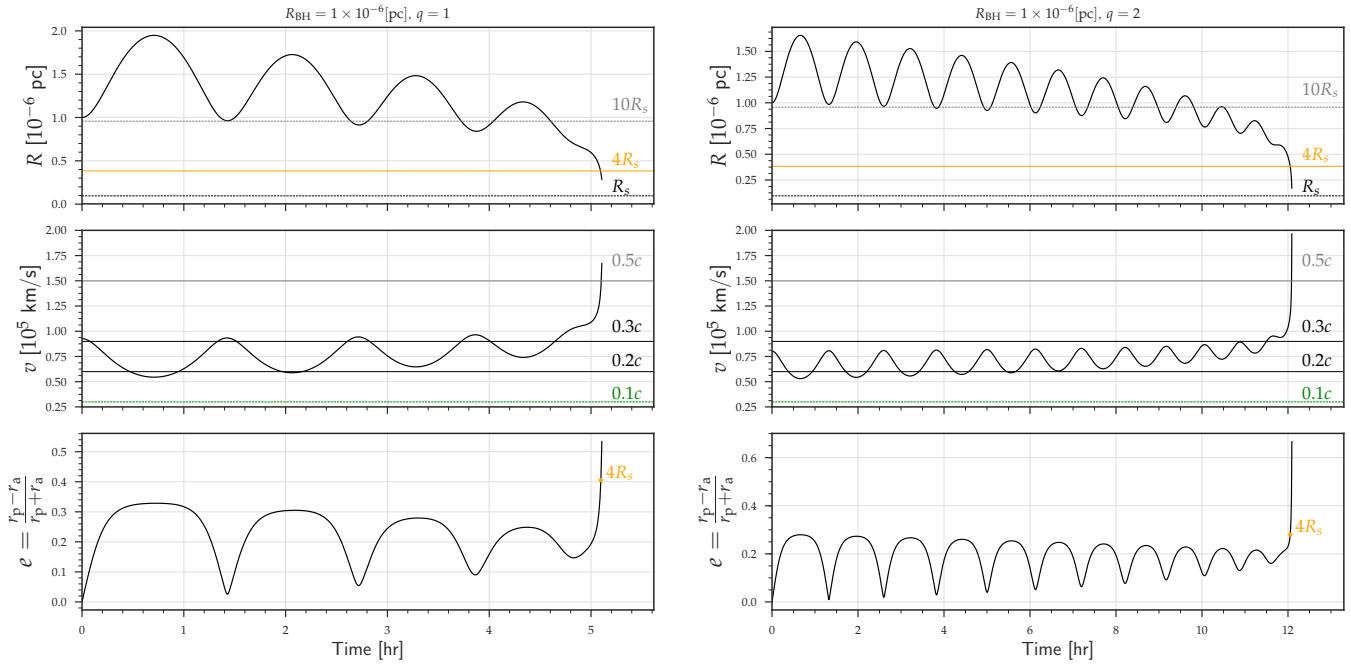
(ii) The code organisation is designed to be highly modular. Every critical process of the integrator is represented by a separate function or chain of functions. Our goal is to produce a code which can be read without difficulties, which makes easier future modifications or forks.

(iii) Since maintainability is one of our main goals, the documentation is also a critical factor. We document every function in the inner procedure of the integrator.

