Chaotic Dynamics in N-body systems

Boekholt, T.C.N.

3

A Parallel Efficient N-body Code: Sakura

Based on: *A Keplerian-based Hamiltonian Splitting for Gravitational N-body Simulations* by G. Gonçalves Ferrari, T. C. N. Boekholt and S. F. Portegies Zwart in Monthly Notices of the Royal Astronomical Society, Volume 440, Issue 1, p.719-730 (2014)

We develop a Keplerian-based Hamiltonian splitting for solving the gravitational N-body problem. This splitting allows us to approximate the solution of a general N-body problem by a composition of multiple, independently evolved 2-body problems. While the Hamiltonian splitting is exact, we show that the composition of independent 2-body problems results in a non-symplectic non-time-symmetric first-order map. A time-symmetric second-order map is then constructed by composing this basic first-order map with its self-adjoint. The resulting method is precise for each individual 2-body solution and produces quick and accurate results for near-Keplerian N-body systems, like planetary systems or a cluster of stars that orbit a supermassive black hole. The method is also suitable for integration of N-body systems with intrinsic hierarchies, like a star cluster with primordial binaries. The superposition of Kepler solutions for each pair of particles makes the method excellently suited for parallel computing; we achieve $\gtrsim 64\%$ efficiency for only eight particles per core, but close to perfect scaling for 16384 particles on a 128 core distributed-memory computer. We present several implementations in Sakura, one of which is publicly available via the AMUSE framework.

3.1 INTRODUCTION

Since the pioneering work of von Hoerner (1960), Aarseth (1963) and **?** N-body simulations have been an essential tool for the theoretical understanding of self-gravitating astrophysical systems. Such systems often show a large dynamic range of time scales. Thus, instead of a

fixed or adaptive global time step, most of the N-body codes adopt individual or block time step algorithms in order to advance the particles in time (**??**Aarseth, 2003). In addition, different approaches to calculate the acceleration of each particle, such as using grids (**?**) or a hierarchical tree data structure (**?**), are commonly employed to decrease the computational cost of the simulations. These approaches allow the use of a larger number of particles, despite only giving an approximation to the true acceleration of each particle. Therefore, these codes should not inadvertently be used in simulations of collisional systems such as planetary systems, dense star clusters or the inner parts of galactic nuclei.

In collisional systems the individual interactions between particles play an important role in the dynamical evolution of the system as a whole. For example, the formation of hard binaries in star cluster core collapse (**?**) requires very precise integration methods to correctly evolve close encounters between particles. This precision is only possible if we use more accurate, direct brute-force methods, to calculate the accelerations due to each pair of particles in the system. The main difficulty here is that with the formation of the first hard binary in the system, the simulation as a whole experiences a slow-down in performance due to the necessity to decrease the time-step size in order to accurately integrate such compact sub-systems.

Currently, the most effective and common approach to overcome such obstacles seems to be a combination of the block time step algorithm (**?**), Ahmad-Cohen neighbour scheme (**?**) and some sort of 2-body regularization (**????**) in order to handle very compact subsystems efficiently. This is the approach used in modern Hermite integrators for collisional stellar systems (**?**Aarseth, 2003).

In this chapter, we develop a new Keplerian-based Hamiltonian splitting for the gravitational N-body problem. This splitting allows us to approximate the solution of a general N-body problem by a composition of independently evolved two-body problems. While the Hamiltonian splitting is exact, we show in section 3.2 that the composition of independent two-body problems results in a non-symplectic non-time-symmetric first-order map. A time-symmetric second-order map is then constructed by composing this basic first-order map with its self-adjoint. The advantages of this Keplerian-based integrator are: i) a guarantee that every pair of particles is always integrated precisely; ii) the method does not suffer from slow-down in performance when tight binaries are present in the simulation, and iii) the method allows for good parallel efficiency (**?**).

3.2 METHOD

3.2.1 Hamiltonian Splitting

We begin the derivation of our scheme for the numerical integration of a gravitational N-body system by considering its Hamiltonian,

$$H = H_T + H_U \,. \tag{3.1}$$

Here,

$$H_T \equiv \sum_{i=1}^{N} H_{T_i} \,, \qquad H_{T_i} \equiv \frac{1}{2} m_i v_i^2 \,, \tag{3.2}$$

and

$$H_U \equiv \frac{1}{2} \sum_{i=1}^{N} \sum_{j \neq i}^{N} H_{U_{ij}} \,, \qquad H_{U_{ij}} \equiv -\frac{m_i m_j}{r_{ij}} \,, \tag{3.3}$$

are the kinetic and potential energies of the system, respectively; m_i and $v_i = |\mathbf{v}_i|$ are the mass and velocity of the i-th particle and $r_{ij} = |\mathbf{r}_{ij}| = |\mathbf{r}_i - \mathbf{r}_j|$ is the relative distance between particles i and j.

The time evolution of a Hamiltonian system is formally given by the operator[1] $e^{\tau \widehat{H}}$, which can be approximated by a composition of individually solvable operators $e^{\tau \widehat{H}_A}$ and $e^{\tau \widehat{H}_B}$ in cases when the Hamiltonian can be split as $\widehat{H} = \widehat{H}_A + \widehat{H}_B$. The simplest example of Hamiltonian splitting is the case when $\widehat{H}_A = \widehat{H}_T$ and $\widehat{H}_B = \widehat{H}_U$, for which we can generate the time-symmetric second-order Drift-Kick-Drift (DKD) variant of the Leapfrog integrator: $e^{\tau \widehat{H}} \approx e^{\frac{\tau}{2} \widehat{H}_T} e^{\tau \widehat{H}_U} e^{\frac{\tau}{2} \widehat{H}_T}$. This Hamiltonian splitting is not the only possibility and many other ways of subdividing the system have been tried (**?????**).

In the present chapter we introduce a way to split the Hamiltonian of an N-body system, which is based on two main arguments: i) the validity of the superposition principle[2], and ii) the existence of an analytical solution for the 2-body problem. Therefore, a natural way to approximate the time evolution of an N-body system is by using a composition of 2-body problems to solve a more general N-body problem. While this approach may seem computationally expensive,

[1] Hamiltonian associated operators are denoted by a $\widehat{\ }$ symbol.

[2] Recall that the gravitational potential and acceleration at the position of a given particle consists of a superposition of 2-body contributions due to the interaction with every other particle in the system.

our aim here is to present a theoretical formulation of the method. Possible optimizations, such as applying the Kepler-solver only to a few close pairs in the simulation, or to make use of Newton's third law during the force loop, are left for future implementations.

We first rewrite the potential energy term in Eq. 3.3 as follows:

$$\begin{aligned} H_U &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} H_{U_{ij}} \\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} -\frac{m_i m_j}{r_{ij}} \\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} -\mu_{ij}\frac{(m_i+m_j)}{r_{ij}} \\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} \mu_{ij}\left\{\left[\frac{1}{2}v_{ij}^2 - \frac{(m_i+m_j)}{r_{ij}}\right] - \frac{1}{2}v_{ij}^2\right\} \\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} \left(H_{K_{ij}} - H_{T_{ij}}\right) \\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} H_{W_{ij}} \equiv H_W\,. \end{aligned} \tag{3.4}$$

Here,

$$H_{K_{ij}} \equiv \mu_{ij}\left[\frac{1}{2}v_{ij}^2 - \frac{(m_i+m_j)}{r_{ij}}\right] \tag{3.5}$$

is the 2-body Keplerian Hamiltonian and

$$H_{T_{ij}} \equiv \frac{1}{2}\mu_{ij}v_{ij}^2, \tag{3.6}$$

where $\mu_{ij} = m_i m_j/(m_i+m_j)$ is the reduced mass of the $i-j$ pair. The original N-body Hamiltonian in Eq. 3.1 can now be rewritten as follows:

$$H = H_T + H_W = \sum_{i=1}^{N} H_{T_i} + \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} H_{W_{ij}}\,, \tag{3.7}$$

with

$$H_{W_{ij}} = H_{K_{ij}} - H_{T_{ij}} \equiv H_{U_{ij}}\,. \tag{3.8}$$

We note that Eq. 3.7, as is clear from the equivalence in Eq. 3.8, can always be reduced by simplification into Eq. 3.1, which implies that, in principle, our Keplerian-based Hamiltonian splitting does not change the dynamics of the system.

