Planetologie

# GPU accelerated n-Body integrators for long-term simulations of planetary systems

Inaugural-Dissertation
zur Erlangung des Doktorgrades der
Naturwissenschaften im Fachbereich Geowissenschaften
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von
Stephan Hellmich

2017

| | |
|---|---|
| Dekan: | Prof. Dr. Harald Strauß |
| Erstgutachter: | Prof. Dr. Tilman Spohn |
| Zweitgutachter: | Prof. Dr. Jürgen Oberst |
| Tag der mündlichen Prüfung: | 21. Dezember 2017 |
| Tag der Promotion: | ______________________ |

# Abstract

Simulations of dynamical processes in planetary systems represent an important tool for studying their orbital evolution [7][82][48]. Using modern numerical integration methods, it is possible to model systems containing many thousands of objects over time scales of several hundred million years. However, in general supercomputers are needed to get reasonable simulation results in acceptable execution times [48]. To exploit the ever growing computation power of Graphics Processing Units (GPUs) in modern desktop computers I implemented cuSwift, a library of numerical integration methods for studying long-term dynamical processes in planetary systems. cuSwift can be seen as a re-implementation of the famous SWIFT integrator package written by Levison and Duncan. cuSwift is written in C/CUDA and contains different integration methods for various purposes. So far, I have implemented three algorithms: a 15th order Radau integrator [40], the Wisdom-Holman Mapping (WHM) integrator [115] and the Regularized Mixed Variable Symplectic (RMVS) Method [65]. These algorithms treat only the planets as mutually gravitationally interacting bodies whereas asteroids and comets (or other minor bodies of interest) are treated as massless test particles which are gravitationally influenced by the massive bodies but do not affect each other or the massive bodies. The main focus of this work is on the symplectic methods (WHM and RMVS) which use a larger time step and thus are capable of integrating many particles over very large time spans. As an additional feature, I implemented the non-gravitational Yarkovsky effect as described by Brož [13].

With cuSwift I show that the use of GPUs makes it possible to speed up these methods up to two orders of magnitude compared to the single-core CPU implementation, thereby enabling modest workstation computers to perform long-term dynamical simulations. I compare the newly implemented algorithms with the original algorithms of SWIFT in order to assess the numerical precision and to demonstrate the speedup achieved with the GPU. As a first application, I use these methods to study the influence of the Yarkovsky effect on Jupiter Trojan asteroids and show that the effect indeed affects the long-term orbital evolution of these bodies.

## Acknowledgements

I am grateful for the support by the German Aerospace Center I received while working on this thesis.

I would like to express my sincere gratitude to my advisor Prof. Tilman Spohn for giving me the opportunity to write this PhD thesis and for his professional advise.

Besides my advisor, I would like to thank Dr. Stefano Mottola, Dr. Ekkehard Kührt and Dr. Gerhard Hahn for their encouragement, patience and readiness to help me in all the time of research and writing of this thesis and for the countless stimulating discussions we had.

Last, but certainly not least, I want to thank family and friends for being there and their motivation. Particularly, I would like to thank my parents Elke and Uwe and my brother Christoph.

# Contents

# CHAPTER 1

# Introduction

## 1.1 Minor planets in the solar system

### 1.1.1 Asteroids

Around 1600 Johannes Kepler postulated a small planet that may circle around the Sun between the orbits of Mars and Jupiter. Since then astronomers were searching for such an object. They got encouraged by the discovery of the Titus-Bode law in 1715, which, despite of being a curious coincidence only, also predicts an object at that location. After not being successful for almost 200 years of searching, during the 2nd European astronomical Congress in 1800, an organization called *Himmelspolizey* was founded in order to coordinate search for the planet among the available observatories. By the beginning of the following year, Guiseppe Piazzi, although not being member of the organization, indeed found a celestial body located within the predicted region. Bad weather conditions and illness forced Piazzi to stop observations in February 1801. Fortunately, combined efforts of Carl Friedrich Gauss, who developed a method of orbit determination based on three observations only and coordinated search around the predicted position by Himmelspolizey lead to rediscovering the object in late 1801. After the discovery was confirmed and the orbit constrained the body was categorized as a planet and named Ceres (⚳). However, it

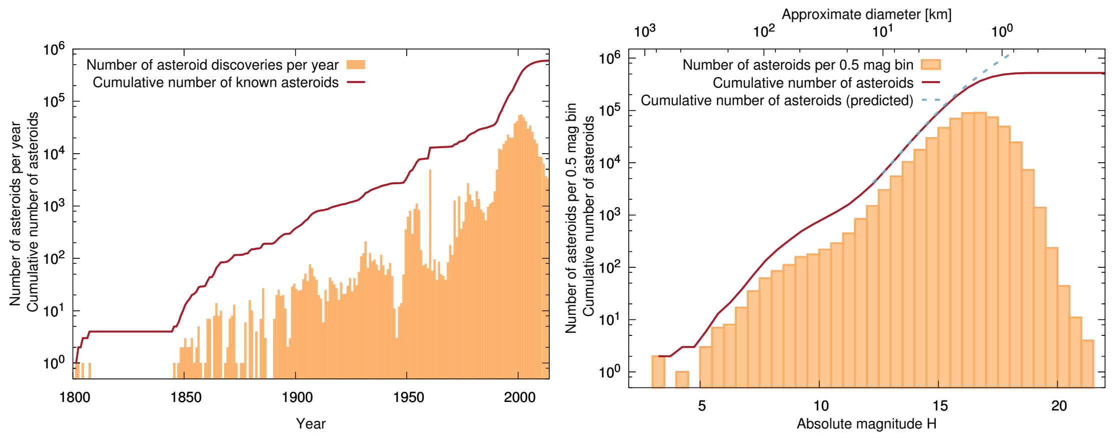

**Figure 1.1:** Left panel: Asteroid discoveries per year since 1800 and total number of known asteroids. Right panel: Size distribution of known asteroids in the main asteroid belt (see Figure 1.2) and predicted number of objects. (MPC data as on August 2017, numbered and multi-opposition objects only)

was soon realized that Ceres was not the only object located between Mars and Jupiter. During the following 50 years the number of discovered objects dramatically increased. Thus, it was decided to introduce a new class of solar system objects named *asteroids* or *minor planets* rather than continuously increasing the number of planets. It was further decided to assign numbers to asteroids instead of symbols, as for the planets. So Ceres (⚳) became (1) Ceres, the first asteroid to be discovered. Less than a century after the discovery of Ceres, in 1890, about 300 asteroids had been found. Due to growing numbers of telescopes available and ever improving observation equipment the trend of growing number of discoveries has been continuing until today (see left panel of Figure 1.1).

Since 1947 the observational data is processed and published by the Minor Planet Center (MPC) located in the United States and as of August 2017, there are 602.013 asteroids which have been observed over multiple apparitions (multi-opposition objects). Looking at the number of asteroid discoveries per year it seems like the rate of discoveries is currently dropping which would imply that now almost all existing asteroids have been found. The reason for this is that the plot contains only those asteroids for which a robust orbit is known, which are the ones observed over more than one apparition. Since obtaining observations from multiple apparitions may take several years, not all objects actually discovered are included in the graph. Another interesting feature can be seen in the right panel of Figure 1.1. The smaller the asteroids, the larger they are in numbers. Naturally, the biggest ones are the first to be found, as they are the easiest to spot. Indeed (1) Ceres, measuring 950 km in diameter, is the largest asteroid between Mars and Jupiter. The smaller, darker or more distant the objects are, the harder it is to detect them. Even

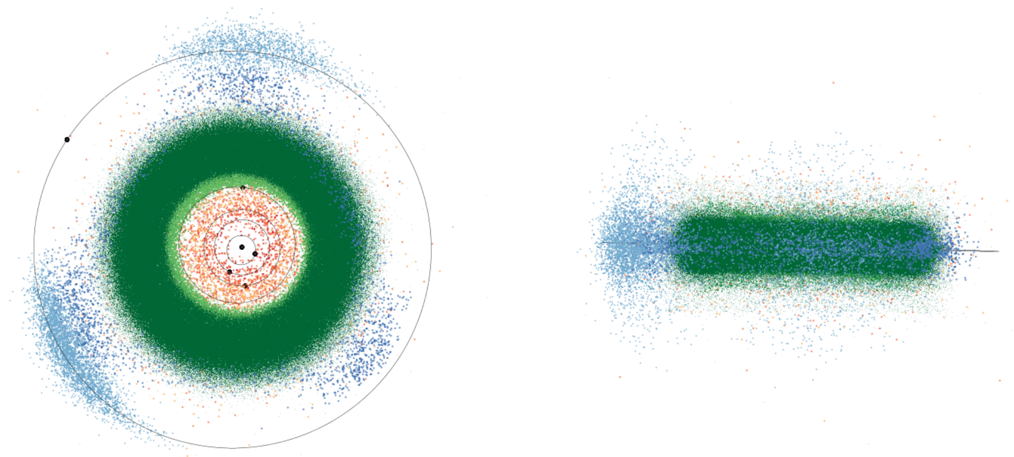

**Figure 1.2:** Snapshot of the inner solar system (left hand panel: top view; right hand panel: edge on view). Sun and Planet positions from Mercury to Jupiter as well as their orbits are shown in black. Each colored dot represents the position of an asteroid. As of August 2017 there are 602.013 numbered and multi-opposition asteroids in the inner solar system.

though not discovered yet, estimations on the number of small asteroids can be made, for example by estimating the performance of asteroid surveys [62] (dashed line in the right panel of Figure 1.1). Comparing the total number of known asteroids to the number of objects predicted, it can be seen that for the main asteroid belt (see Figure 1.2) most objects larger than about 3 km in diameter are known so far but many more smaller ones are to be discovered. It will take continuous observations and ever improving observing equipment in order to fill up the rear end of the graphs in Figure 1.1.

While more and more asteroids were discovered, it turned out not all of them are residing between Mars and Jupiter. Figure 1.2 shows the positions of all numbered and multi-opposition asteroids as of August 2017 in the inner solar system in a heliocentric reference frame. The densely populated region close to the ecliptic between Mars and Jupiter is called the main asteroid belt (green). However, there are other clusters of objects close to the orbit of Jupiter (blue) and in the very inner solar system (orange). When representing the asteroids in terms of their orbital elements rather than their current location the different groups are even easier to distinguish. Figure 1.3 again shows all numbered and multi-opposition asteroids in the solar system, but this time in terms of their orbital parameters semi-major axis, eccentricity and inclination.

According to their orbital parameters different dynamical families of asteroids can be identified: Asteroids of the Atira family move on orbits entirely inside the orbit of Earth. The family is named after (163693) Atira, the first asteroid discovered in this region in 2004. As of August 2017, there are only 18 multi-opposition Atira-asteroids known. Being always closer to Sun than Earth makes them very hard to detect. From Earth they can be observed only right after sunset and just before sunrise. Moving a bit further out there is the Aten family named after (2062) Aten discovered in 1976. Aten family asteroids have aphelion distances greater than 0.983 Astronomical Units (au), Earth's perihelion distance and semi-major axes greater than 1.0 au which means they are crossing the orbit of Earth. Another group of Earth-crossing asteroids is the Apollo family, named after (1862) Apollo discovered in 1932. Apollo type asteroids move on orbits having a semi-major axis greater than 1.0 au and a perihelion distance smaller than 1.017 au, the aphelion distance of Earth. The Amor family, named after (1221) Amor, discovered just before (1862) Apollo in March 1932, is a group of asteroids located close to but not crossing Earth's orbit. Amor family asteroids have semi-major axes greater than 1.0 and perihelion distances between 1.017 and 1.3 au. Being so close to Earth, Atiras, Atens, Amors and Apollos together make up the group of Near Earth Asteroids (NEAs). Most of the NEAs are orbiting the sun on chaotic orbits which means that their orbital elements are constantly changing and thus are scattered over a large volume in phase space.

The region between the orbits of Mars and Jupiter shows the highest density of asteroids in the inner solar system. The main asteroid belt fills up the largest part of that region. Next to the main belt there are two highly inclined groups, the Hungaria and Phocaea family. A bit further out there is the Hilda Family populating a narrow region in semi-major axis. Hilda family Asteroids are in a 3:2 mean motion resonance with Jupiter which means that while they do three revolutions around Sun, Jupiter revolves twice. As can be seen in Figure 1.2 their positions form a triangle just inside the orbit of Jupiter. Jupiter itself shares its orbit with another family of asteroids called the Jupiter Trojans. Jupiter Trojans

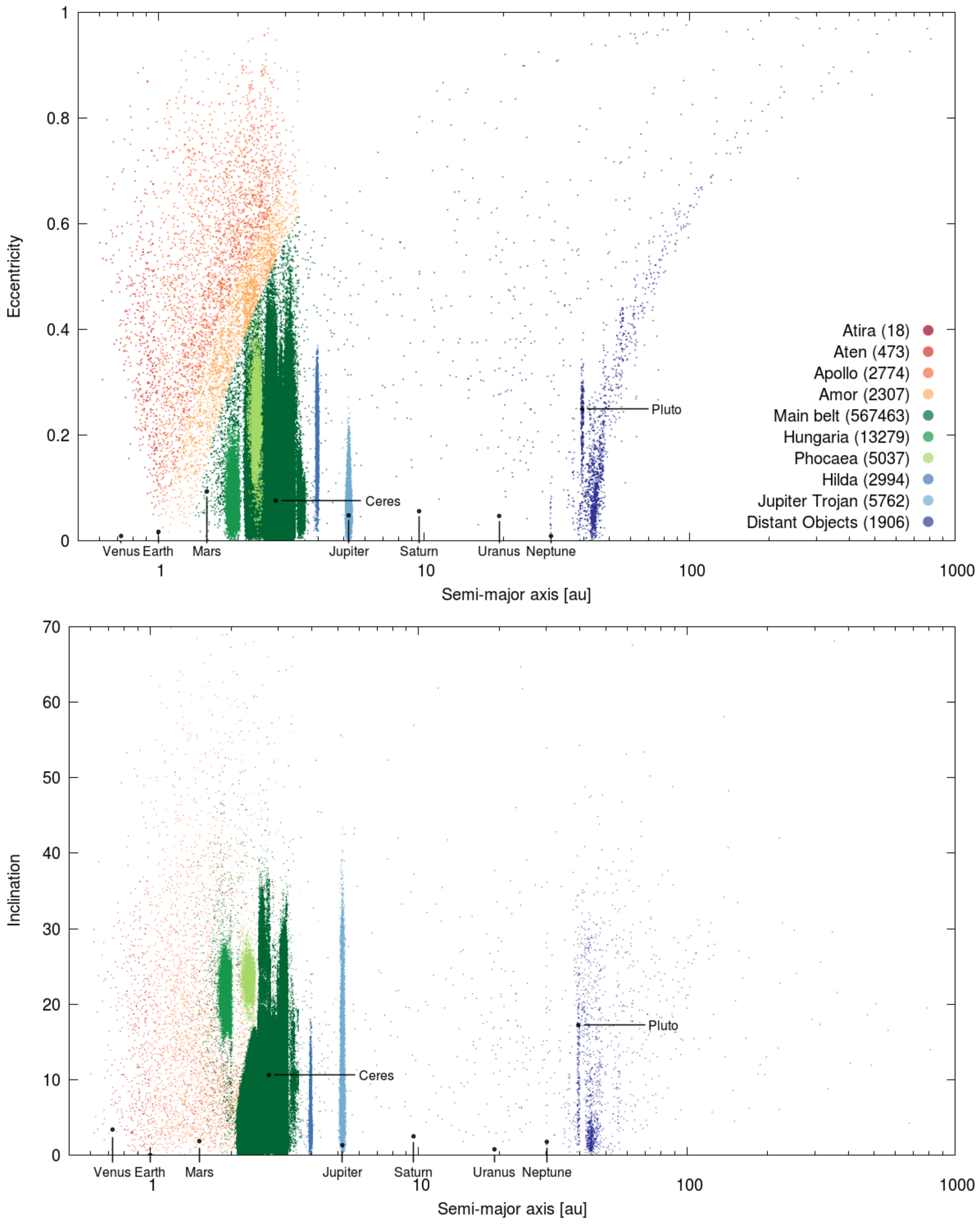

**Figure 1.3:** Orbital distribution of numbered and multi-opposition asteroids as of August 2017 colored by family. To concentrate on the most populated regions only, 2 objects having a semi-major axis greater than 1000 and 36 orbits with inclinations greater than 70° are omitted.

have about the same orbital period as Jupiter and form two distinct clouds on the orbit of Jupiter about 60° ahead and behind the position of the planet. A detailed description of this family can be found in chapter 6.