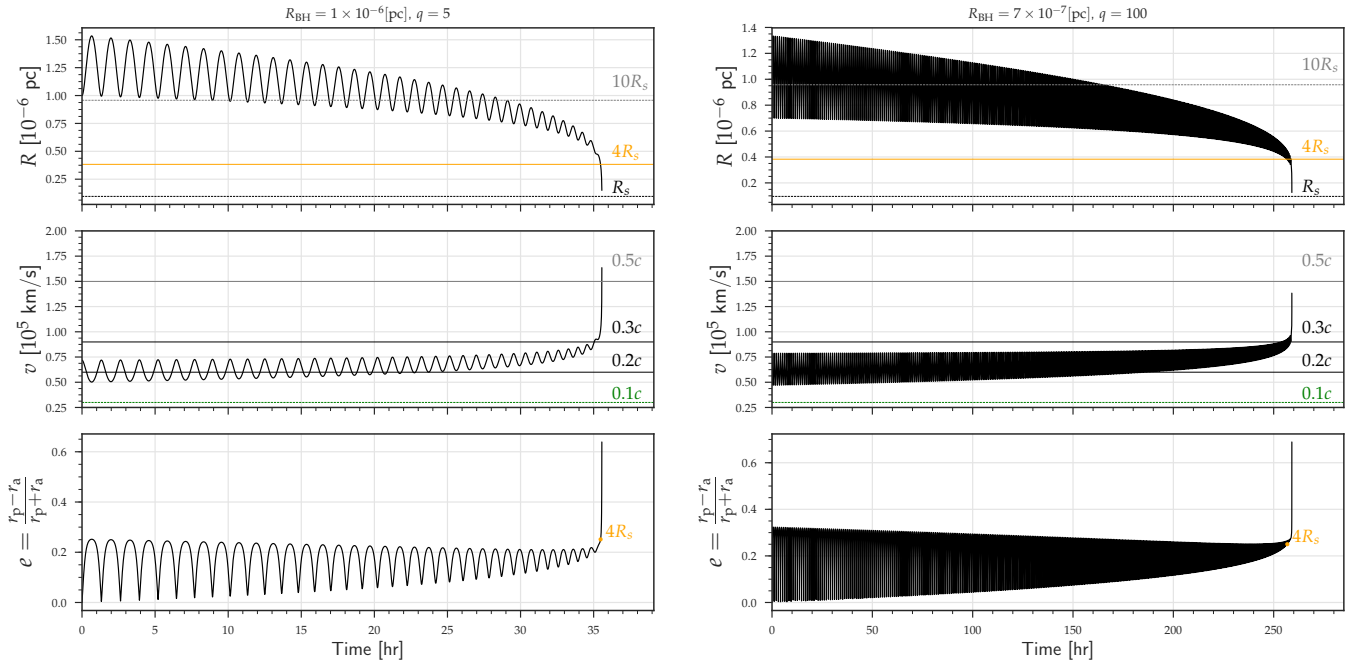
(iv) We use a H4 integrator scheme.

(v) The code uses block time steps to improve the performance of the integrator. We evolve particles in groups of block time steps, which allows for an update of several particles at the same time.

(vi) We use GPU computing techniques, OpenMP and OpenMPI to parallelise the calculation of the gravitational interactions of our system after having localised the hot-spots of our algorithm. The main objective here was to be able to update a relatively small amount of particles which share a common time step in a given moment, a situation which is against the design of GPU cards, developed to reach a high parallelism.



**Figure 14.** *Upper panel:* Evolution of the initial distance  $R_{\text{BH}}$  between the two super massive black holes for a mass ratio  $q = 1$  (and  $q = 2$  in the set of three right panels). The three horizontal lines correspond, from the top to the bottom, to a distance of  $10 R_{\text{S}}$ , with  $R_{\text{S}}$  the Schwarzschild radius,  $4 R_{\text{S}}$ , and  $1 R_{\text{S}}$ . *Mid panel:* Evolution of the relative velocity between the two super massive black holes. The four horizontal lines correspond, from the top to the bottom, to a fraction of the speed of light  $c$  of 50%, 30%, 20% and 10%. *Bottom panel:* Evolution of the eccentricity of the binary as a function of time in hours. We mark the point in the evolution at which the separation is  $4 R_{\text{S}}$  with an orange dot.



**Figure 15.** Same as Fig.(14) but for mass ratios  $q = 5$  (left panels) and  $q = 100$  (right panels), and a different initial separation, of  $7 \times 10^{-7}$  pc (also right panels).

In this first release of GRAVIDY and first paper, we have presented a series of classical tests of the code, as well as a study of the performance of its different “flavours”: the single CPU version, the MPI one and the GPU version. We also address the role of the softening in the global evolution of a system, as integrated with our code. As expected, the value of the softening is crucial in determining the global dynamics, and should not be taken lightly, in particular if one is interested in studying physical phenomena for which relaxation is important, since using a softening translates into a maximum increase of the forces and the a smoothly declination to zero, which is approximate. To study a dynamical process, such as e.g. the collision of two clusters, focusing on the short-term (i.e. for times well below a relaxation time) dynamical behaviour of the system, using a softening should be fine, but the role of the parameter should be assessed carefully by exploring different values.