3.2.2 Equations of Motion

According to the general theory of geometric integrators (**?**) we can construct a time-symmetric second-order method by composing a (possible non-time-symmetric) first order method, $\phi(\tau)$, with its self-adjoint, $\phi^\dagger(\tau)$. Moreover, the composition $\Psi(\tau) = \phi(\frac{\tau}{2}) \circ \phi^\dagger(\frac{\tau}{2})$ is symplectic if both $\phi(\tau)$ and $\phi^\dagger(\tau)$ are symplectic methods.

In our Keplerian-based Hamiltonian splitting, time evolution operators can be constructed by taking into account that

$$e^{\tau \widehat{H}_T} = \prod_{i=1}^{N} e^{\tau \widehat{H}_{T_i}} , \tag{3.9}$$

$$e^{\tau \widehat{H}_U} = \prod_{i=1}^{N} e^{\tau \frac{1}{2} \widehat{H}_{U_i}} = \prod_{i=1}^{N} e^{\tau \frac{1}{2} \sum_{j \neq i}^{N} \widehat{H}_{U_{ij}}} . \tag{3.10}$$

and, by Eq. 3.4,

$$e^{\tau \widehat{H}_W} = \prod_{i=1}^{N} e^{\tau \frac{1}{2} \widehat{H}_{W_i}} = \prod_{i=1}^{N} e^{\tau \frac{1}{2} \sum_{j \neq i}^{N} \widehat{H}_{W_{ij}}} , \tag{3.11}$$

where the last term on the right hand side in eqs. 3.10 and 3.11 is a simple substitution of the definition of operators $\widehat{H}_{U_i}$ and $\widehat{H}_{W_i}$, i.e., $\widehat{H}_{U_i} = \sum_{j \neq i}^{N} \widehat{H}_{U_{ij}}$ and similar for $\widehat{H}_{W_i}$, and the presence of the factor $1/2$ follows from the fact that we have to take into account each $i - j$ pair only once. In eqs. 3.9, 3.10 and 3.11 the individual operators $e^{\tau \widehat{H}_{T_i}}$, $e^{\tau \widehat{H}_{U_i}}$ and $e^{\tau \widehat{H}_{W_i}}$ act on the $6N$ dimensional array $(\mathbf{r}_i, \mathbf{v}_i)$. Here the "one-subscript" operators individually commutate since they can only act on the corresponding coordinates with subscript i. Therefore, the order in which the product of operators is executed in each of eqs. 3.9, 3.10 and 3.11 is unimportant. In order to proceed with the derivation we present these operators in a more explicit form as follows:

$$e^{\tau \widehat{H}_{T_i}} : \quad \begin{pmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{pmatrix} + \tau \begin{pmatrix} \mathbf{v}_i \\ \mathbf{0} \end{pmatrix} , \tag{3.12}$$

$$e^{\tau \widehat{H}_{U_i}} : \quad \begin{pmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{pmatrix} + \tau \begin{pmatrix} \mathbf{0} \\ \mathbf{a}_i \end{pmatrix} , \tag{3.13}$$

$$e^{\tau \widehat{H}_{W_i}} : \quad \begin{pmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{pmatrix} + \begin{pmatrix} \delta\mathbf{r}_i \\ \delta\mathbf{v}_i \end{pmatrix} , \tag{3.14}$$

where $\mathbf{a}_i$ is the acceleration and $(\delta\mathbf{r}_i, \delta\mathbf{v}_i)$ are the increments in absolute coordinates and will be specified later on in Eq. 3.23.

In a similar way, individual "two-subscript" operators are explicitly written as follows:

$$e^{\tau \widehat{H}_{U_{ij}}} : \quad \begin{pmatrix} \mathbf{r}_{ij} \\ \mathbf{v}_{ij} \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{r}_{ij} \\ \mathbf{v}_{ij} \end{pmatrix} + \tau \begin{pmatrix} \mathbf{0} \\ \mathbf{a}_{ij} \end{pmatrix} , \tag{3.15}$$

$$e^{\tau \widehat{H}_{W_{ij}}} : \quad \begin{pmatrix} \mathbf{r}_{ij} \\ \mathbf{v}_{ij} \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{r}_{ij} \\ \mathbf{v}_{ij} \end{pmatrix} + \begin{pmatrix} \delta\mathbf{r}_{ij} \\ \delta\mathbf{v}_{ij} \end{pmatrix} , \tag{3.16}$$

$$e^{\tau\left(-\widehat{H}_{T_{ij}}\right)} : \quad \begin{pmatrix} \mathbf{r}_{ij} \\ \mathbf{v}_{ij} \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{r}_{ij} \\ \mathbf{v}_{ij} \end{pmatrix} - \tau \begin{pmatrix} \mathbf{v}_{ij} \\ \mathbf{0} \end{pmatrix} , \tag{3.17}$$

$$e^{\tau \widehat{H}_{K_{ij}}} : \quad \mathbf{r}_{ij}, \mathbf{v}_{ij} \leftarrow \texttt{kepler\_solver}(\tau, m_{ij}, \mathbf{r}_{ij}, \mathbf{v}_{ij}) , \tag{3.18}$$

where $m_{ij} = m_i + m_j$, $\mathbf{a}_{ij} = -m_{ij}\mathbf{r}_{ij}/r_{ij}^3$ is the relative 2-body acceleration. The increments in relative coordinates, $(\delta\mathbf{r}_{ij}, \delta\mathbf{v}_{ij})$, are obtained independently for each $i-j$ pair from the application of one of the first-order maps:

$$e^{\tau\left(\widehat{H}_{K_{ij}} - \widehat{H}_{T_{ij}}\right)} \approx e^{\tau\left(-\widehat{H}_{T_{ij}}\right)} e^{\tau \widehat{H}_{K_{ij}}} , \tag{3.19a}$$

$$e^{\tau\left(\widehat{H}_{K_{ij}} - \widehat{H}_{T_{ij}}\right)} \approx e^{\tau \widehat{H}_{K_{ij}}} e^{\tau\left(-\widehat{H}_{T_{ij}}\right)} . \tag{3.19b}$$

Eqs. 3.12 to 3.17 are first-order approximations to the respective operators in these equations. It will be clear below that this low-order approximation is enough for our purposes since, ultimately, the order of the full time evolution operator in Eq. 3.25 will be determined by the composition of those operators. In this sense, if a high-order approximation of the method presented here is needed, we argue that this should be obtained not by extending eqs. 3.12 to 3.17 to higher order, but rather, by making a high-order composition of these operators in a similar way as in symplectic integrators (**??**), where a second-order

map is constructed as a composition of first-order operators, and so on.