Beyond the orbit of Jupiter, in the outer solar system, the number of known objects dramatically decreases. Only a few asteroids on orbits between the gas and ice giants are known, but more have been found beyond the orbit of Neptune, Pluto being one of them. Discovered in 1930, Pluto was listed as the 9th planet of the solar system until very recently. Since its discovery more and more objects have been found on similar orbits as Pluto. As some of them are comparable in size and one, Eris, is even greater in mass, Pluto recently suffered from a similar fate as Ceres. In 2006, again a new category of solar system objects named *dwarf planets* was defined and Pluto got assigned to it. However, in the same turn Pluto lost its status as a planet, Ceres was promoted a dwarf planet. More about the orbit of Pluto can be found in Section 5.1. Objects located in the region ranging from the orbit of Neptune up to abut 1000 au are also referred to as Kuiper belt objects (KBOs) or transneptunian objects (TNOs). Although not many objects in this region are known so far, many more are expected to be discovered in the future [93].

The X axis in figure 1.3 ends at the outer edge of the Kuiper belt. However, even more objects are expected to orbit Sun even beyond the TNOs. This theoretical population is called the Oort cloud and may reach as far as Sun's gravitational dominance. Kuiper belt and Oort cloud are believed to be the reservoir for the comets we observe in the inner solar system [81].

### 1.1.2 Comets

Comets are known to exist much longer than asteroids. Once they reach the inner solar system they become active and develop their characteristic tail (or coma), they can become very easy to spot. For the brightest objects no telescope is needed to observe them and some of them become visible even during daytime. Their appearance is documented in history over thousands of years. However, the awareness of them being minor planets of the solar system came much later. One of the first thinking of an explanation for the appearance of comets was Aristotle in 350 B.C.E. In his treatise *Meteorology* he concluded that comets and shooting stars consist of gas, leaving the surface of Earth igniting itself below Moon. While in the case of shooting stars the gas is escaping fast, comets are made of gas leaving Earth more slowly. Later, in the medieval times, comets where seen as a precursor for all kinds of natural or civil disasters like starvation, epidemics or war. A very important step towards understanding the appearance of comets was made by Tycho Brahe in 1577. He concluded that if comets would be located between Earth and moon, it should be possible to measure a parallax effect when observing them from different places on Earth. As he was not able to measure the effect within the accuracy of the instruments he used, he postulated that comets must be at least four times further away from Earth than the moon. Then, there was Isaac Newton, who proved in his *Principia Mathematica* that comets follow the inverse square law of universal gravitation thus must move on orbits shaped like one of the conic sections. He demonstrated this by fitting the orbit of the comet of 1680 to a parabola. 25 Years later, in 1705, Edmond Halley compared the orbits of 23 cometary apparitions computed by the method of Newton and found that three of them had very similar orbital elements. Being able to explain the small differences in their orbital elements by gravitational perturbations of Jupiter and Saturn, Halley concluded that it is very likely the three apparitions belong to the same comet and predicted the comet would return in 1758, which it indeed did. Since then the comet was known as Halley's comet.

After being identified as minor bodies orbiting Sun just like planets or asteroids, the next question raised: Why do comets look so much different than asteroids? In 1835, Friedrich Wilhem Bessel observed Halley's comet as it returned to the inner solar system. Monitoring the change in appearance, Bessel concluded that the coma must come from volatile material which is vaporized by the heat of sun as the comet comes closer. Later during the 19th century, first comet spectra were observed by Giovanni Battista Donati and William Huggins [103]. While Donati took the first spectrum of a comet, Huggins was able to identify the emission lines with $C_2$, CH, CN and $C_3$, substantiating Bessels assumptions. Since then more than two dozen different molecules have been identified among cometary volatiles [6].

As of August 2017, the Minor Planet Center lists 347 numbered periodic comets, 1P/Halley being the first. Compared to the number of known asteroids this may look like there exist only a few comets in our solar system. Also the total number of comets ever observed, 3841, supports this assumption. While asteroids move on rather circular orbits in the inner solar system, comets show more eccentric orbits which means they spend most of the time far away from Sun making them harder to spot. However, studies suggest there may be many more comets residing in the outer solar system.

## 1.2 Dynamical evolution of minor planets

Looking to the orbital distribution of minor planets in the solar system (Figure 1.3) it appears that rather than being randomly distributed, there are voids and clumps of objects. Even though most of the minor bodies orbit Sun on regular orbits having modest eccentricity and inclination, the solar system is far from being dynamically stable. The NEAs for example have dynamical lifetimes of a few Myr only [7], which is very short compared to the age of the solar system of about 4.5 Gyr. As they are on planet crossing orbits, it is only a matter of time until they encounter one of the inner planets which can cause a dramatic change in their orbit and eventually leads to them getting thrown into Sun, being ejected from the solar system or even impacting a planet. Also the comets in the inner solar system are very short lived objects. As they constantly lose material they become smaller and smaller. Some of them eventually break apart during a perihelion passage while others suffer the same fate as NEAs as they are on planet crossing orbits too.

One might think that 4.5 Gyr should be enough time to rule out all the unstable orbits decreasing the amount of chaos in the solar system. However, asteroids and comets are not governed by gravity alone. Their orbits are also altered by collisions and other non-gravitational forces. Thus, the unstable regions are constantly resupplied with material which is then scattered across the whole solar system.

## 1.3 Motivation for this work

Understanding the dynamical processes in our solar system plays an important rule in understanding its evolution. Did the planets orbits change since they have formed? Why is the asteroid belt shaped like we observe it today? Where do the asteroids go once their orbits reach one of the unstable regions? What causes comets to leave the outer solar system and migrate inwards and how often does it happen? How many NEAs should we expect and how probable are impacts of comets and asteroids on Earth?

All these questions can not be answered by observations alone as the time scales for these dynamical processes are ranging from a few thousand to up to several billion years. Simulations which describe the formation and orbital evolution of planetary systems are needed to understand our solar system as we observe it today. These simulations must be capable to cover enormous time scales. Further, to statistically classify relatively rare phenomena like comets coming from the outer system or the production of Atira asteroid orbits, very large numbers of bodies have to be considered in the simulations. The computational hardware used to perform the simulations puts limits to the size of the experiments. To obtain results in reasonable computation time, the simulations have to be carried out on large-scaled computers like clusters or supercomputers or work has to be distributed over many desktop or workstation PCs.

In this work the growing computational power of Graphics Processing Units (GPUs) which recently became available in modern desktop PCs is exploited to significantly speed up the simulation methods on these computers. This allows comparatively cheap hardware performing experiments to simulate the orbital evolution of planets and minor planets involving up to $10^5$ bodies and covering simulation time spans of up to $10^9$ years — experiments which would need to be carried out on large-scale computers when using traditional hardware.

The work is organized as follows: In the following chapter the n-Body problem, which is the underlying problem when simulating gravitational interacting bodies, is introduced. Then, some basic methods for numerically solving the n-Body problem using computers are presented in order to introduce all aspects necessary to understand how these algorithms are working and what their benefits and limitations are. In chapter 3 the GPU as a device for general purpose computation is introduced and a brief review on the history of general purpose computing on GPUs is given. Chapter 4 is intended to describe the newly developed CPU and GPU implementation of the Wisdom and Holman Method (WHM) and the Regularized Mixed Variable Symplectic Method (RMVS) in cuSwift. Extensive tests of the integrators and the advantages over their original implementation are presented in chapter 5. Then, chapter 6 demonstrates a first application of cuSwift. It is studied if and how the non-gravitational Yarkovsky force influences the Jupiter Trojan asteroids. Finally, in chapter 7 some outlook on open points which may be worth tackling in the future is given.

# CHAPTER 2

# Solving the $n$-body problem

## 2.1 The $n$-body problem

The $n$-body problem is one of the fundamental problems in modern astronomy. It was first pointed out by Isaac Newton in his *Philosophiæ Naturalis Principia Mathematica*, published in 1687. The formulation of the problem is as follows: Considering a set of $n$ gravitationally interacting bodies with their masses, positions and velocities known for a certain time, how does the system develop with time? Although the formulation sounds so trivial it took much effort from numerous scientists before a suitable solution was found. The case for n=2 was formulated by Newton who wrote down the equations describing the force ($F$) between two gravitationally interacting bodies depending on their masses ($m_1, m_2$) and positions ($\mathbf{x}_1, \mathbf{x}_2$) in space:

$$F = G\frac{m_1 m_2}{r_{12}^2} \tag{2.1}$$

$G$ denotes the gravitational constant and $r_{12} = ||\mathbf{x}_1 - \mathbf{x}_2||$, the distance between the two masses. In a system of $n$ bodies, the gravitational force exerted on a single body i by all other bodies is

$$\mathbf{F}_i = -G \sum_{j=0;j\neq i}^{n-1} \frac{m_i m_j (\mathbf{x}_i - \mathbf{x}_j)}{r_{ij}^3} \tag{2.2}$$

Having in mind that $\mathbf{F} = m \cdot \mathbf{a}$ and $\mathbf{a} = \mathbf{x}''$, the equation of motion for a single body can be written as:

$$\mathbf{x}_i''(t) = -G \sum_{j=0;j\neq i}^{n-1} \frac{m_j (\mathbf{x}_i - \mathbf{x}_j)}{r_{ij}^3} \tag{2.3}$$

Now, the complexity of the problem becomes visible. In order to determine the position of a single body at a certain time one has to solve its equation of motion. As this equation depends on the positions of all other bodies in the system and as $\mathbf{x} \in R^3$, there is a system of $3n$ 2nd order differential equations to be solved.

Since the problem was formulated by Newton, much effort was put into finding an analytic solution. In 1885, there even was a prize established by King Oscar II of Sweden and Norway for anyone who finds an analytic solution to the problem. The prize should be awarded on King Oscars 60th birthday in January 1889. Unfortunately, no such solution to the problem has been found until that date. However, it was decided that Henri Poincaré should receive the prize for his remarkable contribution to the understanding of the equations of dynamics. It took another century until the problem was finally solved by Wang in 1991 [96]. Unfortunately, his solution has no practical value, as convergence is very slow and millions of terms would be need to be summed up to determine the bodies motion for insignificantly short time intervals and the propagation of round-off errors would make the solution useless [33].

Being unable to exactly solve the $n$-body problem did not prevent scientists working on the field of celestial mechanics for the last two centuries. Since it was formulated, many methods of solving the problem numerically have been developed and many interesting discoveries were made by applying them. And even nowadays, as the analytic solution has proven not to be useful, numerical methods are still the tool of choice when dealing with the $n$-body problem.

## 2.2 Numerical integration

In astrophysics, the $n$-body problem is a fundamental part of numerous fields of research. It is not only important for understanding the motion of planets around the Sun but also for explaining the gravitational interactions between stars or the gravitational behavior of bigger structures like galaxies or even groups of galaxies. There are many different methods to numerically solve the $n$-body problem, each having its benefits in the particular field it is used in. The methods applied in this work are highly specialized to model the long-term orbital evolution of planets and minor planets orbiting a dominant central mass like the Sun. In the following sections, these methods are derived step-by-step, starting from some very basic numerical integration methods.

The equation to be solved in order to follow the path of a single body in a system of n bodies is its equation of motion (See equation 2.3). It can be written as

$$\mathbf{x}_i''(t) = \mathbf{a}_i(t) \tag{2.4}$$

Further, the problem can be reduced to a system of 6 first-order ordinary differential equations for each body:

$$\begin{aligned}\mathbf{x}_i'(t) &= \mathbf{v}_i(t)\\ \mathbf{v}_i'(t) &= \mathbf{a}(\mathbf{x}(t))\end{aligned} \quad \text{or} \quad \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \begin{pmatrix} \mathbf{v} \\ \mathbf{a}(\mathbf{x}) \end{pmatrix} \quad \text{where} \quad \mathbf{y} = \begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix} \tag{2.5}$$

More generally, the system in 2.5 can be expressed as a Hamiltonian $H(\mathbf{q}, \mathbf{p})$ which is a function of some generalized coordinates $\mathbf{q}$ and momentum $\mathbf{p}$. The Hamiltonian equations describe how the system develops with time

$$\begin{aligned}\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} &= \frac{\partial H}{\partial \mathbf{p}}\\ \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} &= -\frac{\partial H}{\partial \mathbf{q}}\end{aligned} \quad \text{or} \quad \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \begin{pmatrix} \frac{\partial H}{\partial \mathbf{p}} \\ -\frac{\partial H}{\partial \mathbf{q}} \end{pmatrix} \quad \text{where} \quad \mathbf{y} = \begin{pmatrix} \mathbf{q} \\ \mathbf{p} \end{pmatrix} \tag{2.6}$$

The Hamiltonian for the $n$-body problem is

$$H(\mathbf{q},\mathbf{p}) = \sum_{i=0}^{n-1} \frac{{p_i}^2}{2m_i} - \sum_{i<j} \frac{Gm_im_j}{q_{ij}} \tag{2.7}$$

where $p_i = ||\mathbf{p}_i||$, $q_{ij} = ||\mathbf{q}_i - \mathbf{q}_j||$ and $\mathbf{p}$ and $\mathbf{q}$ are expressed in a barycentric referential frame.

### 2.2.1 Explicit methods

The most basic way of numerically solving ordinary differential equations is the explicit Euler method, developed by Leonhard Euler in 1768. Starting from a first-order differential equation and an initial condition

$$\begin{aligned} x'(t) &= f(t, x(t)), \\ x(t_0) &= x_0, \end{aligned} \tag{2.8}$$

the Euler method makes use of the Taylor Series expansion to numerically approximate the derivative. The Taylor Series expansion states that if $x : \mathbb{R} \to \mathbb{R}$ is a function of one variable (t) having $k$ continuous derivatives, the value for $x$ at a certain point $t + h$ can be expressed as

$$x(t+h) = x(t) \; + \; hx'(t) \; + \; \frac{1}{2}h^2x''(t) \; + \; \frac{1}{3!}h^3x'''(t) \; + \; \cdots \; + \; \frac{1}{k!}h^kx^{(k)}(\tau) \tag{2.9}$$

where $t < \tau < t + h$. The Euler method truncates the Taylor Series after the first term and ignores the quadratic and higher order terms which yields the finite difference formula for the derivative

$$x'(t) \approx \frac{x(t+h) - x(t)}{h}. \tag{2.10}$$

Starting from the initial condition $x(t_0) = x_0$, the integral is then iteratively developed as

$$x_{n+1} = x_n + hf(t_n, x_n). \tag{2.11}$$

Because the derivative is approximated by truncating the Taylor Series, at each single step developing the equation from $x_n$ to $x_{n+1}$ an error is made. This error is called the local truncation error (LTE) and is defined as the difference between the true value $x(t_n)$ and the approximated value $x_n$. For the Euler method, this error is

$$LTE_{n+1} = x(t_{n+1} + h) - x_n = \frac{1}{2}h^2x''(\tau) \propto \mathcal{O}(h^2). \tag{2.12}$$

The absolute value for the LTE may not be known as the derivatives of $x(t)$ may not be known. However, it is still useful as it is related to order $p$ of the numerical integration method via the maximum value of the LTE over the integration interval $I = [t_0,...,t_k]$ by

$$\max_{n=1}^{k} |LTE_n| = \mathcal{O}(h^{p+1}). \tag{2.13}$$

For the Euler method, the LTE is proportional to $h^2$ which means that it is of first order.

To demonstrate how the Euler method works, a very simple case of a 2-body problem is considered. A system of physical units is chosen in which the gravitational constant and total mass of the system are both unity. Further, instead of the positions of the two bodies, their distance in space is considered. Applying this definition, equation 2.3 simply becomes

$$\mathbf{r}''(t) = -\frac{\mathbf{r}}{r^3} \tag{2.14}$$

where $\mathbf{r}$ is the vector connecting the two bodies and $r = ||\mathbf{r}||$ the distance between them. Solving the system of two first-order differential equations of the $n$-body problem (equation 2.5) yields

$$\begin{aligned} \mathbf{r}_{n+1} &= \mathbf{r}_n + h\mathbf{v}_n \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + h\mathbf{a}(\mathbf{r}_n) \end{aligned} \tag{2.15}$$

Finally the initial conditions for $\mathbf{r}$ and $\mathbf{v}$ are chosen as

$$\mathbf{r}_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad and \quad \mathbf{v}_0 = \begin{pmatrix} 0 \\ 0.75 \\ 0 \end{pmatrix} \tag{2.16}$$

Choosing $||\mathbf{v}_0|| < 1$, lower than the circular orbit velocity $v_{circ} = \sqrt{\frac{G(M_1+M_2)}{r}} = 1$, the two bodies should move on an elliptical orbit around each other. Further, the choice of $\mathbf{r}_0$ and $\mathbf{v}_0$ results in the two bodies being at apocenter (the point in orbit with the largest distance between the bodies) by the start of integration. Figure 2.1 shows the exact solution of the system and how it develops under the Euler method for three orbital periods using different values for the time step. As can be seen, for all different time steps, in the case of the Euler method, the bodies do not move on closed elliptic orbits as they are supposed to according to the initial conditions. Choosing a time step of $10^{-2}$ orbital Periods $P$ leads to the bodies spiraling away from each other such that they break orbit after just half a revolution. Reducing the time step by a factor of ten increases the accuracy but still the two bodies spiral away from each other. After reducing the time step by another magnitude, even though hardly visible in the plot, the bodies still do not move exactly as expected. This clearly demonstrates the disadvantage of the Euler method. In order to get reliable results the time step has to be chosen very small which increases computation time as more steps have to be performed to cover the same period of time.