The on-going development of GRAVIDY includes a close encounter solver, with a time-symmetric integration scheme to treat binaries, such as the one presented in the work of Konstantinidis & Kokkotas (2010). Another immediate goal of the next releases, is to include a central massive particle and the required corrections to the gravitational forces so as to ensure a good conservation of the energy in the system. This massive particle could be envisaged as a massive black hole in a galactic centre or a star in a protoplanetary system. We also plan on bridging the gap between spherical nucleus models that focus on collisional effects and simulations of larger structure that are able to account for complex, more realistic non-spherical geometry. Finally, a future goal is to include stellar evolution routines, from which the modularity of our code will provide an easy scenario. One of the candidate modules for this could be SEVN (Spera et al. 2015).

We will follow the APOD cycle presented in this work, it is necessary to study new computational techniques, so as to improve the performance of our code: from variable precision to new parallel schemes to perform the force interaction calculation, using one or more GPU.

## APPENDIX A: ABOUT THE CODE

GRAVIDY is a C/C++ and CUDA application, that uses the CUDA, OpenMPI and boost libraries.

As an overview, the compilation can be done with: `make <flavour>`, for the `cpu`, `mpi` and `gpu` versions. A simple run of the code is displayed in the Listing 1.

The URL hosting the project is <http://gravidy.xyz>, where you can find the prerequisites, how to get, compile and use the code more detailed. Additionally, documentation regarding the code, input and output files is included.

Inside the repository, there is a `scripts` directory with a set of classes to be able to handle all the output files of the code.

The code was compiled using `gcc (4.9.2)`, `openmpi(1.6.5)`, `CUDA(6.0)` and `boost (1.55)`.

The following compilation FLAGS were used `-O3 -Wall -fopenmp -pipe -fstack-protector -Wl,-z,relro -Wl,-z,now -Wformat-security -Wpointer-arith -Wformat-nonliteral -Wl,-O1 -Wl,--discard-all -Wl,--no-undefined -rdynamic`.

Listing 1: Example run of the integrator. Columns, decimals, and information were modified to fit the output on this document.

```
$ ./gravidy-gpu -i ../input/04-nbody-p1024_m1.in -p -t 1
[2017-01-28 01:60:56] [INFO] GPUs: 1
[2017-01-28 01:60:56] [INFO] Spl. 1024 particles in 1 GPUs
[2017-01-28 01:60:56] [INFO] GPU 0 particles: 1024
Time  Iter  Nsteps  Energy      RelE      CumE      ETime
0.000  0      0      -2.56e-01  0.00e+00  0.00e+00  3.08e-02
0.125  698    30093  -2.56e-01  2.41e-07  2.41e-07  3.84e-01
0.250  1262   61319  -2.56e-01  1.10e-07  1.30e-07  6.60e-01
0.375  1897   91571  -2.56e-01  4.19e-08  8.84e-08  9.49e-01
0.500  2530   121963 -2.56e-01  8.51e-08  3.30e-09  1.23e+00
0.625  3132   150924 -2.56e-01  2.89e-08  3.23e-08  1.52e+00
0.750  3725   180446 -2.56e-01  1.39e-08  1.83e-08  1.76e+00
0.875  4354   212425 -2.56e-01  5.23e-07  5.41e-07  2.02e+00
1.000  5160   244165 -2.56e-01  2.32e-07  3.09e-07  2.32e+00
[2017-01-28 01:60:59] [SUCCESS] Finishing...
```

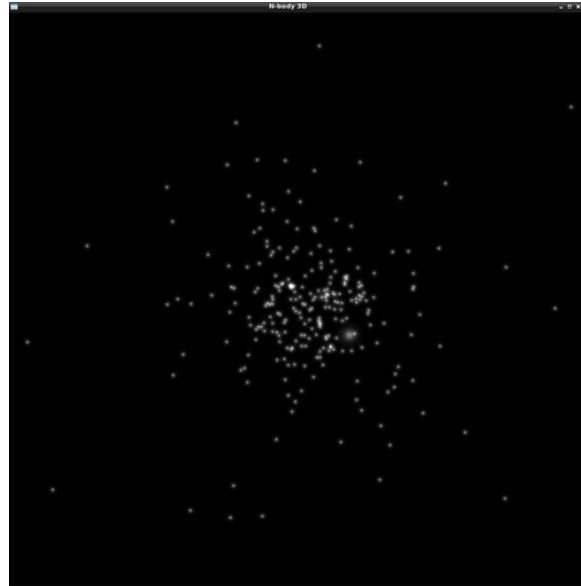


Figure B1. Snapshot pre-visualisation with GraviDyView using a  $N = 1024$  system.