We notice here that, contrary to the "one-subscript" operators, the "two-subscript" operators act on the $6N(N-1)/2$ dimensional array $(\mathbf{r}_{ij}, \mathbf{v}_{ij})$. Therefore, it remains to be shown how to relate "one-subscript" and "two-subscript" operators in a consistent way. From Eq. 3.10 and the definition of $H_{U_{ij}}$, it is easy to see that the equivalence,

$$\prod_{j\neq i}^{N} e^{\tau \widehat{H}_{U_{ij}}} \equiv e^{\tau \sum_{j\neq i}^{N} \widehat{H}_{U_{ij}}} = e^{\tau \widehat{H}_{U_i}} , \tag{3.20}$$

is valid for every N because the operators $e^{\tau \widehat{H}_{U_{ij}}}$ commutate. On the other hand, from Eq. 3.11, an equivalence similar to Eq. 3.20 relating $\widehat{H}_W$-type operators is only possible for $N = 2$. For $N > 2$ the operators $e^{\tau \widehat{H}_{W_{ij}}}$ do not commutate. However, we can write a similar equation approximately as

$$\prod_{j\neq i}^{N} e^{\tau \widehat{H}_{W_{ij}} + \mathcal{O}(\tau^2)} \approx e^{\tau \sum_{j\neq i}^{N} \widehat{H}_{W_{ij}}} = e^{\tau \widehat{H}_{W_i}} , \tag{3.21}$$

where the error $\mathcal{O}(\tau^2)$ is not guaranteed to be Hamiltonian due to the fact that we treat each $i-j$ pair independently. As a consequence *the symplecticity of the present method is lost.*

Apart from the loss of symplecticity, as mentioned above, a time-symmetric second-order method for our Keplerian-based Hamiltonian splitting can still be constructed by using a composition of self-adjoint first-order methods (see **?**).

In order to construct $\phi(\tau)$ and $\phi^\dagger(\tau)$ we first need to specify the increments $\delta\mathbf{r}_i$ and $\delta\mathbf{v}_i$ in Eq. 3.14. Since in the present method we take advantage of a Kepler-solver to evolve each pair of particles independently, the relative increments $(\delta\mathbf{r}_{ij}, \delta\mathbf{v}_{ij})$ can be easily calculated for each interaction after application of one of the maps in eqs. 3.19a or 3.19b. Here, what we seek is an approximate relation between the increments in relative coordinates $(\delta\mathbf{r}_{ij}, \delta\mathbf{v}_{ij})$ and those in absolute coordinates $(\delta\mathbf{r}_i, \delta\mathbf{v}_i)$, in order to construct the full integrator. By noting that increments associated with operators $\widehat{H}_{U_{ij}}$ and $\widehat{H}_{U_i}$ are related by

$$\tau \begin{pmatrix} \mathbf{0} \\ \mathbf{a}_i \end{pmatrix} = \frac{1}{m_i} \sum_{j\neq i}^{N} \mu_{ij} \tau \begin{pmatrix} \mathbf{0} \\ \mathbf{a}_{ij} \end{pmatrix} , \tag{3.22}$$

a way to specify $(\delta\mathbf{r}_i, \delta\mathbf{v}_i)$ consists of exploring the equivalence between H_U and H_W, as first presented in Eq. 3.4. In addition, if we take into account the discussion above regarding to eqs. 3.10, 3.11, 3.20 and 3.21, a relation between relative and absolute increments can be defined in analogy to Eq. 3.22 as follows:

$$\begin{pmatrix}\delta\mathbf{r}_i\\ \delta\mathbf{v}_i\end{pmatrix} = \frac{1}{m_i}\sum_{j\neq i}^{N}\mu_{ij}\begin{pmatrix}\delta\mathbf{r}_{ij}\\ \delta\mathbf{v}_{ij}\end{pmatrix} + \mathcal{O}(\tau^2)\,, \tag{3.23}$$

which constitutes a first-order approximation as explained above (see Eq. 3.21). While we were not able to provide a more formal derivation to Eq. 3.23, we will show below (see explanation about Eq. 3.28) that when we calculate the relative increments from an ordinary Leapfrog map rather than the Kepler-solver in Eq. 3.18, then Eq. 3.23 reduces to Eq. 3.22.

We can now define a time-symmetric second-order map for our Keplerian-based Hamiltonian splitting as follows:

$$\begin{aligned}\Psi(\tau) &\equiv \phi(\frac{\tau}{2})\circ\phi^{\dagger}(\frac{\tau}{2})\,,\\ &\equiv e^{\frac{\tau}{2}\widehat{H}_T}e^{\frac{\tau}{2}\widehat{H}_W}\circ e^{\frac{\tau}{2}\widehat{H}_W}e^{\frac{\tau}{2}\widehat{H}_T}\,,\end{aligned} \tag{3.24}$$

where the increments $(\delta\mathbf{r}_{ij}, \delta\mathbf{v}_{ij})$ which appear in the $e^{\frac{\tau}{2}\widehat{H}_W}$ operator on the left side of $\circ$ are independently obtained after application of Eq. 3.19a for each $i-j$ pair, while those which appear on the right side of $\circ$ are independently obtained after application of the (self-adjoint) method in Eq. 3.19b for each $i-j$ pair. Eq. 3.24 can be further simplified by merging operators on both sides of $\circ$, giving,

$$\Psi(\tau) \equiv e^{\frac{\tau}{2}\widehat{H}_T}e^{\tau\widehat{H}_W}e^{\frac{\tau}{2}\widehat{H}_T}\,, \tag{3.25}$$

in which case the increments $(\delta\mathbf{r}_{ij}, \delta\mathbf{v}_{ij})$ appearing in the $e^{\tau\widehat{H}_W}$ operator should be independently obtained after application of a time-symmetric second-order map for each $i-j$ pair,

$$e^{\tau\left(\widehat{H}_{K_{ij}}-\widehat{H}_{T_{ij}}\right)} \approx e^{\frac{\tau}{2}\left(-\widehat{H}_{T_{ij}}\right)}e^{\tau\widehat{H}_{K_{ij}}}e^{\frac{\tau}{2}\left(-\widehat{H}_{T_{ij}}\right)}\,. \tag{3.26}$$

The equations of motion that result from the full map in Eq. 3.25 can be written in the following discrete form:

$$\mathbf{r}_i^{1/2} = \mathbf{r}_i^0 + \frac{\tau}{2}\mathbf{v}_i^0 \,, \tag{3.27a}$$

$$\tilde{\mathbf{r}}_i = \mathbf{r}_i^{1/2} + \frac{1}{m_i}\sum_{j\neq i}^{N} \mu_{ij}\delta\mathbf{r}_{ij} \,, \tag{3.27b}$$

$$\mathbf{v}_i^1 = \mathbf{v}_i^0 + \frac{1}{m_i}\sum_{j\neq i}^{N} \mu_{ij}\delta\mathbf{v}_{ij} \,, \tag{3.27c}$$

$$\mathbf{r}_i^1 = \tilde{\mathbf{r}}_i + \frac{\tau}{2}\mathbf{v}_i^1 \,, \tag{3.27d}$$

where $\mathbf{r}_i^1 = \mathbf{r}_i(t+\tau)$, $\mathbf{r}_i^0 = \mathbf{r}_i(t)$ and similar for $\mathbf{v}_i$, and the increments $(\delta\mathbf{r}_{ij}, \delta\mathbf{v}_{ij})$ are calculated independently as explained above.