In contrast to the 2-body problem used here for demonstration, in general, the $n$-body problem can not be solved analytically. Thus, there is no way to determine the exact value for the error of the body's position and velocity because their true values for $t \neq t_0$ are

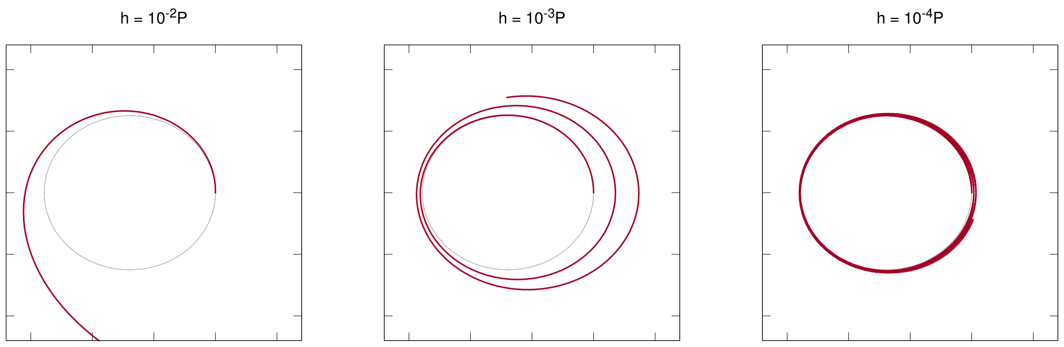

**Figure 2.1:** Relative positions of the bodies in a 2-body system integrated with the explicit Euler method using different values for the time step (red) compared to the exact solution (gray). Decreasing the time step increases accuracy at the expenses of the number of steps that have to be computed.

unknown. However, there is another quantity which can be examined: the total energy of the system. It must remain constant over time no matter in which direction or how fast the bodies move. This means that the change of total energy can be used as a metric for the error of the integration method which is used. The total energy of the $n$-body system is the sum of potential and kinetic energy of all bodies. For the 2-body problem, the potential and kinetic energies are

$$\begin{aligned} E_{pot} &= -\frac{M_1 M_2}{r} \\ E_{kin} &= \frac{1}{2}\frac{M_1 M_2}{M_1 + M_2} v^2 \end{aligned} \tag{2.17}$$

By introducing the reduced mass $\mu$ as

$$\mu = \frac{M_1 M_2}{M_1 + M_2} \tag{2.18}$$

the potential and kinetic energies become

$$\begin{aligned} E_{pot} &= -\frac{\mu(M_1 + M_2)}{r} \\ E_{kin} &= \frac{1}{2}\mu||\mathbf{v}||^2 \end{aligned} \tag{2.19}$$

Finally, as for the 2-body system which is used for demonstration, the sum of masses is defined as unity, both equations have the same dependence on mass and can be further simplified to

$$\begin{aligned} E_{pot} &= -\frac{1}{r} \\ E_{kin} &= \frac{1}{2}||\mathbf{v}||^2 \end{aligned} \tag{2.20}$$

Figure 2.2 shows how the relative error in total energy for the 2-body system develops for 100 steps using different values for the time step. The rapid nonlinear increase of the error in the left panel can be explained by Kepler's 2nd law: At pericenter ($t = 0.5$), the body's velocities and the curvature of their orbital ellipse are at maximum. As the body's positions are advanced in a linear manner and thus do not follow the curvature of the ellipse the truncation error made at each time step grows as the bodies are approaching each other and reaches its maximum at pericenter. For large time steps these nonlinear effects at pericenter are dominating. The middle and left panel do not show this behavior because no pericenter passage is captured as the time span covered with 100 steps decreases when decreasing the time step. When comparing the middle with the left panel of Figure 2.2 the first order character of Euler's method becomes visible. The error in energy decreases by about a factor of 100 when reducing the time step by a factor of 10.

As can be seen in Figures 2.1 and 2.2, reducing the time step results in a better approximation of the real orbit and in the energy error to decrease. However, as will be shown in section 2.3.4, setting the step to arbitrarily small values does not necessarily increase accuracy. More advanced, higher order integration methods like Runge-Kutta would yield better results even for relatively large time steps. But no matter how small the local truncation error is, the global error will grow with time making these integration methods unsuitable for long-term integration.

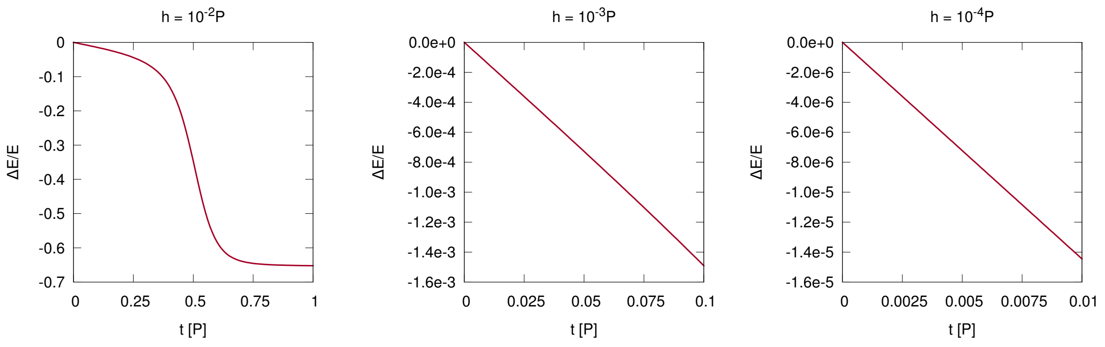

**Figure 2.2:** Relative energy error of the 2-body system integrated with the explicit Euler method using different values for the time step. The rapid incerase of the error in the left panel can be explained by nonlinear effects around pericenter (see text). For smaller time steps the first-order character of the Euler method becomes visible.

### 2.2.2 Symplectic methods

Symplectic integration methods are the tool of choice for modeling the evolution of orbits over large time scales [120], [90]. The striking advantage over traditional methods like the explicit Euler method or Runge-Kutta is that symplectic methods conserve certain geometric quantities of the system. To exactly explain how this works is far beyond the scope of this thesis. A detailed mathematical derivation of symplectic integration methods can be found in [49] and [50]. However, in this section the basic ideas behind these methods will be introduced and again, the $n$-body problem will be used for generic application.

A numerical integration method for solving first-order differential equations like the $n$-body is symplectic if it exactly solves the Hamiltonian equations and conserves volume in phase space. Mathematically, this means that for the vector $\mathbf{y} = \binom{\mathbf{q}}{\mathbf{p}}$ in phase space the time-dependent flow $\mathbf{y}(t, \mathbf{y}_0)$ governed by the Hamiltonian must conserve phase space volume. This is also known as Liouville's theorem. As a result of the theorem, the Jacobian $J(\mathbf{y}, \mathbf{y}_0) = \left|\frac{\partial \mathbf{y}}{\partial \mathbf{y}_0}\right|$ must be unity [90]. This condition can be adduced as a simple test to prove symplecticity for the $n$-body problem. Applied to the explicit Euler method (equations 2.15) it can be shown that

$$J(\mathbf{y}, \mathbf{y}_0) = \frac{\partial(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})}{\partial(\mathbf{x}_n, \mathbf{v}_n)} = \begin{vmatrix} 1 & h \\ h\mathbf{a}'(\mathbf{x}_n) & 1 \end{vmatrix} = 1 - h^2\mathbf{a}'(\mathbf{x}_n) \neq 1 \tag{2.21}$$

and thus, the method is not symplectic. This results in the error in total energy increases while integrating, as can be seen in figure 2.2.

Surprisingly, with a small modification only, symplecticity can be achieved for Euler's method [99]. The integration scheme for the symplectic Euler method is:

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n + h\mathbf{v}_n \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + h\mathbf{a}(\mathbf{x}_{n+1}) \end{aligned} \tag{2.22}$$

Note that the only difference to the original Euler method is that for integrating the velocity ($\mathbf{v}_{n+1}$), the new positions ($\mathbf{x}_{n+1}$) which were just calculated are used rather than the positions from the beginning of the current integration step ($\mathbf{x}_n$). Working out the Jacobian $J(\mathbf{y}, \mathbf{y}_0)$ yields

$$J(\mathbf{y}, \mathbf{y}_0) = \frac{\partial(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})}{\partial(\mathbf{x}_n, \mathbf{v}_n)} = \begin{vmatrix} 1 & h \\ h\mathbf{a}'(\mathbf{x}_{n+1}) & 1 + h^2\mathbf{a}'(\mathbf{x}_{n+1}) \end{vmatrix} = 1 \tag{2.23}$$

which proves the method conserves phase space volume and thus is indeed symplectic. Figure 2.3 shows how the 2-body system described in Section 2.2.1 develops with time when integrated with the symplectic Euler method. Even when choosing a large time step of $10^{-2}P$, which leads to the bodies breaking orbit after half a revolution under the explicit Euler method, the bodies are now moving on closed elliptic orbits. Nevertheless, an error in total energy is still present. But, in contrast to the explicit method it does not increase with time. This is because symplectic methods exactly solve the underlying Hamiltonian but are using an approximation for the integrals (i.e. assuming constant velocity over a full time step) which still produce local truncation errors. However, phase space conservation introduces constraints to the bodies positions and velocities resulting in a bounded total energy error.

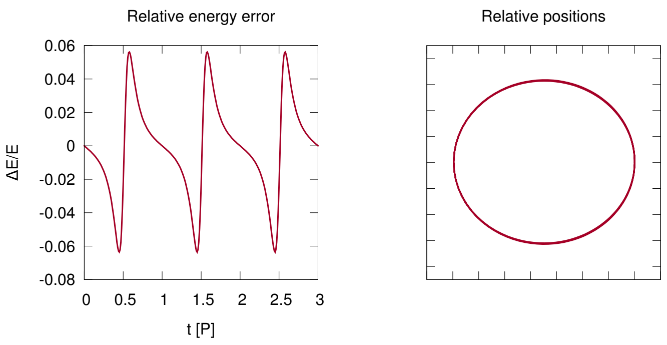

**Figure 2.3:** Relative energy error and relative positions of the 2-body system integrated with the symplectic Euler method for three orbits with a time step $h = 10^{-2}P$. The energy error is bounded due to phase space conservation.

### 2.2.3 A higher order integration scheme

As already mentioned, being the most basic numerical integration method, Euler's method is not suitable to be used for long-term orbital integration. The improved symplectic version of Euler's method leads to the bodies in the 2-body example moving on closed orbits but still the local truncation error is relatively large. Starting from the symplectic Euler method, a much better method with only very little additional computational costs can be constructed. Considering the scheme for the symplectic Euler method

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n + h\mathbf{v}_n \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + h\mathbf{a}(\mathbf{x}_{n+1}) \end{aligned} \tag{2.24}$$

and the equivalent scheme

$$\begin{aligned} \mathbf{v}_{n+1} &= \mathbf{v}_n + h\mathbf{a}(\mathbf{x}_n) \\ \mathbf{x}_{n+1} &= \mathbf{x}_n + h\mathbf{v}_{n+1} \end{aligned} \tag{2.25}$$

a 2nd order method can be constructed by composing half a time step of scheme 2.24 followed by half a step of scheme 2.25 which gives

$$\begin{aligned} \mathbf{x}_{n+\frac{1}{2}} &= \mathbf{x}_n &&+ \frac{h}{2}\mathbf{v}_n \\ \mathbf{v}_{n+1} &= \mathbf{v}_n &&+ h\mathbf{a}(\mathbf{x}_{n+\frac{1}{2}}) \\ \mathbf{x}_{n+1} &= \mathbf{x}_{n+\frac{1}{2}} &&+ \frac{h}{2}\mathbf{v}_{n+1} \end{aligned} \tag{2.26}$$

This scheme is referred to as the leapfrog integrator and is one of (if not the most) popular integration methods used in various fields. It was independently discovered by different scientists and thus is also called Newton-Störmer-Verlet method. A detailed historical overview about the method as well as a comprehensive discussion of their geometric properties is given in [49]. The first and last line in the integration scheme 2.26 are also referred to as *kick* steps while the step in the middle is often called *drift* step. Note that scheme 2.24 ends with a drift step while scheme 2.25 begins with a drift step so that when combining the two schemes, these two steps can be summarized as a single drift step over a full time step $h$. As both schemes for the Euler method are symplectic, their composition also is symplectic. Further, as can easily be seen for the new integration scheme, advancing the system by one time step $h$, and then applying another step for $-h$ immediately leads to the starting conditions again. That is, in contrast to the standard Euler method, the new integrator being time reversible, another very favorable property for integration methods.

Once again, the 2-body example will be used to demonstrate the advantage of the leapfrog method, especially in terms of long-term integration, over Euler's method or Runge-Kutta. This time, the bodies are followed over a time span as long as thousand orbital periods. Because the evaluation of the accelerations is computationally the most expensive part for the $n$-body problem (recall the $(n^2)$ complexity in equation 2.3), for better comparison, the time step for each method is chosen such that the number of acceleration calculations per orbit is the same rather than using equal time steps for each method. For the Runge-Kutta method the accelerations must be evaluated four times per time step. Thus, the step size for Runge-Kutta is set four times larger than for the other methods resulting in a similar computational cost for each run. The step sizes used for this example are 0.01 orbital periods for the leapfrog and the Euler methods and 0.04 orbital periods for Runge-Kutta.

Figure 2.4 shows how the error in total energy develops over the integration time. As expected, the explicit Euler method performs rather poorly. For the two symplectic methods the difference in order is clearly visible: Leapfrog beats the symplectic Euler method by about an order of magnitude. Runge-Kutta performs much better than any of the other methods for about hundred orbits but then gets beaten by leapfrog. This comparison clearly demonstrates the advantage of symplectic integration methods when used for long-term integration. For this example only thousand orbital periods are considered. Typical integration times for studying dynamical processes in the solar system are ranging from a few hundred thousand up to several billion orbital periods, making this comparison even more important.

There is however a major drawback of symplectic integrators one should be aware of. All the very favorable properties of symplectic integration methods like phase space volume conservation and being time reversible are only true assuming a fixed value for time step $h$. Changing the time step size is a powerful and and a common technique to reduce the error made during integration. For example the error occurring at pericenter can be reduced by reducing the time step whenever two bodies are approaching each other. For symplectic integrators however, it is in general not possible to change the size for the time step while integrating. The easiest way to see this is by imagining the time step being a function of the bodies' current positions and velocities $h(\mathbf{v_n}, \mathbf{v_n})$ and than realizing the condition for being symplectic $J(\mathbf{y}, \mathbf{y}_0) = 1$ is violated. There are methods to overcome this limitation (e.g. [76]) which are not discussed here.