## APPENDIX B: $N$ -body VISUALISATION TOOL

A graphical representation of  $N$ -body simulations is always an attractive idea to display how the simulation was performed. Due to this reason, we decided to write a small application to have a simple 3D visualisation of GRAVIDY snapshots, based in OpenGL.

GRAVIDYVIEW is a lightweight and simple OpenGL  $N$ -body visualisation tool, written in C/C++. It can be downloaded from:

- <https://gitlab.com/cmaureir/gravidy-view>.

## ACKNOWLEDGEMENTS

We are thankful for hints, help and discussion to Sverre Aarseth, Holger Baumgardt, Peter Berczik, Douglas Heggie, Piet Hut, Simos Konstantinidis, Patrick Brem, Keigo Nitadori, Ulf Löckmann and Rainer Spurzem. In general, we are indebted with the very friendly  $N$ -body commu-

nity, for keeping their codes publicly available to everybody. PAS acknowledges support from the Ramón y Cajal Programme of the Ministry of Economy, Industry and Competitiveness of Spain. CMF thanks the support and guidance of Prof. Luis Salinas during the development of his Master thesis, which was the motivation of this project. This work has been supported by the “Dirección General de Investigación y Postgrado” (DGIP) by means of the “Programa Incentivo a la Iniciación Científica” (PIIC) at the Universidad Técnica Federico Santa María, the “Comisión Nacional de Investigación Científica y Tecnológica de Chile” (CONICYT) through its Master scholarships program and the Transregio 7 “Gravitational Wave Astronomy” financed by the Deutsche Forschungsgemeinschaft DFG (German Research Foundation). CMF acknowledges support from the DFG Project “Supermassive black holes, accretion discs, stellar dynamics and tidal disruptions”, awarded to PAS, and the International Max-Planck Research School.