As it can be seen, eqs. 3.27 are remarkably similar to the Leapfrog method. It remains to be shown that these equations effectively reduce to the Leapfrog equations when we substitute the 2-body Kepler-solver to a simple DKD-type integrator. In this case, the map in Eq. 3.26 becomes:

$$\mathbf{r}_{ij} \leftarrow \mathbf{r}_{ij} - \frac{\tau}{2}\mathbf{v}_{ij} \,, \tag{3.28a}$$

$$\mathbf{r}_{ij} \leftarrow \mathbf{r}_{ij} + \frac{\tau}{2}\mathbf{v}_{ij} \,, \tag{3.28b}$$

$$\mathbf{v}_{ij} \leftarrow \mathbf{v}_{ij} + \tau\,\mathbf{a}_{ij} \,, \tag{3.28c}$$

$$\mathbf{r}_{ij} \leftarrow \mathbf{r}_{ij} + \frac{\tau}{2}\mathbf{v}_{ij} \,, \tag{3.28d}$$

$$\mathbf{r}_{ij} \leftarrow \mathbf{r}_{ij} - \frac{\tau}{2}\mathbf{v}_{ij} \,, \tag{3.28e}$$

which results in $\delta\mathbf{v}_{ij} = \tau\mathbf{a}_{ij}$ and $\delta\mathbf{r}_{ij} = \mathbf{0}$ and, in view of eqs. 3.22 and 3.23, completes the demonstration. It should be noted that in this particular case, the error in Eq. 3.23 disappears because $\delta\mathbf{r}_{ij} = \mathbf{0}$ and Eq. 3.21 reduces to Eq. 3.20, restoring the symplecticity of the method. Note also that this is true only if we use a DKD-type integrator as a 2-body solver. For a KDK-type 2-body solver the symplecticity of the method is not restored because the order in which $(\mathbf{r}_{ij}, \mathbf{v}_{ij})$ is evolved in eqs. 3.28 changes and $\delta\mathbf{r}_{ij} \neq \mathbf{0}$. In other words, using a simple DKD-type integrator as a 2-body solver in the scheme above results in a very expensive implementation of a traditional Leapfrog method.

On the other hand, with the Kepler-solver function as a 2-body solver, a non-Hamiltonian error is made due to the non-commutativity

of the $e^{\tau \widehat{H}_{W_{ij}}}$ operators and the fact that each $i-j$ pair is treated independently, leading to the loss of symplecticity of the resulting method. Because our Keplerian-based integrator is constructed as a composition of self-adjoint first-order maps, it still preserves time-reversibility and second-order convergence (error $\mathcal{O}(\tau^3)$).

The advantage of using the Kepler-solver instead, comes from the fact that it is guaranteed that all pairwise interactions are always integrated precisely, which, in practical N-body simulations, is a much stronger requirement than the symplecticity of the Hamiltonian flow.

3.2.3 Implementation

The method described in the previous section has been implemented in a new code called Sakura, which is available in Astrophysical MUltipurpose Software Environment (AMUSE[3], **?**). In order to clarify the implementation, Listing 3.1 shows a Python[4] code for the main loop calculation which evolves the particle's coordinates according to the map in Eq. 3.25 or, equivalently, eqs. 3.27. The Kepler-solver function at line 47 implements a universal variable Kepler-solver closely following **?**. Note that the memory and CPU requirements of this code scales as $O(N)$ and $O(N^2)$, respectively.

Listing 3.1: `Python` code for the main loop in Sakura integrator

```
1  """The functions below implement the main
2  steps of Sakura integrator.
3
4  The required parameters are the following:
5
6  :param tau: the time-step size.
7  :param n: the number of particles.
8  :param m: array with particles' masses.
9  :param r: 3D array with particles' positions.
10 :param v: 3D array with particles' velocities.
11 """
12
13 def do_step(tau, n, m, r, v):
14   r, v = evolve_HT(tau/2, n, m, r, v)
15   r, v = evolve_HW(tau, n, m, r, v)
16   r, v = evolve_HT(tau/2, n, m, r, v)
17   return r, v
18
19 def evolve_HT(tau, n, m, r, v):
```

[3] www.amusecode.org

[4] The actual implementation has been done in C/C++ for efficiency purposes.

```
  for i in range(n):
    for k in range(3):
      r[i][k] += v[i][k] * tau
  return r, v

def evolve_HW(tau, n, m, r, v):
  # Allocate/initialize 3D arrays to store
  # increments in position/velocity due to
  # 2-body interactions.
  dmr = numpy.zeros((n, 3))
  dmv = numpy.zeros((n, 3))

  # For each i-j pair, this corresponds to
  # the Eq. 26 in the main text.
  for i in range(n):
    for j in range(n):
      if i != j:
        mij = m[i] + m[j]
        mu = m[i] * m[j] / mij
        for k in range(3):
          rr0[k] = r[i][k] - r[j][k]
          vv0[k] = v[i][k] - v[j][k]
        ###
        for k in range(3):
          r0[k] = rr0[k] - vv0[k] * tau / 2
          v0[k] = vv0[k]
        #
        r1, v1 = kepler_solver(tau, mij, r0, v0)
        #
        for k in range(3):
          rr1[k] = r1[k] - v1[k] * tau / 2
          vv1[k] = v1[k]
        ###
        for k in range(3):
          dmr[i][k] += mu * (rr1[k] - rr0[k])
          dmv[i][k] += mu * (vv1[k] - vv0[k])

  # This corresponds to eqs. 27b and 27c
  # in the main text.
  for i in range(n):
    for k in range(3):
      r[i][k] += dmr[i][k] / m[i]
      v[i][k] += dmv[i][k] / m[i]
  return r, v
```

3.3 VALIDATION AND PERFORMANCE

In order to verify that Sakura performs well on collisional N-body systems, we present some tests for N ranging from a few to a thousand. We compare the results of Sakura to those obtained using a modified version of the Leapfrog integrator and a standard 4-th order Hermite integrator, available in the AMUSE framework. The modification in the Leapfrog integrator consists of the introduction of a routine to allow the use of adaptive time-steps. In this case the time-symmetry of the Leapfrog method is still preserved because we adopted the recipe for time-symmetrisation as suggested in **?**. A comparison of the computational costs and scalings with N is also presented. We emphasize that the base time-step size in each of the tests of Sakura is kept constant during the simulation, whilst in Leapfrog and Hermite integrations a shared adaptive time-step scheme has been adopted. The time-step criterion used within Leapfrog integrations is the time-symmetrized version of $\tau \sim \min((r_{ij}/a_{ij})^{1/2})$, whilst in Hermite code the standard Aarseth-criterion is used. For other details about these codes we refer the reader to the AMUSE documentation[3]. The value of the constant time-step size in Sakura is chosen in such a way that the same number of integration steps is taken as in the case of the Hermite integrations. Similarly, the time-step parameter in Leapfrog integrations is chosen to give approximately the same number of steps as in Hermite integrations. Note that, by construction, Sakura does not admit any softening parameter. Therefore, we also use zero softening in the other methods.

3.3.1 Small-N Systems

We start by presenting some numerical tests for well known simple small-N systems including the figure-eight system ($N = 3$; **?**), the Pythagorean system ($N = 3$; **?**) and the sun with planets[5] ($N = 10$; Ito & Tanikawa (2002)). We do not show results for a single binary system ($N = 2$), since in this case Sakura reduces to an ordinary Kepler-solver which gives a solution for the binary orbit accurate to machine precision. The simulation time spans 100 N-body units (**?**) in the case of the first two systems and 10^3 yr in the case of the solar system.