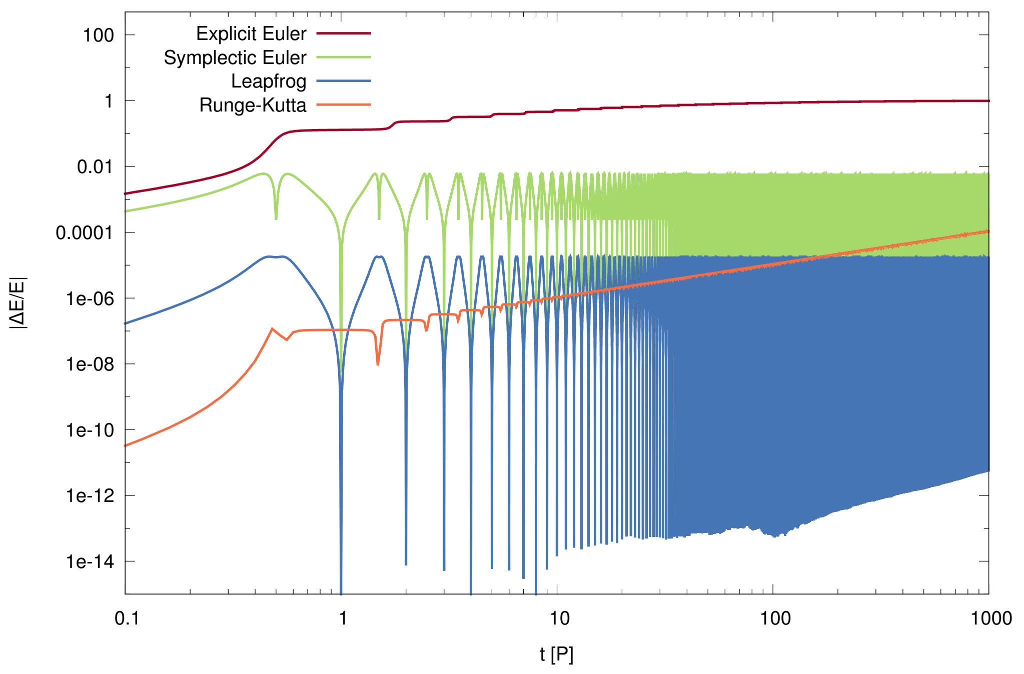

**Figure 2.4:** Comparison of the relative error in total energy for different integration methods. The Explicit Euler method performs worst. For the symplectic methods the relative error in energy is bounded and the difference in order of the Symplectic Euler and leapfrog is clearly visible. Runge-Kutta shows good results for about hundred orbits but then is beaten by leapfrog as, in contrast to the symplectic methods, the error for Runge-Kutta increases with time.

## 2.3 Tuning the method for solar system like $n$-body problems

Planetary systems like the solar system can be seen as a special case of the $n$-body problem. They consist of a star, a dominant central mass and the less massive planets which move on rather circular orbits around the star. This property can be exploited by numerical methods in order to increase accuracy. For planetary systems it is common to split the Hamiltonian into a part corresponding to the bodies' Keplerian motion around the center of mass and a part which describes the direct and indirect perturbation terms due to interactions with the other bodies [115].

$$H = H_{Kepler} + H_{interaction} \tag{2.27}$$

The advantage of this splitting is that the two resulting Hamiltonians can then be individually solved in an appropriate manner. Under the Keplerian Hamiltonian, the positions and velocities of the bodies are advanced on their Keplerian orbits around the center of mass as described by their orbital elements rather than assuming linear motion like the methods introduced before. This is accomplished by using Gauss' famous $f$ and $g$ functions [95]

$$\begin{aligned} \mathbf{x}_{t+h} &= f(t+h)\mathbf{x}_t + g(t+h)\mathbf{v}_t \\ \mathbf{v}_{t+h} &= f'(t+h)\mathbf{x}_t + g'(t+h)\mathbf{v}_t \end{aligned} \tag{2.28}$$

which provide an efficient way to propagate Keplerian orbits without having to calculate all of the actual orbital elements. Highly effective methods for solving the $f$ and $g$ functions analytically are given by Danby [27] and Prussing & Conway [95]. The interaction Hamiltonian is a function of the bodies' positions only and can also be integrated analytically.

### 2.3.1 Mixed Variable Symplectic method

Expressing the $n$-body Hamiltonian as a sum of a Keplerian and an interaction Hamiltonian introduces some additional complexity to the problem. As described by Wisdom & Holman [115], the motion of the bodies can be cast into Jacobi coordinates in order to derive a Keplerian Hamiltonian for the $n$-body problem. The first Jacobi coordinate $\hat{\mathbf{x}}_0$ is the center of mass of the system. The Jacobi coordinates for the other bodies are then the difference vectors between their barycentric positions $\mathbf{x}_i$ and the center of mass up to body $i$

$$\hat{\mathbf{x}}_i = \begin{cases} \frac{1}{M_{tot}} \sum\limits_{i=0}^{n-1} m_i \mathbf{x}_i & \text{for } i{=}0 \\ \mathbf{x}_i - \frac{1}{\eta_{i-1}} \sum\limits_{j=0}^{i-1} m_j \mathbf{x}_j & \text{for } 0{<}i{<}\text{n} \end{cases} \tag{2.29}$$

where $\eta_i = \sum_{j=0}^{i} m_j$, the sum of masses up to body $i$ and $M_{tot} = \eta_{n-1}$ the total mass of the system. Accordingly, the Jacobi momenta are given by

$$\hat{\mathbf{p}}_i = \begin{cases} \sum\limits_{j=0}^{n-1} \mathbf{p}_j & \text{for } i{=}0 \\ \frac{\eta_{i-1}}{\eta_i} \mathbf{p}_i - \frac{m_i}{\eta_i} \sum\limits_{j=0}^{i-1} \mathbf{p}_j & \text{for } 0{<}i{<}\text{n} \end{cases} \tag{2.30}$$

They can be calculated by $\hat{\mathbf{p}}_i = \hat{m}_i \hat{\mathbf{x}}'_i$ where

$$\hat{m}_i = \begin{cases} M_{tot} & \text{for } i{=}0 \\ \frac{\eta_{i-1} m_i}{\eta_i} & \text{for } 0{<}i{<}\text{n} \end{cases} \tag{2.31}$$

As the Jacobi coordinates describe the location of each body with respect to the center of mass of the bodies below it, they rely on the bodies to be ordered by increasing distance from the central mass.

Using Jacobi coordinates, the $n$-body Hamiltonian can be expressed in the desired form (equation 2.27) with the Keplerian Hamiltonian

$$H_{Kepler} = \sum_{i=1}^{n-1} \left( \frac{\hat{p}_i^2}{2\hat{m}_i} - \frac{Gm_i m_0}{\hat{r}_i} \right) \tag{2.32}$$

and the interaction Hamiltonian

$$H_{interaction} = \sum_{i=1}^{n-1} Gm_i m_0 \left( \frac{1}{\hat{r}_i} - \frac{1}{r_{i0}} \right) - \sum_{0<i<j} \frac{Gm_i m_j}{r_{ij}} \tag{2.33}$$

where $\hat{r}_i = ||\hat{\mathbf{x}}_i||$ and $\hat{p}_i = ||\hat{\mathbf{p}}_i||$. This transformation called a symplectic map is also known as Wisdom-Holman map.

The whole system is then advanced by applying the leapfrog integration scheme. For each time step $h$, the positions and velocities are advanced under $H_{Kepler}$ for $h/2$, then the perturbations are evaluated under $H_{interaction}$ for $h$ followed by another step of $H_{Kepler}$ for $h/2$. Since different types of coordinates are used under the two Hamiltonians, this algorithm is called Mixed Variable Symplectic method (MVS). Although still being a 2nd order integrator, MVS performs much better than the ordinary leapfrog scheme (Equations 2.26) making it to a standard tool which is widely applied for integrating planetary systems [19].

However, this symplectic map also introduces some unfavorable behavior. The method performs very well as long as the perturbations described by $H_{interaction}$ are much smaller than $H_{Kepler}$, which is true if the planets move on low eccentricity orbits that are well separated from each other. When choosing orbits causing close encounters between the planets, the assumption $H_{Kepler} >> H_{interaction}$ is violated during the encounter causing the error to increase. Further, using Jacobi coordinates, the method also does not allow the planets' orbits to cross each other (since these coordinates require distance-sorted planet indices). There are several methods to overcome these problems. Duncan et al. [35] described a variant of MVS which is able to resolve close encounters by a multiple time step method called Symplectic Massive Body Algorithm (SyMBA). Another method, suggested by Chambers [18], uses canonical heliocentric coordinates instead of Jacobi coordinates to allow planet crossing orbits and permits close encounters by using a conventional integrator to resolve the encounters. The integrators implemented in this work however, are meant to be applied on systems like our solar system where no close encounters between the planets occur and thus are based on the MVS method.

### 2.3.2 Integrating minor bodies of infinitesimal mass

The methods described in the last sections are suitable to treat the heavier objects in planetary systems. There is, however, another class of bodies which is of particular interest for this work which have been left out so far: all the asteroids and comets. As outlined in Section 1.1 these objects are very large in numbers, but on the other hand very small in mass. In fact, the combined mass of all asteroids in the main belt is only about 4% the mass of Earth's moon [60]. Being so small in mass compared to the Sun and the planets, the gravitational influence of the minor bodies on the planets as well as the particles' mutual interaction can be neglected. However, the particles feel the gravitational influence caused by the planets. For the $n$-body problem, this is a huge simplification as there are only the Sun and a few planets to be evaluated in the $\mathcal{O}(n^2)$ part and not the enormous number of asteroids and comets. When assigning those objects, further called *test particles*, a Jacobi index below those of the massive planets, the interaction Hamiltonian for a particle becomes [115]

$$H_{TestParticle} = -\sum_{i>1}^{n-1} \left( \frac{Gm_i}{r_{tp\,i}} - \frac{Gm_i \hat{\mathbf{x}}_{tp} \cdot \mathbf{x}_{i0}}{r_{i0}^3} \right) \tag{2.34}$$

where $\hat{\mathbf{x}}_{tp}$ is the particle's Jacobi coordinate, $r_{tp\,i} = ||\mathbf{x}_{tp} - \mathbf{x}_i||$, the distance between test particle and body $i$ and $\mathbf{x}_{i0} = \mathbf{x}_i - \mathbf{x}_0$, the vector from the central mass to body $i$. Note that $n$ in equation 2.34 corresponds to the number of planets in the system. Assigning the test particles a Jacobi index below the planets causes them to be advanced on an orbit around the central mass under the Keplerian Hamiltonian.

### 2.3.3 Encounters between minor bodies and planets

Asteroids and comets may not behave as *modestly* as the planets: many of them move on highly eccentric, parabolic or even hyperbolic orbits, which means that close encounters between particles and planets are very likely to occur. As already described, the error for the MVS method grows when two massive bodies approach each other. This is also true for the case when a mass-less particle approaches one of the massive planets because the force by the encountered planet on the particle becomes comparable to the force by the central body. So the assumption that the particle circles around the central mass on a Keplerian orbit does not hold anymore. Further, the minor bodies are very sensible to close encounters. Depending on the encounter distance and relative velocity, a particle's orbit may dramatically change during an encounter. In this situation, the advantage of the fairly large time step which can be used for symplectic integration methods leads to the disadvantage that the close encounters between particles and planets are very poorly resolved.

To overcome this problem, Levison & Duncan [65] introduced a technique of integrating the test particles permitting close encounters between them and the massive bodies called Regularized Mixed Variable Symplectic (RMVS) Method which has became very popular. The first step to resolve close encounters is to detect their occurrence. A good measure for that is the region around a planet where its gravitational influence exceeds the influence by the Sun. This region is also called the Hill sphere and its radius is defined as [65]

$$r_{hill} = a_p \left( \frac{1}{3} \frac{m_{pl}}{m_{sun} + m_{pl}} \right)^{1/3} \tag{2.35}$$

where $a_p$ denotes the semi-major axis of the planet and $m_{sun}$ and $m_{pl}$ the masses of the Sun and the planet. In RMVS, once a test particle penetrates a planet's Hill radius, the coordinate system for this particle is switched to a reference frame centered on the encountered planet. That is, the test particle is advanced on a Kepler orbit around the planet rather than around the Sun under the Keplerian Hamiltonian. As during the close encounter, the trajectory of the particles can be better described by a conic section around the planet rather than around the Sun, advancing the particle on a Kepler orbit around the planet causes the orbit of the test particle to be much better resolved. Further, to follow the trajectory of the particles more accurately, the time step for the encountering particle is reduced. In order to not loose the symplectic property for integrating the planets (see Section 2.2.3), the planets intermediate positions, which are needed to refine the time step for the encountering particle, are interpolated on their Kepler orbits between their positions at the beginning and at the end of the current time step. Furthermore, the region just outside the Hill radius of a planet, where the forces by the planet and the Sun are comparable in magnitude, also has to be considered. In RMVS, this is achieved by defining an intermediate encounter region around each planet which is 3.5 times the planets Hill radius. If a particle enters the intermediate encounter region, the time step for this particle is reduced by the method just explained. Taken all together, a close encounter in RMVS is resolved as follows: When, at the beginning of a time step, a test particle lies or is predicted to be at the end of the time step within the extended encounter region, the step size for the particle is reduced by a factor of ten and the planet positions required for each sub step are interpolated. If the particle further penetrates the planets Hill radius, the coordinate system is switched to a planetocentric reference frame and the time step is reduced by another factor of three.

It is important to note that close encounter handling has implications on the size of the time step. To make sure no encounter is missed, the step size must be chosen such that a particle can not completely move over a distance larger than 3.5 times a planet's Hill sphere within one time step. For simulations of the outer solar system, where usually only the heavy outer planets are included in the simulation, this is not a problem as the Hill radii of those planets are large and the relative velocities between the planets and the particles are small. Step sizes of about 40 days are sufficient for such simulations [65]. However, when including the inner planets of the solar system, high speed encounters are very likely to occur as for eccentric orbits the relative velocities in the inner solar system can be considerably high [47]. The fact that the inner planets are rather small in mass and thus have only small Hill radii makes things even more complicated. In order not to miss close encounters with the inner planets time step has to be set to values as small as several hours only [48].

### 2.3.4 Limitations due to machine precision

Using computers for numerical integration always limits the accuracy to be achieved. To understand the limitations of numerical methods introduced by precision limits of computers it is necessary to understand the floating-point arithmetic model. Most microprocessors comply with the IEEE Standard for Floating-Point Arithmetic (IEEE 754). According to this standard a floating point number is represented as $x = s \cdot m \cdot b^e$, where $s$ is the sign, consisting of one bit, $m$ is the mantissa, $b$ is the base and $e$ the exponent. The number of significant places of $m$ and $e$ are depending on the floating point format (see Table 2.1). Note that total number of bits appears to exceed the format width as there is one more bit for the sign which is not listed in the table. This is because in the IEEE 754 standard floating-point numbers are normalized which means the first bit of the mantissa is always one and thus does not have to be explicitly stored.

There are two important factors affecting the accuracy of numerical methods: the step size and the order of the method. The Euler method can be used to construct a simple example demonstrating the limitation of the time step size. Integrating the 2-body system introduced in the previous section with a step size of $h = 0.01$, at $t = 0.4$ the state vectors $\mathbf{r}$ and $\mathbf{v}$ are

$$\mathbf{r}(0.4) = \begin{pmatrix} 0.921807 \\ 0.292436 \\ 0 \end{pmatrix} \quad \textit{and} \quad \mathbf{v}(0.4) = \begin{pmatrix} -0.402286 \\ 0.689362 \\ 0 \end{pmatrix} \tag{2.36}$$

The integration scheme of the Euler method (Equations 2.15) consists of a multiplication and an addition each. Considering a floating-point model where $b = 10$, $m$ has 5 and $e$ has 1 decimal places, $r_x(t + \Delta t)$ for $t = 0.4$ becomes

$$r_x = 9.2181 \cdot 10^{-1} + (-4.0229) \cdot 10^{-1} \cdot 1.0000 \cdot 10^{-2} \tag{2.37}$$

As one of the addends is multiplied by the time step, the two addends are differing in absolute value by 2 orders of magnitude (depending on the value of $\Delta t$). This difference will reflect in the sum:

$$\begin{aligned} r_x &= \mathsf{9.2181} \cdot \mathsf{10}^{-1} + (-\mathsf{4.0229}) \cdot \mathsf{10}^{-3} \\ &= \mathsf{0.92181} \\ &\quad -\mathsf{0.0040229} \\ &= \mathsf{0.9177871} \\ &= \mathsf{9.1779} \cdot \mathsf{10}^{-1} \end{aligned}$$

As can be seen in the example, the last two digits of the second addend are lost during addition and the result is rounded accordingly. Thus, the relative error of the sum grows as the time step shrinks. At some point, the round-off error dominates and finally, using a time step of $h \leq 10^{-6}$ there would be no change in position at all. Figure 2.5 shows how the relative error for the total energy develops when decreasing the time step employing single-precision floating-point numbers. Choosing too small values for the time step causes

**Table 2.1:** IEEE 754 Floating-Point Format

| Format | Width [bits] | Mantissa [bits] | Exponent [bits] |
|---|---|---|---|
| Single | 32 | 24 | 8 |
| Single-Extended | 43 | 32 | 11 |
| Double | 64 | 53 | 11 |
| Double-Extended | 79 | 64 | 15 |

the total error to grow due to limited machine precision.