## REFERENCES

- Aarseth S. J., 1985, in Brackbill J. U., Cohen B. I., eds, Multiple time scales, p. 377 - 418. pp 377–418
- Aarseth S. J., 1999, The Publications of the Astronomical Society of the Pacific, 111, 1333
- Aarseth S. J., 2003, Gravitational N-Body Simulations. ISBN 0521432723. Cambridge, UK: Cambridge University Press, November 2003.
- Aarseth S. J., Zare K., 1974, *Celestial Mechanics*, 10, 185
- Aarseth S. J., Hénon M., Wielen R., 1974, *A&A*, 37, 183
- Amaro-Seoane P., 2012, preprint, ([arXiv:1205.5240](https://arxiv.org/abs/1205.5240))
- Amaro-Seoane P., Chen X., 2016, *MNRAS*, 458, 3075
- Amaro-Seoane P., Freitag M., Spurzem R., 2004, *MNRAS*, Barnes J., Hut P., 1986, *Nat*, 324, 446
- Belleman R. G., Bédorf J., Portegies Zwart S. F., 2008, *New Astronomy*, 13, 103
- Berczik P., et al., 2011, pp 8–18
- Berczik P., Spurzem R., Wang L., Zhong S., Huang S., 2013, in Third International Conference “High Performance Computing”, HPC-UA 2013, p. 52-59. pp 52–59 ([arXiv:1312.1789](https://arxiv.org/abs/1312.1789))
- Brem P., Amaro-Seoane P., Spurzem R., 2013, *MNRAS*, 434, 2999
- Capuzzo-Dolcetta R., Spera M., 2013, *Computer Physics Communications*, 184, 2528
- Capuzzo-Dolcetta R., Spera M., Punzo D., 2013, *Journal of Computational Physics*, 236, 580
- Fukushige T., Makino J., Kawai A., 2005, *PASJ*, 57, 1009
- Gaburov E., Harfst S., Zwart S. P., 2009, *New Astronomy*, 14, 630
- Giersz M., Spurzem R., 1994, *MNRAS*, 269, 241
- Greendard L., 1987, PhD thesis, Yale University, New Haven, CT
- Hamada T., Iitaka T., 2007, *New Astronomy*,
- Harfst S., Gualandris A., Merritt D., Mikkola S., 2008, *MNRAS*, 389, 2
- Heggie D., Hut P., 2003, *The Gravitational Million-Body Problem: A Multidisciplinary Approach to Star Cluster Dynamics*, by Douglas Heggie and Piet Hut. Cambridge University Press, 2003, 372 pp.
- Heggie D. C., Mathieu R. D., 1986, in Hut P., McMillan S. L. W., eds, *Lecture Notes in Physics*, Berlin Springer Verlag Vol. 267, The Use of Supercomputers in Stellar Dynamics. p. 233, doi:10.1007/BFb0116419
- Hénon M. H., 1971, *A&AS*, 14, 151
- Holmberg E., 1941, *ApJ*, 94, 385
- Hut P., 2003, in Makino J., Hut P., eds, *IAU Symposium Vol. 208, Astrophysical Supercomputing using Particle Simulations*. p. 331 ([arXiv:astro-ph/0204431](https://arxiv.org/abs/astro-ph/0204431))
- Inagaki S., Wiyanto P., 1984, *PASJ*, 36, 391
- Kim S. S., Lee H. M., 1997, *Journal of Korean Astronomical Society*, 30, 115
- Kim S. S., Lee H. M., Goodman J., 1998, *ApJ*, 495, 786
- Konstantinidis S., Kokkotas K. D., 2010, *A&A*, 522, A70
- Kroupa P., 2001, *MNRAS*, 322, 231
- Kupi G., Amaro-Seoane P., Spurzem R., 2006, *MNRAS*, pp L77+
- Küpper A. H. W., Maschberger T., Kroupa P., Baumgardt H., 2011, *MNRAS*, 417, 2300
- Kustaanheimo P. E., Stiefel E. L., 1965, *J. Reine Angew. Math.*, 218, 204
- Makino J., 1991, *ApJ*, 369, 200
- Makino J., 1998, *Highlights in Astronomy*, 11, 597
- Makino J., Aarseth S. J., 1992, *PASJ*, 44, 141
- Makino J., Taiji M., 1998, *Scientific simulations with special-purpose computers : The GRAPE systems*. Scientific simulations with special-purpose computers : The GRAPE systems /by Junichiro Makino & Makoto Taiji. Chichester ; Toronto : John Wiley & Sons, c1998.
- Nguyen H., 2007, *Gpu gems 3*, first edn. Addison-Wesley Professional
- Nitadori K., 2009, PhD thesis, University of Tokyo
- Nitadori K., Aarseth S. J., 2012, *MNRAS*, 424, 545
- Nitadori K., Makino J., 2008, *na*, 13, 498
- Plummer H. C., 1911, *MNRAS*, 71, 460
- Portegies Zwart S. F., McMillan S. L. W., Hut P., Makino J., 2001a, *MNRAS*, 321, 199
- Portegies Zwart S. F., McMillan S. L. W., Hut P., Makino J., 2001b, *MNRAS*, 321, 199
- Portegies Zwart S. F., Belleman R. G., Geldof P. M., 2007, *New Astronomy*, 12, 641
- Press W. H., 1986, in Hut P., McMillan S. L. W., eds, *The Use of Supercomputers in Stellar Dynamics*. Springer-Verlag, p. 184
- Schneider J., Amaro-Seoane P., Spurzem R., 2011, *MNRAS*, 410, 432
- Spera M., Mapelli M., Bressan A., 2015, *MNRAS*, 451, 4086
- Spitzer L., 1987, *Dynamical evolution of globular clusters*. Princeton, NJ, Princeton University Press, 1987, 191 p.
- Spitzer L. J., Hart M. H., 1971, *ApJ*, 166, 483
- Spurzem R., 1999, *Journal of Computational and Applied Mathematics*, 109, 407
- Taiji M., Makino J., Fukushige T., Ebisuzaki T., Sugimoto D., 1996, in Hut P., Makino J., eds, *IAU Symp. 174: Dynamical Evolution of Star Clusters: Confrontation of Theory and Observations*. p. 141
- Wang L., Spurzem R., Aarseth S., Nitadori K., Berczik P., Kouwenhoven M. B. N., Naab T., 2015, *mn*, 450, 4070
- Wang L., et al., 2016, *mn*, 458, 1450
- von Hoerner S., 1960, *Z. Astrophys.*, 50, 184
- von Hoerner S., 1963, *Z. Astrophys.*, 57, 47