In Fig. 3.1, we present the relative energy error as a function of the average time-step size (left panels) and CPU time vs relative

[5]We include Pluto in our simulations of the solar system since we use the initial conditions as given in Ito & Tanikawa (2002).

energy error (right panels) for the figure-eight system (top panels), Pythagorean system (middle panels) and sun with planets (bottom panels), for the Leapfrog, 4-th order Hermite and Sakura. We note that for the figure-eight system the 4-th order Hermite usually performs better than Leapfrog and Sakura for a level of energy conservation $\lesssim 10^{-6}$. We attribute this to the fact that in this system the intrinsic time-step size of the particles does not change considerably during the orbital evolution and then, for smaller τ, the 4-th order convergence rate of the Hermite integrator outperforms Leapfrog and Sakura, which are of 2-nd order. We notice that in this case, where all three particles democratically interact among themselves, Sakura is not expected to be the most suitable method of integration due to the non-commutativity of 2-body interactions. Nevertheless, as we see in Fig. 3.1 (top panels), its performance is comparable to that of the Leapfrog integrator. For the Pythagorean system, which contains several close encounters between particles during its orbital evolution, all three integration methods are somewhat comparable, despite Sakura using constant time-steps and the other two methods using adaptive time-steps. For the solar-system, in which the orbital evolution of the planets is almost Keplerian, Sakura delivers about four orders of magnitude better energy conservation than Leapfrog, being also more precise than Hermite integration for time-steps $\gtrsim 10^{-3}$, while consuming the least amount of CPU time.

For those kind of systems, an integration step using Sakura is usually more expensive than an integration step using Hermite or Leapfrog by a factor $2-4$. Also, since all these codes scales as $O(N^2)$, these figures are expected to remain unchanged when the number of particles increases. However, due to the fact that Sakura can handle compact binaries and/or resolve close encounters even with constant τ, less time-steps are required for a given level of energy conservation implying that in these cases Sakura might outperform Hermite and Leapfrog integrations. In order to confirm this, we also include a test with a specially constructed initial condition which consists of a hierarchical binary system ($N = 4$) with two tight binaries orbiting around each other in a circular orbit with semi-major axis $a_{\rm outer} = 1$ (N-body units). The particles in each tight binary are themselves in a circular orbit with semi-major axis $a_{\rm inner}$. We have selected a semi-major axis ratio in the range $a_{\rm outer}/a_{\rm inner} = 10 - 1000$, and performed a simulation for these systems for a time span of one $P_{\rm outer}$, i.e., the largest orbital period in the system (which is the same for all semi-major axis ratios). In Fig. 3.2 we present the relative energy error as a function of

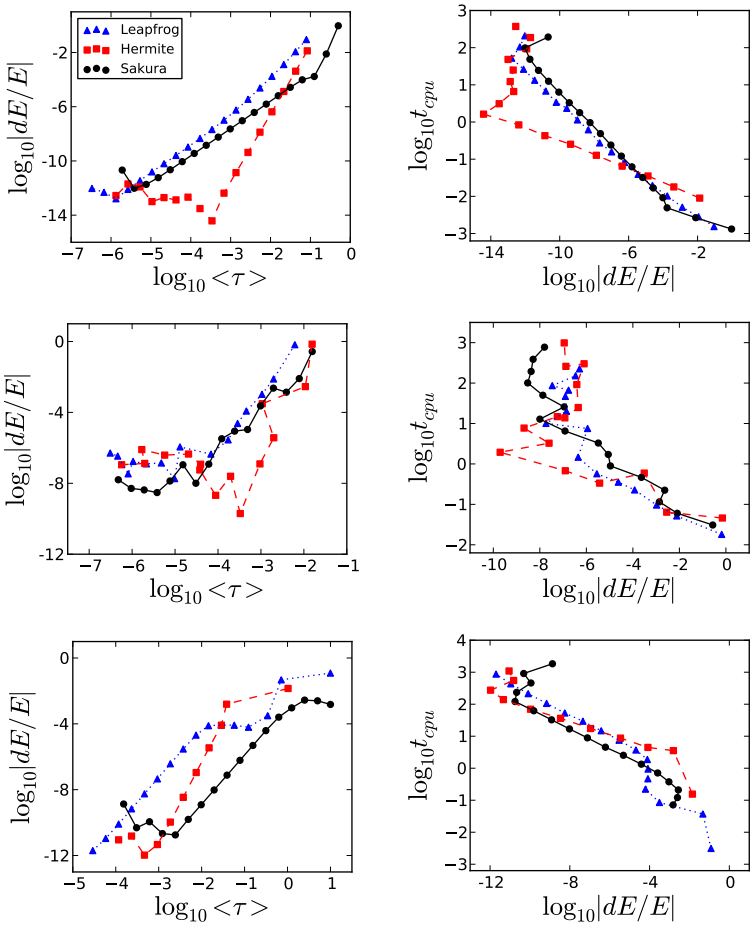

Figure 3.1: Relative energy error as a function of the average time-step size (left panels) and CPU time (in seconds) vs relative energy error (right panels) for the Leapfrog integrator (triangles), 4-th order Hermite (squares) and Sakura (bullets) for three different systems: figure-eight system (top panels), Pythagorean system (middle panels) and sun with planets (bottom panels). $\langle\tau\rangle$ is given in N-body units and stands for the average value of the shared adaptive time-step size in Hermite integrations.

the time-step size (left panels) and CPU time vs relative energy error (right panels) for the 4-th order Hermite, Leapfrog and Sakura.

For the $a_{\rm outer}/a_{\rm inner} = 10$ case (top panels in Fig. 3.2), Sakura delivers the same level of energy conservation as Leapfrog, although being more time consuming, whilst 4-th order Hermite has better energy conservation due to its higher order convergence for time-step sizes $\lesssim 10^{-2}$. However, for tighter interacting binaries (middle and bottom panels in Fig. 3.2), Sakura shows increasingly better performance with the compactness of the interacting binaries. In particular, for a level of energy conservation of 10^{-6}, typically adopted in collisional N-body simulations, Sakura is more than a order of magnitude faster than Hermite for the tightest binary configuration, $a_{\rm outer}/a_{\rm inner} = 1000$, while having a similar speed as Leapfrog. Also for the tightest binary configuration, Sakura is the most precise integration method for a range in time-steps of 6 orders of magnitude. On the other hand, for this latter system, the 4-th order Hermite results only start converging to good energy conservation when using time-steps $\lesssim 10^{-5.5}$, which in some circumstances might be impractical in computational terms, when systems of this kind are present in a large-scale simulation.

3.3.2 Large-N Systems

To test how Sakura behaves with a more general N-body problem, we use as initial condition a 128-body Plummer sphere containing a black-hole in its center. We assume equal mass for the stars and construct the system in virial equilibrium but for different black-hole to star mass ratios, $q \equiv M_{\rm bh}/M_{\rm star}$, ranging from $q = 1$ (no black-hole) to $q = 10^{12}$. We performed simulations for each of these initial conditions for 1 N-body time unit. Once again, the performance of Sakura is compared with that of the Leapfrog and standard 4-th order Hermite integrators. The results are shown in Fig. 3.3 which presents the relative energy error as a function of the mass ratio for time-step sizes $\langle\tau\rangle = 10^{-3}, 10^{-4}, 10^{-5}$ (top, middle and bottom lines), and Fig. 3.4 which present the CPU time vs relative energy error for different mass ratios: $q = 10^3$ (top left), $q = 10^6$ (top right), $q = 10^9$ (bottom left) and $q = 10^{12}$ (bottom right).