The issue of accuracy limits due to machine precision must be considered when selecting or implementing an integrator. A higher-order method increases accuracy and allows larger step sizes. However, for small step sizes the magnitudes in the higher-order terms of the Taylor series (see Equation 2.9) are rapidly decreasing so that the last terms may not influence the result at all and thus are only causing unnecessary computations. Consider, for example a 4th order integration method used on a computer having a word length of 32 bit in single-precision. Equation 2.39 shows the Taylor expansion for such an integration method.

$$x(t+h) = x(t) \;+\; f(t)(h) \;+\; \frac{f'(t)}{2}(h)^2 \;+\; \frac{f''(t)}{3!}(h)^3 \;+\; \frac{f'''(t)}{4!}(h)^4 \;+\; \mathcal{O}(t)(h)^5 \tag{2.38}$$

As 24 out of the 32 bits are available for the mantissa (see Table 2.1), the round-off error is in the order of $10^{-6}$. Assuming the values for $x(t)$ are in the order of $10^1$ and a time step in the order of $10^{-2}$, the magnitudes of the addends in Equation 2.38 become

$$||x(t+h)|| \approx 10 \;+\; 0.1 \;+\; 10^{-4} \;+\; 10^{-6} \;+\; 10^{-8} \tag{2.39}$$

Each term contributes fewer significant digits to the result. For the 2nd term, already two digits are lost due to round-off and the addends from the 4th term on do not contribute anything to the result at all, which means that they only introduce unnecessary computational cost. Thus, in this particular case a lower order integration scheme or, if high accuracy is required, a double-precision representation would be much more appropriate. Another important criterion for choosing the order of the integration method is the quality of the initial conditions. It would be a waste of computational resources to use a high order integrator on low quality data.

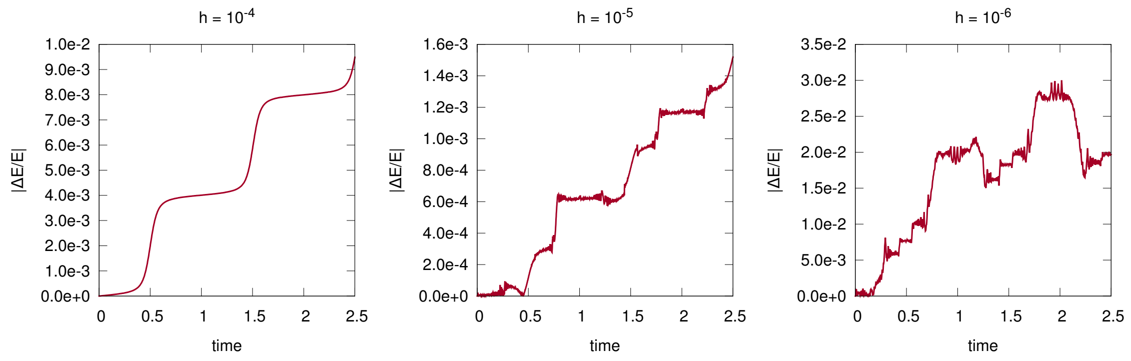

**Figure 2.5:** Relative energy error of the 2-body system integrated with the explicit Euler method using different values for the time step. Due to limited machine precision the error increases when choosing too small time steps.

## 2.4 The Yarkovsky Effect

The integration methods introduced in the previous sections are describing very well the motion of planets and minor planets in solar system like $n$-body problems caused by their mutual gravitational interaction. However, gravity is not the only force acting on these bodies in such systems. Around 1900, the Russian engineer Ivan Osipovich Yarkovsky proposed a tiny force, caused by diurnal heating and asymmetric re-radiation of energy received from the Sun which acts on rotating objects in the solar system. The discovery made by Yarkovsky was almost forgotten and the original publication is apparently lost. Luckily, Ernst Öpik who read the publication around 1909, later recalled the effect and was able to re-derive it in 1951 [91]. Since then, it was realized that the Yarkovsky effect plays an important role in the long-term evolution of asteroid orbits. Despite the force is very small, it can change the bodies' orbit over long time-scales. A good review about the history of the Yarkovsky effect is given by Hartman et al. [53]. In 2003, the first direct measurement of the Yarkovsky effect was made by Chesley et al. [21]. They used the Arecibo radar to observe near earth asteroid (6489) Golevka and found the object was displaced due to the Yarkovsky force by about 15 km over twelve years.

The strength of the Yarkovsky force depends on the physical characteristics of the body's surface, its rotational and orbital properties as well as its density and detailed shape [9]. However, as shapes are known for only very few asteroids [37] and because for irregular shapes calculating the Yarkovsky force becomes computationally very expensive [86], most methods for modelling the Yarkovsky effect for many bodies assume the objects to have a spherical shape.

Rather than giving a detailed mathematical description how the effect is modeled, this section is intended to give a brief overview on the basic principles of how the Yarkovsky effect works, what the driving parameters are and how they influence the effect. A very detailed physical description of the Yarkovsky effect can be found in the dissertation of Brož [13]. The Yarkovsky effect is a combination of two distinct components which are described in the following.

### Diurnal Component

On the body's day side, some of the sunlight is absorbed and warms up the surface. Depending on the thermal properties of the asteroid surface layer, the stored energy is then released later during the afternoon as the object continues spinning. The momentum carried away by the radiated energy causes a force pointing away from the asteroid's warmer side. Despite being very weak, the force causes the object's semi-major axis to systematically decrease or increase as it has a component pointing alongside the asteroid's orbit. To a lower degree the effect also affects orbital eccentricity.

The strength of the diurnal component and the direction of the semi-major axis drift are depending on several parameters. If the asteroid's rotational axis is perpendicular to the orbital plane, the effect is most efficient, while if the rotational axis lies in the orbital plane or the rotation period is the same as the orbital period, the effect vanishes as the force pointing alongside the asteroid's orbit disappears. Also for very fast rotating objects, the diurnal effect vanishes, as the amplitude of the diurnal temperature curve would be close to zero. The magnitude of the force further depends on the albedo of the object's

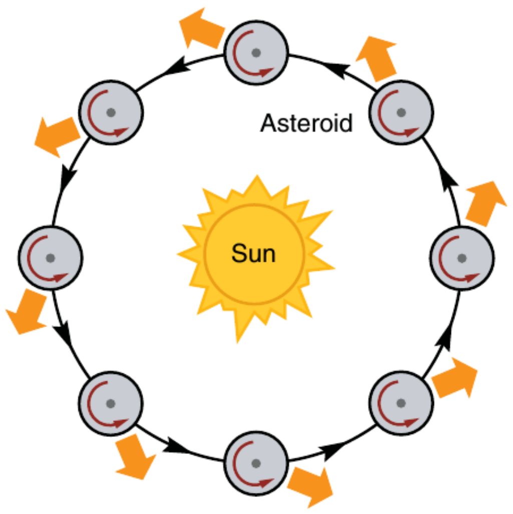

**Figure 2.6:** Diurnal component of the Yarkovsky effect on an asteroid with its spin axis normal to the orbital plane. Due to the thermal properties of the surface material most energy is radiated at the asteroid's afternoon side, causing a force acting alongside the orbit leading to a secular increase of the semi-major axis. (Image taken from Figure 1 in Bottke et al.[10])

surface which defines how much of the energy received from the Sun is absorbed by the asteroid. Another important parameter for the diurnal component of the Yarkovsky effect is the sensitivity to changes in the received energy of the material on the body's surface. This property is called thermal inertia and is defined as

$$\varGamma = \sqrt{k\rho C} \tag{2.40}$$

where $k$ is the thermal conductivity, $\rho$ the density and $C$ the specific heat capacity. The parameter $\varTheta_\omega$, an adimensional quantity, which describes the time lag $t_d$ between maximum absorption and maximum emission is defined as [107]

$$\varTheta_\omega = \frac{t_d}{P_r} = \frac{\varGamma\sqrt{\omega}}{\sigma\varepsilon T_\star^3} \tag{2.41}$$

where $P_r$ is the rotational period, $\omega$ the angular rotation velocity, $\sigma$ the Stefan-Boltzmann constant, $\varepsilon$ the thermal emissivity and $T_\star$ denotes the effective subsolar temperature of an object at a distance from the Sun corresponding to the semi-major axis of its orbit. If the surface material would have zero thermal inertia, the energy would be immediately re-radiated and the diurnal effect would vanish as $\varTheta_\omega = 0$. However, because the material in general does have thermal inertia, re-radiation occurs with a certain delay $t_d$, which results in most of the energy being emitted after noon, causing the component of the force pointing alongside the body's orbit.

The direction of the semi-major axis drift depends on the object's sense of rotation. On a prograde rotator, which means the sense of rotation is the same as the objects orbital direction, the direction of the Yarkovsky force is orientated such that the asteroid is accelerated and leads to a secular increase of the semi-major (see Figure 2.6). For a retrograde rotator, the semi-major axis would decrease as the alongside component of the force points against the body's orbital motion and thus acts like a constant brake. The strength of the diurnal Yarkovsky effect is also influenced by the body's size. On bigger objects like very large asteroids or planets, which have a small area-to-mass ratio, the force is so weak that it can be neglected. Also for very small objects the strength of the effect is very small as the diurnal thermal wave completely penetrates the body, which results in the heat being more evenly distributed through the whole asteroid, causing smaller temperature differences across the body's surface.

### Seasonal Component

The second component of the Yarkovsky effect is caused by seasonal heating of the asteroid's different hemispheres during a revolution around the Sun; hence it is called seasonal component. The mechanism is illustrated in Figure 2.7 for a body on a circular orbit around the Sun with its rotational axis lying in the orbital plane. At point A, the northern hemisphere receives the greatest amount of sunlight, but due to the asteroid's thermal inertia, the northern hemisphere re-radiates most energy at point B. This process repeats for the southern hemisphere at points C and D. In contrast to the diurnal component, the thermal parameter $\Theta_n$ for the seasonal component of the Yarkovsky effect depends on the orbital period $P_o$ rather than the rotational period. It is defined as [109]:

$$\Theta_n = \frac{t_s}{P_o} = \frac{\Gamma\sqrt{n}}{\sigma\varepsilon T_\star^3} \tag{2.42}$$

where $t_s$ is the time lag between maximum absorption and maximum emission for the seasonal component and $n$ the mean orbital motion.

Independent from the body's sense of rotation, the seasonal component always acts like a drag and leads to orbital decay. It also tends to circularize the orbit for small orbital eccentricities [9]. In contrast to the diurnal component, the seasonal effect vanishes if the rotational axis is perpendicular to the orbital plane and is at maximum if the rotational axis lies in the orbital plane.

While the diurnal component of the Yarkovsky effect dominates the semi-major axis drift for regolith-covered, stone-like objects, the seasonal component is more important for metal-rich objects [9]. This is mainly because of the different thermal conductivity of these types of asteroids which influences the penetration depth $l_s$ of the seasonal thermal wave [109]:

$$l_s =\simeq \sqrt{\frac{k}{\rho\omega C}} \tag{2.43}$$

where $\omega$ corresponds to the frequency of seasonal heating. Regolith-covered bodies have a low thermal conductivity which results in the penetration depth of the seasonal thermal wave being small, while for metal-rich objects, due to the much higher thermal conductivity, the seasonal thermal wave reaches much deeper, which results in more energy being stored

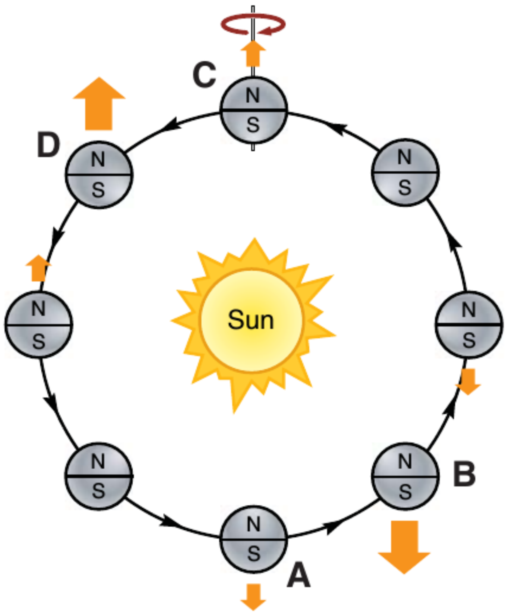

**Figure 2.7:** Seasonal component of the Yarkovsky effect on an asteroid with its spin axis lying in the orbital plane. The Northern and southern hemispheres are receiving most energy from the sun at points A and C, respectively. Due to the thermal properties of the asteroid, the hemispheres re-radiate most energy at points B and D. The seasonal component always acts like a drag and leads to orbital decay (Image taken from Figure 1 in Bottke et al.[10])

over the period of seasonal heating. Thermal conductivity can vary over several orders of magnitudes for the different types of objects [41].

The direction of semi-major axis drift depends on which component is dominant. Asteroids dominated by the diurnal effect would undergo a random walk in semi-major axis due to collisions which may change spin rate and direction from time to time, while a dominant seasonal effect would always lead to orbital decay. The strength of the effect also depends on the body's proximity to the Sun. More distant asteroids receive fewer sunlight and are thus much less influenced by the Yarkovsky effect.

The Yarkovsky effect provides an explanation for several phenomena observed in the solar system. It is the dominant mechanism which drives asteroids from stable regions in the main belt into chaotic resonance regions where they can get excited and eventually ejected from the main belt [41]. Due to this process, the population of NEOs and Mars-crossing asteroids remains in a steady state [7]. Also the dynamical spreading of asteroid families originating from a catastrophic disruption event in the distant past [8] or the slow displacement of artificial satellites can be explained by the Yarkovsky effect.

Besides the Yarkovsky effect, there are also other non-gravitational forces acting on small bodies in the solar system. These effects are not further considered in this work but should nevertheless be mentioned. The Yarkovsky-O'Keefe-Radzievskii-Paddack, or YORP effect [98], which is also caused by the Yarkovsky force, can alter spin rate and spin axis orientation over long time-scales. Also collisions play an important role in the long-term evolution of asteroids. They were long time thought to be the main driving mechanism for orbital migration of small bodies [20]. However, it was realized that the Yarkovsky effect is much more efficient in changing the orbits of asteroids [41]. On the other hand, the Yarkovsky and YORP effects are influenced by collisions as well, as the latter can change spin axis orientation and spin rate, too. For dust particles or very small bodies with radii of much less than 0.1 m, the Poynting-Robertson drag [15], a force due to radiation pressure which causes the particles to spiral inward, becomes more important. When studying the orbital evolution of active comets, rocket-like accelerations caused by outgassing near perihelion have to be taken into account [117].

# CHAPTER 3

# General-Purpose Computation on Graphics Processing Units

This Chapter is intended to give an introduction of what Graphics Processing Units (GPUs) are and how they can be used for general purpose computing. Although there are currently two different programming models in use (CUDA and OpenCL) the main focus of this work is on CUDA.

## 3.1 The History of Graphics Processing Units (GPUs)

A graphics processing unit is a dedicated device optimized for performing graphical computations in order to take load off the central processing unit (CPU). The term GPU was introduced by Nvidia in 1999 [26]. However, devices specialized for image processing and display have been existing since the 1980s. One of the first of these devices for personal computers (PCs) was the IBM Professional Graphics Controller (PGC), a video card consisting of an Intel 8088 microprocessor and 320kB of display RAM, introduced in 1984 [56]. The advantage of the IBM PGC was the dedicated microprocessor on board, which made it possible to directly perform primitive operations like rotating, translating or scaling on the image data. Most of the other devices available at that time were not much more than a simple frame buffer and thus needed the CPU to perform all computations on the image data and then write the resulting images to the frame buffer through the slow system bus. In the late 1980s the concept of moving computational capabilities to the GPU was improved and more data processing capabilities such as shaded solids and vertex lightning were implemented to the GPU, further relieving the CPU.

However, due to the lack of graphics hard- and software standards, early GPUs often showed compatibility problems making them cumbersome to use on the different computing platforms available at that time. These problems were not solved until the OpenGL application programming interface (API) was introduced in 1991. With OpenGL, new features to be implemented to the graphics pipeline, proposed by the hardware vendors, operating system designers and technology companies are standardized and then implemented to the API. Thus, more and more functionality has been added to the GPU. During the 1990s, several graphics APIs besides OpenGL like Glide API and Direct3D competed each other. While Direct3D became the dominant API to be used by the video game industry, OpenGL is now widely applied in professional computer-aided design (CAD) applications and for scientific visualization.