In Fig. 3.3 we see that the relative energy error for all three methods initially increases with the mass ratio till the point when $q \sim 10^2$. For larger mass ratios, the behaviour of Sakura clearly differs from the other two methods. While in Leapfrog and Hermite integrators the energy error stabilizes at a certain level, in Sakura we observe a very interesting trend in which its energy error decreases with increasing

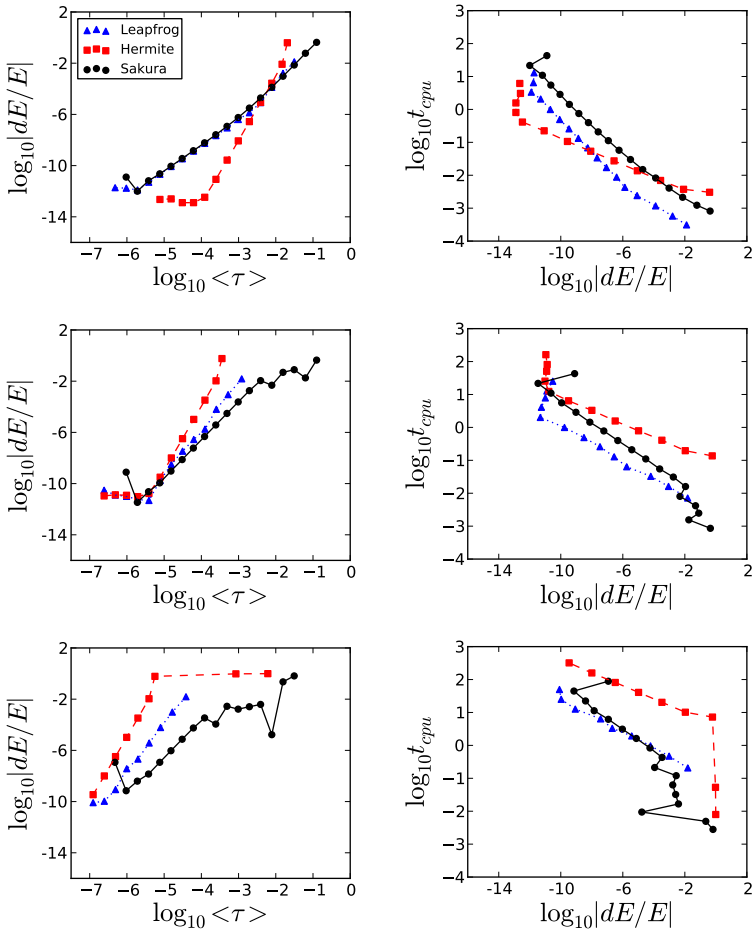

Figure 3.2: The same as Fig. 3.1 but for the hierarchical binary system for the following semi-major axis ratios: $a_{\rm outer}/a_{\rm inner} = 10$ (top panels), $a_{\rm outer}/a_{\rm inner} = 100$ (middle panels), $a_{outer}/a_{inner} = 1000$ (bottom panels).

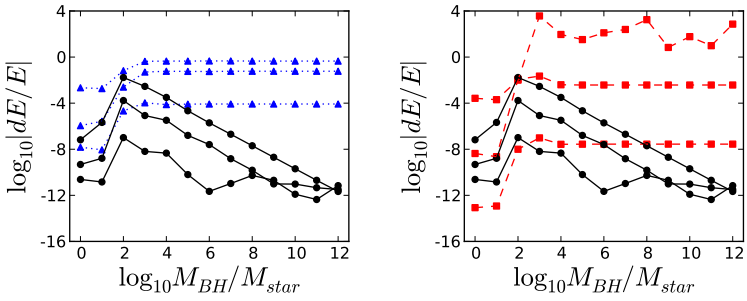

Figure 3.3: For a Plummer sphere with a central black hole, the panels show a comparison of the relative energy error as a function of the black-hole to stellar mass ratio for time-step sizes $\langle\tau\rangle = 10^{-3}, 10^{-4}, 10^{-5}$ (top, middle and bottom lines). The left panel present the results for Leapfrog (triangles) and Sakura (bullets) and the right panel present the results for 4-th order Hermite (squares) and Sakura (bullets).

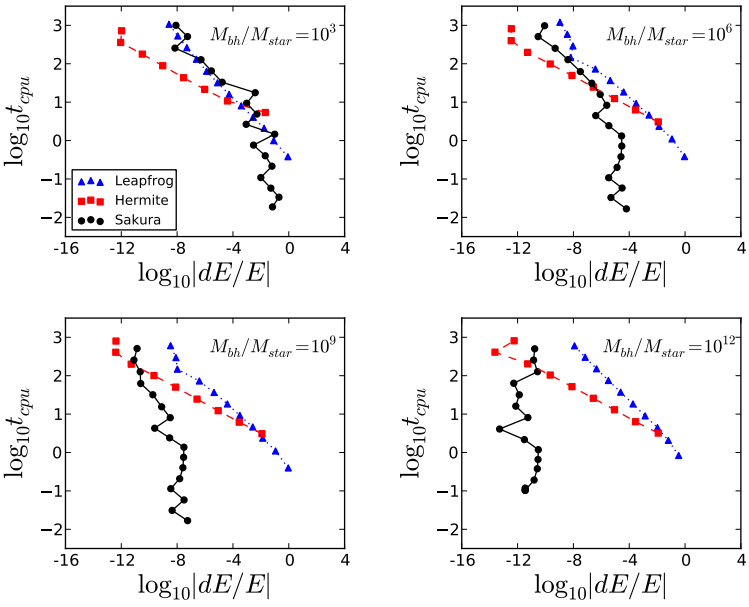

Figure 3.4: For the same system as in Fig. 3.3, the panels show the CPU time (in seconds) vs relative energy error for the following mass ratios: $q \equiv M_{\rm bh}/M_{\rm star} = 10^3$ (top left), $q = 10^6$ (top right), $q = 10^9$ (bottom left) and $q = 10^{12}$ (bottom right).

mass ratio. In other words, Sakura becomes more precise and therefore more efficient when the mass ratio grows, as can be seen in Fig. 3.4 for mass ratios (top left), $q = 10^6$ (top right), $q = 10^9$ (bottom left) and $q = 10^{12}$ (bottom right). An explanation of why these methods behave this way is as follows.

When no dominant massive particle is present in the system ($q \sim 1 - 10$), after only 1 N-body time unit the system has not evolved for enough time to form a close binary (which is an outcome of strong few-body interactions, see e.g. **?**). Therefore, in these circumstances most of the particles interact weakly among themselves and all the methods are able to integrate the orbital evolution of stars with relatively good energy conservation. Around a mass ratio $q \sim 10 - 10^3$ the massive particle quickly forms a binary system with a close neighbour, which eventually experiences several interactions with close perturbers, thus deteriorating the precision of the integration in all three methods. For mass ratios $q \gtrsim 10^3$ the orbital motion of stars becomes predominantly Keplerian. In this regime, the orbits in the system become mostly regular, and close encounters between stars become gradually less important. Therefore, the energy error is expected to converge to the truncation error associated to each of these methods. In Leapfrog and Hermite integrators, by decreasing the time-step size the energy conservation is thus improved but it remains approximately at the same level of conservation regardless the mass ratio (for $q \gtrsim 10^3$). On the other hand, Sakura departs from a constant level of energy conservation observed in the other two integrators, and becomes increasingly more precise with the mass ratio. This happens because in Sakura, the truncation error comes from two different sources: i) the error due to the Kepler-solver, which is essentially at machine precision, and ii) the error associated to the non-commutativity of 2-body interactions in close multiple-body encounters. With this knowledge, it is easy to intuitively understand why Sakura becomes more precise with the increase of the mass ratio: simply because the error associated to the non-commutativity of 2-body interactions becomes less important and, thus the overall error of the integrator converges to that of the Kepler-solver.

For $\langle \tau \rangle \sim 10^{-4}$, which corresponds to the middle lines (for each integrator) in Fig. 3.3, Sakura is ~ 5 (~ 6) orders of magnitude more precise than Hermite (Leapfrog), for a mass ratio $q = 10^6$. Also, as is shown in Fig. 3.4, Sakura's performance is similar to Leapfrog, for a mass ratio $q = 10^3$, and becomes gradually more efficient than Hermite and Leapfrog, when the mass ratio increases. This happens due to a change in slope of Sakura's curves in panels showing the CPU

time vs relative energy error when the mass ratio goes from $q = 10^3$ to $q = 10^{12}$ in Fig. 3.4, which means that for mass ratios $q \gtrsim 10^{12}$, Sakura can give very accurate results ($dE/E \sim 10^{-10} - 10^{-12}$) even when using relatively large time-steps, thus saving a big amount of computational time compared to Leapfrog and Hermite integrators.

As an additional general N-body test we performed a simulation of a 1024-body system through core collapse using Sakura with several time-step sizes $\tau = 10^0, 10^{-1}, 10^{-2}, 10^{-4}$, and the Leapfrog and standard 4-th order Hermite code using shared adaptive time-steps. For the parameter of precision we choose $\eta = 2^{-5} \approx 0.03$ in order to have a level of energy conservation of about 10^{-4} by the moment of core collapse in Hermite integration. In this particular test, we have used a parallel version of Sakura (see section 3.4) running on a 4-core Intel Xeon CPU @2.40 GHz. For the Leapfrog and Hermite codes (which are also parallelised) we setup the number of MPI processes to 4. In Fig. 3.5 we present the time evolution of the core radius using these codes. We see from this figure that for a sufficiently small time-step size ($\tau \sim 10^{-4}$, lowest black curve in Fig. 3.5) Sakura is able to evolve the system through core collapse. As expected from the exponential orbital instability (**?**), the results from Sakura slightly differ from Hermite and Leapfrog calculations. Apart from that, the core radius evolution obtained using Sakura follows remarkably well the results from the other two integrators.

In Sakura, the appearance of close binaries does not represent a computational challenge. Therefore, in this simulation no slow down in performance is observed, as is the case in most other N-body codes that also try to correctly evolve such compact sub-systems. As a consequence, the most expensive simulation using Sakura (bottom black line in Fig. 3.5) was completed in about three days of CPU time. The Leapfrog integration took about a week of processing time, whereas the Hermite simulation, after more than a month of CPU time (on the same machine), had not been completed, due to the dynamical formation of very close binaries and consequent decrease of the adaptive time-step size.

Although Sakura integrates all pairwise interactions exactly, the presence of close perturbers for a particular $i - j$ pair represents the main source of error during the integration. The reason for that originates from our assumption that each pair of particles can be treated as an independent 2-body problem during a time-step τ. If τ is larger than the time scale of interaction between the $i - j$ pair and its perturber, the perturbation will be delayed by τ, leading to spurious integration of a tight multi-component sub-system in an N-body sim-

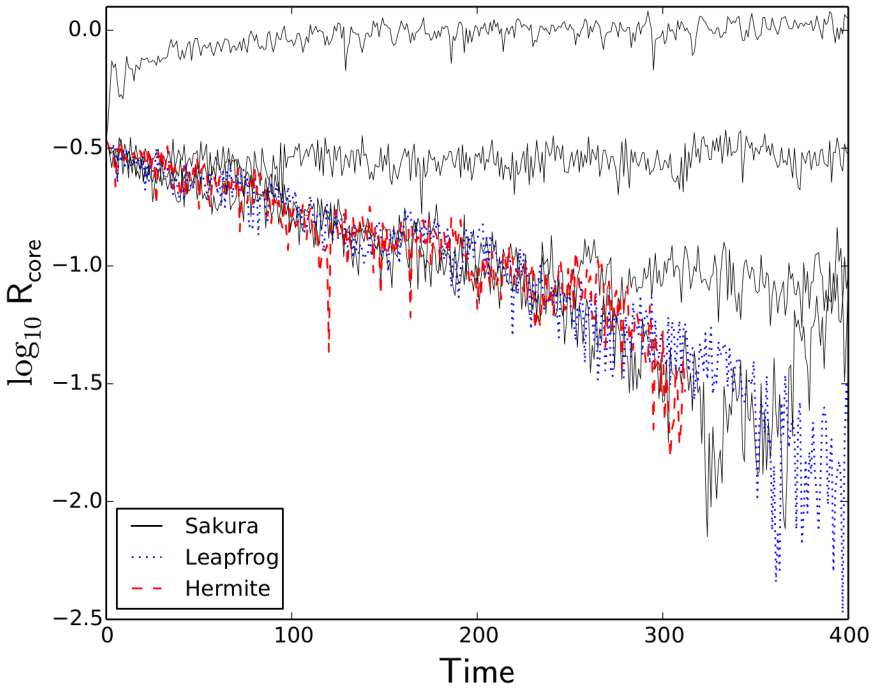

Figure 3.5: Core radius vs simulation time for a 1024-body Plummer sphere. We compare Sakura using different time-step sizes (solid lines, $\tau = 10^0, 10^{-1}, 10^{-2}, 10^{-4}$ from top to bottom) to Leapfrog (dotted line) and standard 4-th order Hermite (dashed line), using shared adaptive time-steps with a parameter of precision $\eta \approx 0.03$. All the quantities are presented in N-body units.

ulation. This is a consequence of the non-commutativity of 2-body interactions. In Fig. 3.5, the use of relatively large time-steps reveals this issue: although the system as a whole stays bound, strong few-body interactions in the cluster core are not correctly integrated and as a consequence the core radius expands. However, by using smaller τ the numerical issues due to strong perturbations on the $i-j$ pair is diminished and as a consequence Sakura evolves the multi-component sub-systems that may form dynamically during the simulation more precisely. In those calculations, the level of energy conservation at the moment of core collapse stayed within $dE/E \lesssim 10^{-4}$ for Hermite, and $dE/E \lesssim 10^{-2}$ for Leapfrog and Sakura (for the bottom black line in Fig. 3.5), even though Sakura used a constant time-step.

The possibility to include a variable time-step scheme in Sakura might improve its results and is currently under investigation. The fact that Sakura evolves each pair of particles exactly, implies that the time-step criterion does not need to be so restrictive as in the case of traditional integration schemes. For example, if we consider the case of a hierarchical triple system in which the orbital period of the inner binary is a certain factor shorter than the time-scale of interaction between the binary and the outer perturber, we have observed in our tests (not reported here) that choosing a time-step size comparable to the longest time-scale still preserves the binary orbital evolution. In traditional codes, this would not be possible and the inner binary would end up being artificially disrupted if the time-step size has not been decreased to a fraction of its orbital period. Therefore, for Sakura we suspect that a time-step criterion based on the closest perturber distance to a given pair being evolved seems to be a more appropriate choice than an Aarseth-like time-step criterion. We will further discuss this issue on section 3.5.

3.4 PARALLELIZATION

We have implemented three different versions of Sakura: i) a single GPU implementation using OpenCL; ii) a distributed memory parallel implementation using MPI, and iii) a serial implementation in C/C++ (used in all the tests presented above, with exception of the one in Fig. 3.5, for which the MPI version was used). The parallelisation schemes adopted for distributed memory and GPU versions are quite similar as those adopted for conventional N-body codes on those platforms (see Portegies Zwart et al. 2008 and **?**, respectively). At the current stage of development our GPU implementation is not yet very efficient due to many branch conditions present in the Kepler-solver.