With increasing functionality and growing amount of graphics data to be processed, the demands on computation capabilities also increased. To challenge these demands individual units of the graphics pipelines were duplicated. For example the GeForce256

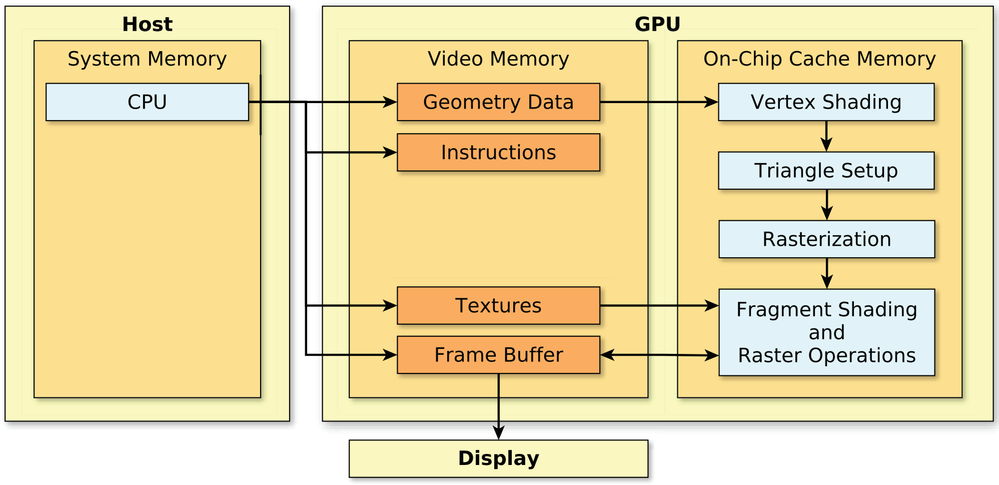

**Figure 3.1:** Graphics Processing Unit and the Graphics Pipeline

series, introduced by Nvidia in 1999 as the worlds first GPU, included 4 pixel shaders, texture mapping units and render output units each. This trend of implementing more and more parallel processing units on GPUs is still continuing. Modern GPUs contain thousands of processing units in order to meet the ever-growing requirements of latest video games, CAD and visualization software.

## 3.2 General-Purpose Computation on Graphics Processing Units

General-purpose computation on graphics processing Units (GPGPU) has become more and more popular in the recent years. It benefits from the enormous parallel computation power of modern GPUs. As described in the last section, the number of parallel computing units implemented on a single GPU has been increased and also more functionality has been added to these units. The most important processing units in terms of GPGPU are the shading units, which were intentionally build to perform primitive operations on graphics data like pixels, vertices or textures. With growing functionality programming (shading) languages were introduced which made it possible to implement more complex algorithms to the shading units. One of the first, who used this flexibility for a more general purpose than processing pixels and triangles were Fung et al. in 2002 [44]. They introduced OpenVIDIA, a library and API providing graphics hardware accelerated image processing and computer vision. With OpenVIDIA they exploited the parallel computation capabilities of GPUs and demonstrated how to set up a power-full parallel processing architecture with cheap standard desktop PC hardware. A comprehensive survey on early GPGPU work is given by Owens et al. [92].

However, GPGPU programming during its early phase was much more difficult than it is today. The programmer needed to have very specific understanding of the hardware and knowledge how to use the shading programming languages for general purpose computation. There also were many limitations such as a lack of 64-bit double precision and integer arithmetic support of early GPUs. This dramatically changed with the introduction of the Unified Shader Model in OpenGL and Shader Model 4.0 in Direct3D, respectively, in 2006. In these models the different shader types where replaced by universal floating point processing units — called unified shader. Unifying the different processing units has the advantage that the same instruction set is used for all shaders which makes it a lot easier to develop GPGPU software. The unified shading units are organized as stream processors in a single instruction, multiple data (SIMD) design. This way, the fixed function graphics pipeline evolved to a more flexible, highly parallel computing device. The first GPUs implementing the new architecture were Nvidia's GeForce8 series and ATI Radeon R600.

With the unified shaders the hardware vendors also introduced programming frameworks (Close to Metal by ATI which later became OpenCL and CUDA by Nvidia) dedicated to GPGPU programming. These tools provide higher level programming languages and an easy to use interface to the GPU. While CUDA is only supported on Nvidia hardware OpenCL can also be used on AMD GPUs and CPUs. Both frameworks follow a similar programming principle: First, the data is copied from the system memory (RAM) to the GPU memory. Then, a C-like program — called kernel, which utilizes the GPU's streaming processors, is executed. When the kernel has finished the result data can be copied back to the CPU memory or other kernels for further processing can be launched. Although both frameworks are commonly used, this work focuses on CUDA, which seems to be more popular in scientific applications.

Since 2006 the hardware and the programming frameworks have evolved. Many features vital for scientific computations have been introduced. In fact, GPGPU became so popular that in 2007 Nvidia started the Tesla series, GPUs specialized for general purpose computation. As of November 2015, there are two GPU powered supercomputers in the top ten of the TOP500[1] list of the worlds fastest computers. In this work however, no GPU super computer is used. The aim is to demonstrate how comparatively cheap GPUs can enable ordinary workstation PCs to perform scientific tasks which earlier required super computers.

## 3.3 CUDA Computing architecture on Nvidia GPUs

In the following sections the devices used for this work as well as their fundamental differences are introduced. Further, the techniques for developing GPGPU programs for NVIDIA GPUs are described, focusing mainly on the aspects essential for this work. A more comprehensive overview on these topics can be found in the "CUDA C best practice Guide" [24] and the "CUDA C Programming Guide" [25], both available online on the Nvidia website, as well as in several guidebooks for GPGPU development [59] [100].

### 3.3.1 Fundamental differences between GPU and CPU

As already mentioned in the last section, calculations performed on a GPU utilize many computing cores, the unified shaders. The main difference between a CPU core and a GPU core is that the latter is much more lightweight. This is due to the different tasks each device is designed for. CPUs are optimized for high execution speed on complex

1 http://www.top500.org/lists/2015/11/ (accessed: 2015-12-10)

sequential code which is basically achieved by high clock speed and complex on-chip logic like out-of-order execution or branch prediction. Due to limitations on the maximum clock speed it became common to implement multiple cores on a single CPU. Each CPU core is an out-of-order, multiple-instruction processor implementing the full x86 instruction set, having its own caches and register files. This means that on a multi-core CPU single threaded tasks can not be executed faster than on a single core CPU but more than one task can independently be executed at the same time. A task can also spawn multiple threads which run on more than one core in order to speed up computations. A GPU core is designed to perform simple operations and can utilize much less resources than a CPU core. However, as the number of cores on a GPU is very large, it is capable of executing many thousands of simple tasks at the same time and thus is very fast in processing large amounts of primitive graphics data in parallel. Table 3.1 lists the detailed specifications for the CPU and GPU used for this work.

**Table 3.1:** Comparison of some key-specifications of CPU and GPU used for this work

| | Intel Core i7-4930K | Nvidia GeForce GTX Titan Black |
|---|---|---|
| Number of cores | 6 | 2880 |
| Number of threads | 12 | 30720 |
| Clock speed | 3.4 GHz / 3.9 GHz (Turbo) | 889 MHz / 980 MHz (Boost) |
| L1 Cache | 32 KB per core | 48 KB per SMX |
| L2 Cache | 265 KB per core | 1536 KB |
| Last level cache | 12 MB shared | - |
| Memory size | up to 64 GB DDR3 | 6 GB GDDR5 |
| Memory clock speed | up to 1866 MHz | 3500 MHz |
| Memory bandwidth | 59.7 GB/s | 384 GB/s |
| Launch date | Q3 2013 | Q1 2014 |
| Release Price | $594 | $999 |

Figure 3.2 shows the configuration of CUDA cores on a Nvidia GeForce GTX Titan Black's Kepler GK110 GPU. Each of the green rectangles represents a single core. The cores are organized as 15 multiprocessors (SMX) each consisting of 192 single-precision cores, 64 double-precision cores and 32 special function units each. An SMX also contains 65 KB of shared memory and a $65536 \times 32$-bit register file, both shared among all cores within the SMX. The shared memory can be configured as 16 KB of L1 cache and 48 KB shared memory or 48 KB L1 cache and 16 KB shared memory. Finally, there is 1536 KB of L2 cache which is shared among all SMX.

The Titan Black graphics card was originally intended to be used for gaming. However, it employs the same graphics chip as the Nvidia Tesla K20, K20X and K40 computing modules and thus is often applied in GPGPU. In fact, as the Titan Black uses an overclocked GK110 chip, it outperforms the Tesla K40 by about 20% in processing power. The Tesla cards on the other hand are using ECC memory and are available with up to 12 GB GDDR5 RAM.

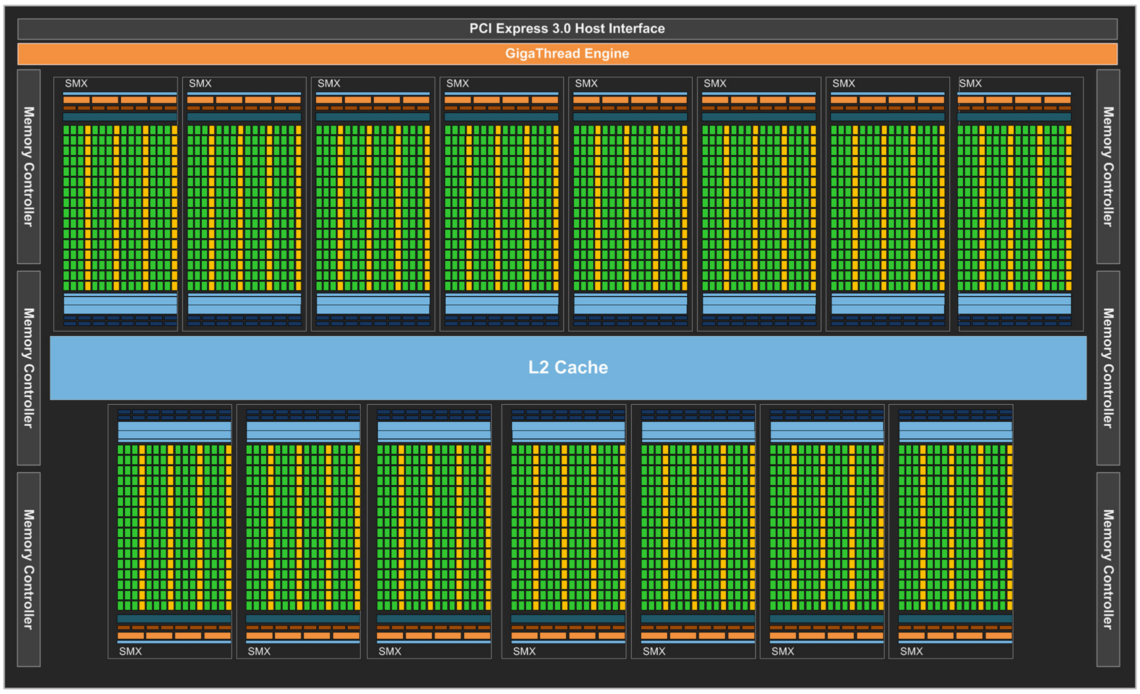

**Figure 3.2:** Nvidia GK110 GPU block diagram. Nvidia GK110 GPU block diagram. Cores are organized in 15 streaming processors (SMX). L1 cache is shared among all cores within a SMX, L2 cache is shared among all SMX.

In contrast to a CPU core, resources for a GPU core are very limited. Whereas on a CPU a thread can utilize many registers and large caches dedicated to every single core, on a GPU the number of registers and amount of cache for each thread is very limited as registers and caches are shared between many cores residing on the same multiprocessor. Fortunately, as the GPU threads are also more lightweight, switching between the threads is much less expensive and faster than on the CPU. While on the CPU, creating and scheduling threads typically requires several thousand clock cycles, the GPU can easily create and schedule groups of threads within only a few cycles [59]. This means that if threads are blocked, e.g. because they are waiting for data to be loaded, the GPU can easily continue working on other threads in the meantime. To achieve good throughput it is necessary to have a fast memory interface in order to keep the cores busy by quickly providing them with new data. Thus, the Titan Black is equipped with a memory interface allowing RAM bandwidths up to 384 GB/s, six times higher than memory transfers can be performed on the i7-4930K.

### 3.3.2 Programming model

In terms of GPGPU, the CPU is often referred to as the *host* while the GPU is called *device*. There are two different ways to perform general purpose calculations on the device: employing GPU-optimized libraries like cuBLAS, cuFFT or Thrust which automatically utilize the GPU, or using a C/C++ like programming language to write specialized device routines, called *kernels*, which have to be called manually. While for the first method, memory transfers between host and device are automatically performed, for the second method, which was used for implementing the algorithms for this work, the device memory must be allocated and initialized manually and after the kernel has finished the results must be transferred back to the host memory.

When calling a kernel, the GPU automatically creates, schedules and executes the number of threads required for processing. The threads are hierarchically structured in *warps*, which is a group of 32 threads each, *blocks* containing several warps and the *grid*, which represents all threads needed for executing a kernel. Within a warp, the threads start together at the same program address and one common instruction is executed at a time for all threads. Each thread however has its own instruction address counter and register state which means threads can branch the common execution at any time and run independently. A block represents multiple warps which are altogether executed on one of the GPUs multiprocessors. There may be several active blocks on one multiprocessor. The number of threads in a block can be defined when launching the kernel and should preferably be a multiple of 32 to not create warps containing inactive threads. Further, as all threads within a block share registers and cache, setting the block size has implications on the number of registers and amount of cache available for each thread.

### 3.3.3 Performance guidelines

There are several important design guidelines for developing GPU code. Those essential for this work are briefly reviewed in the following paragraphs.

#### Launch as many threads as possible

First of all, the programmer should identify portions of the program that are suited to be executed on the device. These are those parts showing a large degree of data parallelism. Applying some arithmetical operations to each element of an array of some tens to hundreds elements would run much faster on the CPU while the GPU would be faster for array sizes of thousands or more elements. To achieve adequate device utilization it is important to launch as many threads as possible. Modern GPUs are able to use several compute streams, allowing asynchronous kernel execution. This means that by calling several concurrent kernels even more threads can be launched.

#### Economize memory transfers between host and device

Another important rule is to keep the number of memory transfers between host and device memory as low as possible. Since these transfers employ the PCI interface, they can be very time consuming. Data should be kept on the device as long as possible and intermediate results should directly be used for further processing on the GPU rather than being downloaded and uploaded again later. Further, memory transfers can be *hidden* behind calculations: By working with different compute streams, memory operations can be performed while another kernel is currently active.

#### Maximize device memory bandwidth

The memory access pattern of the parallel threads also plays an important role. The programmer should make sure an advantageous pattern is used when accessing data residing in the device RAM (global memory) from the parallel threads. The memory interfaces are optimized for sequential access patterns which means that neighboring threads should access neighboring memory, e.g. thread $i$ works on $i$th data element. To achieve high memory throughput and low latency, the data structures to be transferred should map to the memory transaction sizes of 32, 64, or 128 bytes. If a thread requires data residing in global memory, the access for the whole warp is coalesced and one or more transactions are executed. For example, if each of the 32 threads within a wrap accesses one element of an array in a random access pattern and each element is 8 bytes in size, 32 32-byte transfers are executed. As the transfers can not be coalesced each transfer carries one element of the array while the remaining space is filled up with padding elements. This results in 1024 bytes are transferred to load only 256 bytes and thus, memory throughput is divided by 4. If the threads accessed the data in an ascending order, the data transfer could be coalesced and 4 8-byte elements are transferred within one 32-byte transaction. This results in only 8 transactions and the memory bandwidth being fully utilized.

Also unnecessary memory access should be prevented. If the same data is needed by all threads it would be inefficient to load the data by every single thread. Instead, one thread can be used to load the data into shared memory where every other thread in the same block can access it without latency. Loading the data into shared memory can even be done in parallel employing several threads within the current block.

#### Avoid divergence within a single warp

Divergent execution paths within the same kernel are also significantly affecting performance. As already mentioned, the threads within a warp can break common execution at any time and continue on separate execution paths. However, while the threads follow their own execution path, their instructions must be serialized, which increases the total number of instructions used to process the warp and results in reduced performance.