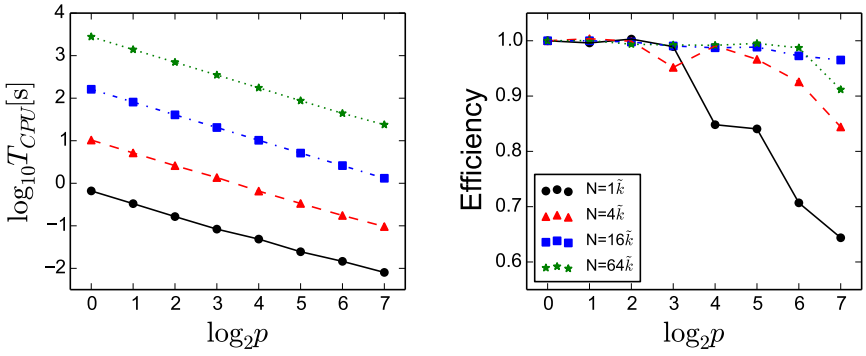

Figure 3.6: For the MPI version of Sakura the plots show the strong scaling (left panel) and the parallel efficiency (right panel) for four different problem sizes: $N = 1\tilde{k}$ (solid lines), $N = 4\tilde{k}$ (dashed lines), $N = 16\tilde{k}$ (dot-dashed lines) and $N = 64\tilde{k}$ (dotted lines). Here, $\tilde{k}$ stands for 1024 and p is the number of processor cores used for the run.

Here we mainly present some performance results using the MPI version of Sakura for tests using up to 128 CPU cores. The test simulations consist of a Plummer sphere with N equal mass particles being integrated for 1 N-body time unit. We use four different number of particles $N = 1\tilde{k}$, $4\tilde{k}$, $16\tilde{k}$, $64\tilde{k}$ ($\tilde{k}$ stands for 1024) and in each case we measure the total wall clock time needed to complete the simulation with different number of cores. In Fig. 3.6 we present, for four different problem sizes, the performance measurements in the form of the strong scaling ($T_{CPU}(p)$ vs p) and the parallel efficiency:

$$\text{Efficiency} \equiv \frac{T_{CPU}(p)}{pT_{CPU}(1)}\,, \tag{3.29}$$

where $T_{CPU}(p)$ is the CPU time measured when using p processor cores.

As is evident from the Fig. 3.6, Sakura exhibits an almost perfect strong scaling (top panel) and a remarkably good parallel efficiency (bottom panel). For the worst case scenario presented here ($N = 1\tilde{k}$, using 128 CPU cores), Sakura achieves a parallel efficiency as good as 64%, even though the workload in this case is as small as 8 particles per core. In addition, the strong scaling plot shows that, even in this worst case scenario, the CPU time could still be decreased by using a higher number p of processor cores. For $N > 4\tilde{k}$, the parallel efficiency of Sakura stays very close to 100%.

3.5 SUMMARY AND DISCUSSION

We have described a Keplerian-based Hamiltonian splitting for gravitational N-body simulations and its implementation in a new code called Sakura. In this method a general N-body problem can be solved as a composition of multiple, independent, 2-body problems. The integration scheme is constructed on the assumption that, during a small time interval τ, each pair of particles in the system can be treated as an independent 2-body problem. With this splitting an analytical Kepler-solver can be used to accurately, and independently, evolve each 2-body interaction in the system, thus making the code especially suitable for simulations in which compact primordial binaries or close dynamically formed binaries are present. Hierarchies in which one of the components is a compact binary and systems with a central dominant mass are also examples of physical systems in which Sakura performs well when compared to traditional codes.

Because Sakura can easily handle arbitrarily compact binaries in an N-body simulation, the code is able to evolve a star-cluster through core-collapse without much difficulty. In particular, since Sakura can do this even with the use of constant time-steps, the simulation does not suffer from any slow down in performance as is the case in other non-regularized N-body codes. As an example, in the 1024-bodies core-collapse simulation presented in section 3.3.2, Sakura was able to complete the run in about 3 days of CPU time on a 4-core machine. The same system being integrated with a 4-th order Hermite integrator took more than one month of CPU time on the same machine, due to a severe slow down in performance after the formation of the first hard binary in the system.

There are, however, some circumstances in which Sakura may not be the most suitable code to perform an N-body simulation. For example, for systems in which multiple bodies democratically interact among themselves, Sakura may perform almost as badly as a simple Leapfrog integrator, as demonstrated in the integrations of a figure-eight system in section 3.3.1. This happens because of our underlying assumption that the N-body problem can be decomposed in multiple, independent, 2-body problems. Such decomposition in fact constitutes the main source of error when a given $i-j$ pair is being integrated with a time-step τ which is larger than the time-scale of the perturbation due to a close neighbour. In many cases this issue may be surpassed by decreasing the constant time-step size used in the simulation. However, the cause of the problem lies on the non-commutativity of 2-body interactions when multiple bodies are involved in a democratic close

encounter. While it is not easy to solve this issue without breaking our Keplerian splitting approach, the introduction of an adaptive time-step scheme in Sakura might alleviate these numerical difficulties and is currently under investigation.

According to some of our tests (not reported in the present paper), a time-step criterion based on the strength of the perturbation on a given $i-j$ pair seems to work relatively well compared to a constant τ. However, this improvement is only significant when close multiple-body encounters take place. On the other hand, one could in principle choose $\tau \sim \min(r_{ij}/v_{ij})$, $\tau \sim \min((r_{ij}/a_{ij})^{1/2})$ or use a traditional Aarseth-like time-step criterion, but we advocate that this may not be the optimal choice because these criteria also include the contribution of the $i-j$ pair itself, which in principle contributes to a severe decrease in time-steps if a close binary is present in the system. In Sakura, these severely short time-steps are not necessary, because the use of a Keplerian treatment for each pair of particles automatically regularizes every 2-body interaction in the system. It is only when multiple-body encounters happens that the time-step should adapt itself to properly resolve the approximation of a perturber. Therefore, we stress here our preference for a perturbation-based time-step criterion rather than an Aarseth-like criterion for use in Sakura. Whether or not such perturbation-based criterion is the best choice for Sakura is a matter that will be addressed elsewhere.

Another point we want to emphasize here is the behaviour of Sakura when integrating a system with a central massive black-hole. As shown in Fig. 3.3, the level of energy conservation in Leapfrog and 4-th order Hermite integrations remains approximately constant with the increase of the black-hole to stellar mass ratio. For Sakura, we found that it performs much better than previous approaches, becoming gradually more precise with the increase of the mass ratio. In particular, for the case of a mass ratio $q = 10^6$ Sakura can give $\gtrsim 5$ orders of magnitude better energy conservation than Hermite integrator, being at the same time up to 4 orders of magnitude faster when the mass ratio increases to $q \gtrsim 10^9$. The fact that Sakura can be, at the same time, fast and accurate in this regime, makes this code highly suitable for nearly Keplerian systems where a massive particle dominates the evolution of surrounding particles, such as in planetary systems and galactic nuclei with super-massive black-holes.

Lastly, Sakura has proven to be quite easy to parallelise for distributed memory systems using MPI. The GPU implementation, even though theoretically easy, is still not totally efficient due to the presence of many branching conditions in the Kepler-solver. In algorithmic

terms, the bulk of computation in Sakura occurs inside a double loop, similar to the one used to calculate the acceleration of particles in conventional N-body codes. Therefore, we were able to immediately employ existent parallelisation schemes in Sakura without much effort. We argue that the fact that our GPU implementation is not yet very efficient is not a problem due to the parallelisation scheme itself, but rather due to the poor/inefficient support for branch conditions in current GPUs. A restructure in our Kepler-solver in order to eliminate (or minimize) these branch conditions may address this issue, and possibly speed up even more the MPI version on CPUs, which has already shown a remarkable parallel efficiency, with close to 100 percent efficiency for $16\tilde{k}$ particles on 128 cores, and 64 percent efficiency when using only eight particles per core.