**Table 3.2:** GK110 specifications

| | |
|---|---|
| Total amount of shared memory per block | 49152 bytes |
| Total number of registers available per block | 65536 |
| Maximum number of threads per multiprocessor | 2048 |
| Maximum number of threads per block | 1024 |
| Warp size | 32 |

#### Find the right ratio between block size and register count

The number of warps on a certain multiprocessor depends on the number of registers each thread needs. For example, a kernel using 32 registers and requiring only a small amount of shared memory launched with a block size of 1024 threads on a GK110 chip, it is possible to execute two blocks of 32 warps each per multiprocessor, achieving 100 % occupancy (see table 3.2). If the first block is waiting for data to be loaded into shared memory from the device RAM, the warp scheduler can continue working on warps of the 2nd block. If each thread used just one more register, only one block could reside on each multiprocessor which means that it would be idle when the block is waiting for data as no other block would be active on the same multiprocessor and thus, occupancy would be reduced to 50%. To control occupancy, the maximum number of registers for each thread can be set when compiling the program. However, reducing the register count results in more data needed to be swapped to the cache, as now threads might not be able to allocate as many registers as they need. This introduces latency every time the data is loaded from and stored to the cache. A trade-off between the number of available registers per thread and the number of active blocks per multiprocessor must be found. Reducing the maximum register count to a value allowing the maximum occupancy of 2048 active threads per multiprocessor might be useless if a kernel required a large number of registers because it would spend most of the time waiting for data to be loaded from or stored to the cache. Setting no limit to the register count on the other hand may reduce the number of warps which can be executed in parallel. If the number of registers needed by each thread is large, only a few warps quickly consume the available space in the register file. The best values for block size and the maximum number of registers to be used per thread varies from device to device and is best determined with an occupancy calculator[1] and carrying out experiments.

1 http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls (accessed: 2015-04-08)

# CHAPTER 4

# cuSwift - a library of GPU based n-Body Integrators

The purpose of this chapter is to describe how the individual integration methods were implemented and optimized. There are three integration methods included in cuSwift so far: Wisdom Holman Mapping (WHM) [115] also known as Mixed Variable Symplectic method (MVS, see Section 2.3.1), the Regularized Mixed Variable Symplectic Integrator (RMVS, see section 2.3.3) [65] and a 15th order Radau integrator [40]. For each method, a parallel CPU as well as a GPU version is available which makes cuSwift suitable to be also used on multi-core computers without power-full GPUs. As most effort was put into the widely used symplectic integrators WHM and RMVS, only these two methods are described and evaluated. The original SWIFT package by Levison and Duncan as well as swifter[1], a translation of SWIFT to Fortran90, were taken as a reference for implementation and testing. In order to make it possible to study the orbital evolution under the Yarkovsky force, a model for the Yarkovsky effect, as described by Brož [12], was added to WHM.

## 4.1 Included Integration Methods

### 4.1.1 Wisdom-Holman-Mapping (WHM)

cuSwift is intended to be used for modelling the long-term evolution of minor bodies (test particles) in the solar system (or similar configurations). As already explained, this means that there are many more test particles than planets in the simulation, which results in most of the computation time spent for stepping the test particles. Thus, only the integration of the test particles has been parallelized. Implementing a parallel version of WHM (cuWHM) is straight forward. As the particles do not interact with each other they can be easily processed independently from each other in parallel.

#### CPU implementation

The focus for re-implementing the integration methods in SWIFT and swifter lay on optimizing the algorithms such that they can better utilize the computational capacities of modern workstation PCs. Not only the GPU but also the CPU version had to be parallelized such that all available CPU cores can be utilized. This was achieved by employing the OpenMP API for processing the test particles in parallel. With OpenMP, parallelization is achieved by adding preprocessor directives to certain sections of the code which should be executed in parallel. When running the program, these sections will be processed employing a predefined number of separate threads. After each of the

1 http://www.boulder.swri.edu/swifter/ (accessed: 2015-05-10)

parallel threads has finished, they join back to the master thread which continues sequential execution. Thread creation, distribution of workload among the threads as well as launch and synchronization are done automatically.

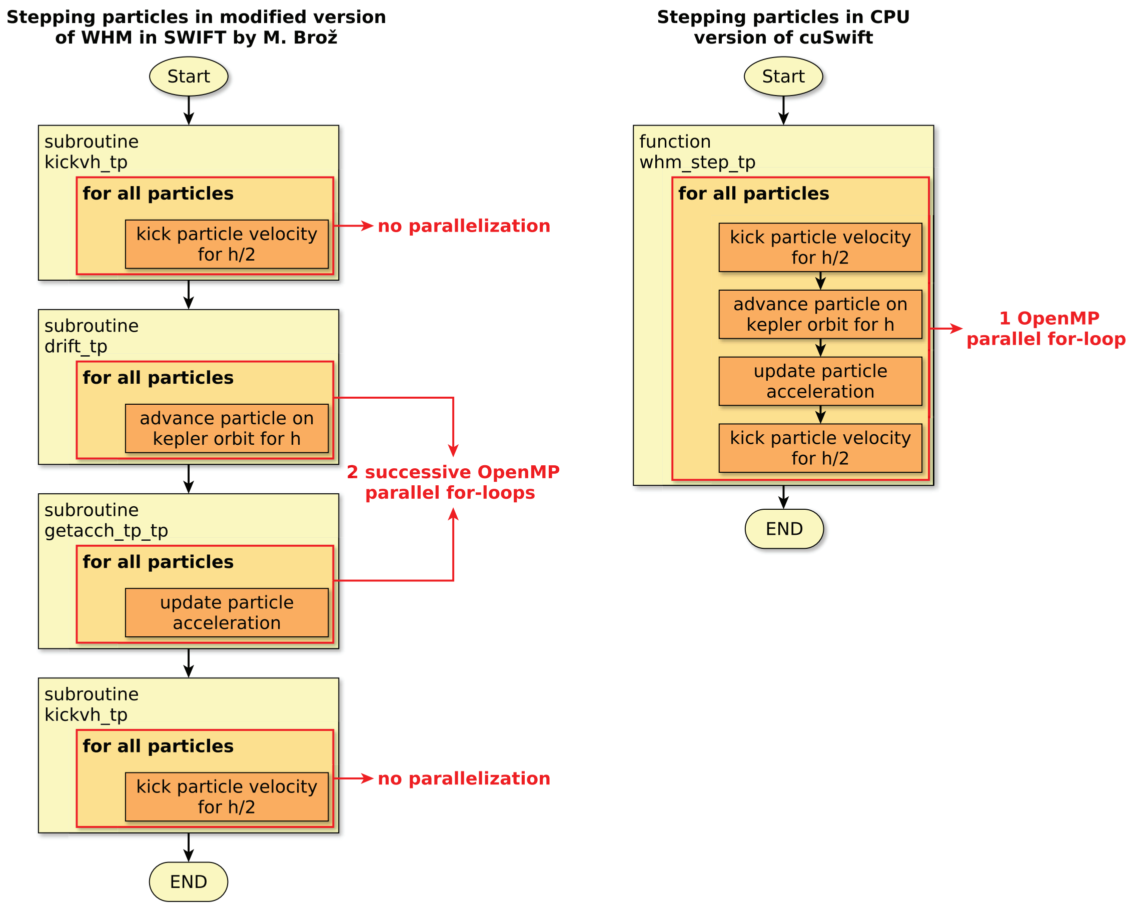
Stepping particles in modified version of WHM in SWIFT by M. Brož
Start
subroutine kickvh_tp
for all particles
kick particle velocity for h/2
no parallelization
subroutine drift_tp
for all particles
advance particle on kepler orbit for h
2 successive OpenMP parallel for-loops
subroutine getacch_tp_tp
for all particles
update particle acceleration
subroutine kickvh_tp
for all particles
kick particle velocity for h/2
no parallelization
END
Stepping particles in CPU version of cuSwift
Start
function whm_step_tp
for all particles
kick particle velocity for h/2
advance particle on kepler orbit for h
update particle acceleration
kick particle velocity for h/2
1 OpenMP parallel for-loop
END

**Figure 4.1:** Comparison of advancing the particles for one time step (h) as implemented in the parallel version of SWIFT by Brož [12] and cuSwift (CPU version). In the implementation by Brož, not the whole process is parallelized but two successive parallel for-loops are used. In cuSwift, the whole time step is processed in a single parallel for-loop

This method of parallelization has already been applied to WHM in SWIFT by Brož [12] but no significant speedup was achieved, which can be explained by the approach used by Brož and the design of the implementation of the algorithms in SWIFT. In SWIFT, each integration sub-step is done using a separate subroutine containing a for-loop over all particles each. This means that when parallelizing these subroutines with OpenMP, the overhead for thread management occurs several times during a single time step. As calculating the particles' velocities is computationally much cheaper than calculating the accellerations and advancing the positions, Brož paralellized the latter routines only (see Figure 4.1, left panel). In cuWHM, stepping the particles is done in a single routine containing one OpenMP parallel loop over all particles (see Figure 4.1, right panel). This reduces the amount of overhead resulting from thread management to a minimum. Further, as advancing the velocities is also included in the parallel loop, the whole process of stepping the particles is executed in parallel.

### GPU Implementation

The main difference between the CPU and the GPU version of cuWHM is that in the GPU version all the routines for processing the test particles were implemented in device code to be executed on the GPU. The initial positions and velocities are copied to device memory before starting the simulation. To keep time consuming memory transfers between host and device at a minimum, only the data absolutely necessary are transferred while the simulations are running.

Fortunately, for WHM, almost no communication between CPU and GPU is needed. The only data to be uploaded to the GPU during a single time step are the planet positions and velocities which are integrated on the CPU before the particles are stepped. The only data downloaded from the GPU each time step is a status flag indicating possibly discarded particles and, if particles were discarded, the position and velocity of only those particles. The whole arrays of particle positions and velocities are only copied back to the CPU when they are to be written to the output file.

In the following paragraphs, the main aspects of optimizing the algorithm for the GPU are described in detail.

### Data layout

As already explained in Section 3.3.3, the data layout plays an important role for achieving high device memory bandwidth utilization. Maximum bandwidth can only be achieved if the data to be accessed from global memory from each thread aligns with the transaction size and follows a regular access pattern. Employing such a pattern allows multiple elements to be loaded within a single memory transaction without using padding elements, which do not transfer any data. This has implications on the design of the data structures to be used [74]. A common approach of storing the data of a single particle would be a structure containing all information necessary. An array of structures (AoS) would then be used to represent the whole set of particles (see Figure 4.2, left panel). There are two reasons why this approach is not suitable for a GPU. First, a kernel which works on the particles position only (e.g. to calculate the acceleration) would have to load the data for the whole particle instead of the position only, resulting in unnecessary memory transfers. The second reason is that the structure contains nine 64-bit values and thus does not align with the transaction sizes of 32, 64, or 128 bytes. A more favourable design employed in cuSwift can be seen in the right panel of Figure 4.2. Here, the structure contains three arrays of aligned structures (AoaS) for the particles' positions, velocities and accelerations. Choosing this approach, a kernel working on the positions would load a single double4 value only. Further, the built-in double4 structure is 16-byte aligned and thus matches the memory transaction size. Although the coordinates are only 3-dimensional, double4 is used to store positions, velocities and accelerations as for double3 an 8-byte padding element would be added anyway in order to fulfill alignment. To save memory, the additional value is used to store further data. For planets which have similar data structures than particles, radius and mass are stored in the fourth double4 value for position and velocity, respectively. When incorporating the Yarkovsky effect, the particle radius is stored in the fourth double4 value of the particles velocity and the 3-dimensional spin vector is stored in a double4 variable with the spin rate stored in the fourth value.

```
typedef struct _Particle
  double3 x;
  double3 v;
  double3 a;
   ...
}Particle;

Particle *tp;
int ntp=100;

tp=(Particle*)malloc(ntp*sizeof(Particle));
```

```
typedef struct _Particles
  double4 *x;
  double4 *v;
  double4 *a;
   ...
}Particles;

Particles tp;
int ntp=100;

tp.x=(double4*)malloc(ntp*sizeof(double4));
tp.v=(double4*)malloc(ntp*sizeof(double4));
tp.a=(double4*)malloc(ntp*sizeof(double4));
```

**Figure 4.2:** Code example for two different data layouts containing cartesian positions, velocities and accelerations of a particle and their declaration. Left panel: Array of Structures (AoS). Right panel Structure of aligned Arrays (SoaA).

Note that loading a double4 value is split into two 128-bit loads, as the maximum transaction size for a single thread is 16 bytes. This splitting results in imperfect coalescing because of the inherent gaps when loading the lower and higher 16 byte part of double4. To test whether this influences overall performance a simple kernel adding four double values was implemented in a plain double version and a double4 version. For the plain double version, loading 4 double values from global memory results in 4 64-bit loads which are perfectly coalesced. The execution time for adding 50,000$\times$4 double values increased by less than 10 percent when using double4. This shows that the imperfect coalesced memory access in the double4 version is almost compensated by the fewer number of load/store operations. Since the actual kernels used in cuSwift do much more computational work than a single addition only, the ratio between calculations and load/store operations is much higher and thus, the overhead for using double4 can be neglected.

### Memory management

Another important aspect for achieving good performance is device memory management. As described in Section 3.3.1, there are different types of memory available on the GPU. Particle positions and velocities are initially located in global memory. When executing the kernel which steps the particles, as many threads are launched as there are particles in the simulation and each thread has to fetch the data of the particle it is working on from global memory. The planets' positions and velocities however, are the same for all threads. Thus, it would be inefficient if each thread loads them from global memory. To reduce the number of global memory accesses, shared memory is employed which can be accessed by all threads within the same block. This means that the planets' positions and velocities would have to be loaded only once for every block of parallel threads. To achieve maximal memory throughput, the planets' positions and velocities are loaded in parallel by the first $n$ threads of each block, where $n$ is the number of planets involved in the simulations. Note that this also implies that the number of threads per block must always be greater than or equal to the number of planets. However, as commonly block sizes of several hundred threads are used, this should not be an issue.

### Thread divergence

Divergent execution paths for different threads are another performance bottleneck. If threads within the same wrap diverge, they must be serialized which increases the number of total instructions required to process the wrap and thus significantly slows down execution. In order to maximize instruction throughput, the number of divergent execution paths should be minimized.

The drift step where a particle is advanced on its orbit is performed once every time step. This involves solving Kepler's equation

$$M = E - e\, sin\, E \tag{4.1}$$

for the eccentric anomaly E (M denotes the mean anomaly and e is the eccentricity) for each particle. As Kepler's equation is a transcendental equation, it can not be solved algebraically. Numerical methods are needed to evaluate E which makes the drift step to the computationally most demanding part of the integrator. There are various algorithms for efficiently solving Kepler's equation. In SWIFT and swifter, a recipe by Danby [27] is applied (further called Danby drift). The advantage of this method is that it completely avoids computationally expensive trigonometric functions. On the other hand, its execution path is highly divergent. Depending on the eccentricity, different algorithms are used to solve Kepler's equation. Thus, the drift step, as implemented in SWIFT and swifter, can not be efficiently executed on the GPU. In QYMSYM [80], a GPU accelerated n-Body integrator by More and Quillen, another approach for solving Kepler's equation is used. Their method employs universal variables, as described by Prussing and Conway, [95] to iteratively solve Kepler's equation (further called prussing drift). The advantage of this method is that it applies the same procedure, independent of the orbit's eccentricity. This results in almost no divergence in the execution path, making the prussing drift much more efficient when executed on the GPU. On the other hand, the prussing drift uses trigonometric functions such as sine, cosine, hyperbolic sine and hyperbolic cosine which, besides being computationally expensive, may produce slightly different results, depending on whether they are computed on the CPU or the GPU [25]. cuSwift contains both drift methods and it can be decided which method should be used on CPU and GPU before compiling. A detailed comparison between both drift methods on both devices is given in Sections 5.1.3 and 5.2.3.

### Dynamic memory reallocation

Since particles may be discarded during the simulation, the number of active particles may decrease with time. In SWIFT data for the particles is stored in arrays having a static size which can only be changed in the source code. This means that the whole package has to be re-compiled when changing the maximum number of particles. During the integration the number of particles remains constant and discarded or unused particles are skipped. This approach would be inefficient for a GPU implementation. As there would be always as many threads as particles, there might be many threads which do no work at all. The number of unused threads increases during the simulation when more and more particles get discarded. To avoid launching too many threads for discarded or unused particles in cuSwift, the size of the memory for the particles is dynamically reduced in a block-wise manner. If the number of inactive particles reaches the number of threads in a block, the memory for the particles is reorganized and the whole block is deallocated.

**Table 4.1:** Kernel functions for cuWHM

| Kernel | Description |
|---|---|
| whm_get_acch_tp_gpu | called once to initialize accelerations |
| whm_step_tp_gpu | performs a kick-drift-kick step for particles |
| discard_tp_gpu | checks for particles to be discarded |

### 4.1.2 RMVS

The RMVS integrator can be seen as an extension to MVS, which introduces close encounter handling for test particles. All optimization aspects mentioned in the last section were also applied for RMVS.

#### CPU Implementation

The CPU version of RMVS in cuSwift is implemented basically in the same way as cuWHM. The routines for close encounter detection and handling are directly translated from their Fortran versions and optimized for parallel processing using OpenMP while the planets are sequentially stepped.

#### GPU Implementation

Optimizing RMVS to efficiently be executed on the GPU requires some more effort. Resolving the encounters on the GPU is difficult because each time step only a few particles, if any at all, are involved in encounters. When performing encounter handling on the GPU, this means that a few threads would cause serious additional computational cost (see Section 2.3.3). The threads for the particles involved in encounters would break common execution and thus slow down the whole warp of parallel threads. A solution would be to separate the encountering particles on the device and launch another kernel for processing them on the GPU. By using concurrent threads, this could be done in parallel to stepping the particles which are not involved in encouters. However, even for scenarios which cause many encounters (see Section 5.4.1), the number of close encounters per time step would be so small that this approach would not be efficient at all. Thus, in cuRMVS close encounter processing is done on the CPU (see Figure 4.3). Before each time step it is checked which particles currently are or might be during the next time step located in a planet's encounter region. These particles are then downloaded to the host RAM to be processed by the CPU. After close encounter handling is finished, the new positions and velocities of the encountering particles are uploaded back to the GPU.

As explained in Section 3.3.3, it is possible to overlap kernel execution with memory transfers and CPU processing by employing different compute streams. In cuRMVS this feature is utilized for efficient close encounter handling. The whole process of downloading, processing and uploading the particles involved in close encounters is performed in parallel to stepping all other particles on the GPU. Thus, encounter handling does not significantly increases calculation time. For hiding close encounter processing completely behind GPU calculations the timing is very important. Memory transfers have to be optimized and CPU processing must be done using multiple threads in order to not slow down GPU calculations. Figure 4.3 shows a detailed flow chart of CPU and GPU operations and memory transfers for one cuRMVS time step.

### Memory management for close encounter processing

For each particle which is involved in a close encounter, its position, velocity and acceleration has to be downloaded to the CPU. If there are $n$ encountering particles, $3 \times n$ double4 values have to be fetched. The locations of the encountering particles in memory are randomly distributed. Thus, it is not possible to download them with a single memory transaction. Since performing $3 \times n$ individual double4 transfers would not be efficient, a more advanced method is applied to download and upload the data for the encountering particles in cuRMVS. The kernel checking for close encounters does not only mark the affected particles but also copies their positions, velocities and accelerations from the main data arrays to an auxiliary array in device memory. After encounter detection is finished, the data for the particles which has to be processed on the CPU can be downloaded from the auxiliary with a single transaction only. Doing so, it is important to ensure that concurrent threads which detect a close encounter at the same time do not access the same address in the auxiliary array. A counter variable which is accessible from all threads is employed to monitor the current number of close encounters and to mark the current index in the auxiliary array. This variable is incremented using the *atomicAdd* operation which ensures save reading, increment and writing of the counter by only one thread at a time. This ensures that different threads do not access the same index in the auxiliary array.

### Close encounter handling

Resolving the close encounters on the CPU is done in parallel using OpenMP. However, employing the maximum number of available CPU threads would not always be efficient. If there are for example only four particles involved in close encounters, the overhead for thread management on the CPU would dominate the whole procedure of close encounter processing when four CPU threads are used. Depending on the speed of the GPU, the total number of particles in the simulation and the number of particles involved in close encounters, the overhead for CPU thread management may even cause close encounter handling to take longer than stepping the particles which are not encountering a planet on the GPU. Thus, in cuRMVS, the number of threads which will be used to resolve the close encounters is variable and dynamically determined for each time step as

$$n_T = min(\, n_{T_{max}}, (int)(n_{TP_{enc}} \,/\, n_{TP_{min}})) \tag{4.2}$$

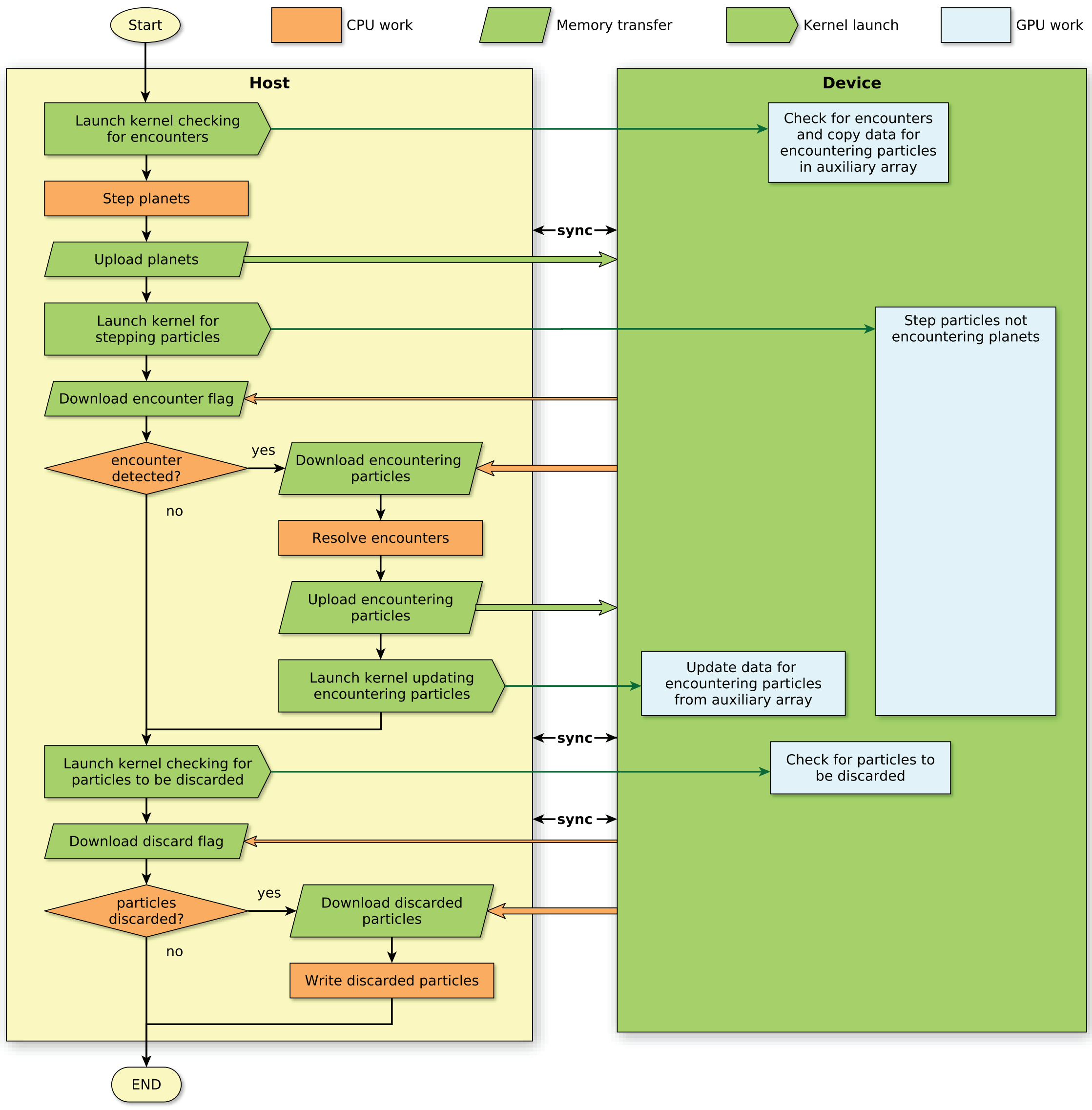
Start
CPU work
Memory transfer
Kernel launch
GPU work
Host
Device
Launch kernel checking for encounters
Check for encounters and copy data for encountering particles in auxiliary array
Step planets
sync
Upload planets
Launch kernel for stepping particles
Step particles not encountering planets
Download encounter flag
encounter detected?
yes
Download encountering particles
no
Resolve encounters
Upload encountering particles
Launch kernel updating encountering particles
Update data for encountering particles from auxiliary array
sync
Launch kernel checking for particles to be discarded
Check for particles to be discarded
sync
Download discard flag
particles discarded?
yes
Download discarded particles
no
Write discarded particles
END

**Figure 4.3:** Flowchart for one cuRMVS time step. Memory transfers are optimized and close encounter handling on the CPU is overlapped with stepping all other particles on the GPU.

where $n_T$ is the number of threads to be used, $n_{T_{max}}$ the maximum number of CPU threads, $n_{TP_{enc}}$ the number of particles involved in close encounters and $n_{TP_{min}}$ the minimum number of particles per CPU thread. The best values for $n_{T_{max}}$ and $n_{TP_{min}}$ are depending on the computer which is used and thus can be set by the user. For the machine used in this work (i7-4930K + GeForce GTX Titan Black) $n_{T_{max}} = 6$ and $n_{TP_{min}} = 15$ have proven to be a good choice.

After close encounter handling is finished, the new positions, velocities and accelerations of the encountering particles are uploaded back to the auxiliary array on the device and a kernel which copies the data back to the main data arrays is executed. Since this kernel alters only the particles marked as having a close encounter, it can be launched while the kernel stepping the particles which are not involved in close encounters is still active.

The advantage of this procedure is that close encounter handling is performed completely independently. Further, as close encounters are stepped in parallel, GPU and CPU are efficiently utilized. This results in close encounter handling does not significantly slow down the simulation as will be shown in Section 5.4.2, where the performance of all available implementations of the integrators are compared.

**Table 4.2:** Kernel functions for cuRMVS

| Kernel | Description |
|---|---|
| whm_get_acch_tp_gpu | called once to initialize accelerations |
| rmvs_check_gpu | checks for close encounters and copies data for encountering particles to auxiliary array |
| whm_step_tp_gpu | performs a kick-drift-kick step for particles not involved in close encounters |
| rmvs_update_tp_enc_gpu | copies data of the encountering particles from the auxiliary array to the main arrays |
| discard_tp_gpu | checks for particles to be discarded |

## 4.2 Yarkovsky effect

For the implementation of the Yarkovsky effect, the modified version of SWIFT by Brǒz (swift_mvs2_fp_ye) [12] was used as a reference. In swift_mvs2_fp_ye, Brǒz applied theories developed by Vokrouhlický and Farinella [107][109] who introduced models for the diurnal and seasonal Yarkovsky effect for spherical bodies. A very detailed description of the implementation of the Yarkovsky force in swift_mvs2_fp_ye can be found in Brǒz' dissertation [13]. So far, in cuSwift, the Yarkovsky effect can be considered with the WHM integrator only.

Calculating the acceleration caused by the diurnal component of the Yarkovsky effect involves the evaluation of transcendental functions. On the GPU, these functions are available in different versions. The fastest way of evaluating transcendental functions is to use their intrinsic versions, which are implemented to the GPU's special function units. Unfortunately, the intrinsic versions are available in single-precision only. Further, the algorithms for evaluating transcendental functions on the GPU differ from those implemented to the CPU. This means that the GPU may produce approximations which do not comply to the IEEE standard for single-floating point arithmetic. In cuSWIFT, double-precision is used for transcendental functions. There are double-precision versions for those functions available on the GPU, but these functions are not implemented to the special function units and thus must be evaluated utilizing the single and double-precision cores, slowing down computations. For the case both, sine and cosine of an argument are needed, the `sincos` function, which simultaneously calculates sine and cosine, is employed. Note that even the double-precision versions for the transcendental functions on the GPU do not fully comply to the IEEE standard for floating point arithmetic. The maximum errors to be expected are listed in the CUDA Programming Guide [25]. For the functions used in cuSwift, the errors are also listed in table 5.1 in the next section.

The routines for calculating the Yarkovsky force are not an exact translation from their Fortran version in swift_mvs2_fp_ye because accuracy and performance issues have been identified when studying the original code. These issues will be described in detail in section 5.3.1 where the implementation of the Yarkovsky effect in cuSwift is evaluated.

**Table 4.3:** Kernel functions for the Yarkovsky effect

| Kernel | Description |
|---|---|
| whm_get_acch_tp_gpu_yarko | calculates the accelerations due to gravitation and the seasonal and diurnal component of the Yarkovsky force (called once to initialize accelerations) |
| yarko_update_seasonal_coeff_gpu | calculates auxiliary coefficients needed to evaluate the seasonal component of the Yarkovsky force (not called at each time step) |
| whm_step_tp_gpu_yarko | performs a WHM kick-drift-kick step and considers the acceleration due to the seasonal and diurnal component of the Yarkovsky force when evaluating the acceleration |

# CHAPTER 5

# Validation and Benchmarking

The first steps to validate cuSwift have been already performed during implementation. Results for each function have been compared with the corresponding routines in SWIFT/swifter. To test the entire algorithm runs with identical starting conditions were carried out with swifter and cuSwift and the binary output files were compared. These tests were done using only those optimization flags which are available for both, the Fortran (gfortran) and C/C++/CUDA (gcc, nvcc) compilers used to compile SWIFT, swifter and cuSwift. For the nvcc compiler, the options `-prec-div` and `-prec-sqrt`, which control the precision of division and square root on the GPU, were set to use the IEEE round-to-nearest mode ensuring identical results for these operations on both devices. With these settings the CPU implementation of cuSwift exactly reproduces the results obtained with swifter. The binary output files for runs using the same initial conditions are identical for swifter and cuSwift. However, results obtained with the GPU may differ from those obtained with the CPU even if the compiler optimization flags were the same. As already mentioned, this is due to the results of some transcendental functions on the GPU may differ from those on the CPU. A list of these functions which are used in cuSwift as well as their maximum ULP (unit in the last place) error on the GPU is given in table 5.1. The maximum ULP error corresponds to the absolute value of the difference in ULPs between a correctly rounded double-precision result and the result obtained on the GPU [25].

The CUDA compiler also introduces some optimizations which are not available for the C/C++ compiler. Most of these optimizations such as forcing the use of intrinsic routines for transcendental functions or relaxing the accuracy in division or square root are not applicable for cuSwift because they would introduce a serious loss of precision. But there is also an optimization for the nvcc compiler which might even result in higher accuracy on the GPU than on the CPU. Using this optimization called fused multiply-add (FMA), terms of the form $x = a * b + x$ will be evaluated with a single instruction using a single rounding operations only [111]. This leads to results possibly not complying to the IEEE floating point standard but, as the same term on the CPU is evaluated with two rounding operations, the result computed on the GPU should be more accurate.

Another aspect to investigate is how the two different methods for the drift-step which may also produce slightly different results influence accuracy. Especially during close encounters and for chaotic orbits the tiniest differences may lead to completely diverging results. To understand how the individual implementations and optimizations affect the accuracy of the integrators, test cases were set up and experiments were carried out. In the following sections these experiments and their results are described in detail.

**Table 5.1:** Mathematical standard library functions and their maximum ULP (unit in the last place) error used in cuSwift

| Function | max ULP error | Usage in cuSwift |
|---|---|---|
| `cbrt(x)` | 1 | Danby drift |
| `exp(x)` | 1 | Yarkovsky effect |
| `log(x)` | 1 | Yarkovsky effect |
| `sin(x)` | 1 | Prussing drift; Yarkovsky effect |
| `cos(x)` | 1 | Prussing drift; Yarkovsky effect |
| `tan(x)` | 2 | Yarkovsky effect |
| `sincos(x,sptr,cptr)` | 1 | Prussing drift; Yarkovsky effect |
| `acos(x)` | 1 | Yarkovsky effect |
| `atan2(y,x)` | 2 | Yarkovsky effect |
| `sinh(x)` | 1 | Prussing drift; Yarkovsky effect |
| `cosh(x)` | 1 | Prussing drift; Yarkovsky effect |
| `pow(x,y)` | 2 | Prussing drift; Yarkovsky effect |

## 5.1 Test setup for WHM

### 5.1.1 The dynamics of Pluto

For validating cuWHM, the dynamical properties of Pluto are providing a good test case. The orbit of Pluto is one of the most peculiar orbits in the solar system. It is highly inclined ($i \approx 17°$) and very eccentric ($e \approx 0.25$). In fact, the eccentricity is so large that when Pluto reaches its perihelion it is closer to the Sun than Neptune. A planet-crossing orbit might lead to the assumption that Pluto might not be stable over very long time scales and will eventually be removed from its orbit by close encounters with Neptune. However, as will be explained in the next section, the orbit of Pluto is stable over millions of years. Being such an oddity in the solar system, the dynamics of Pluto have been studied in great detail since its discovery of Pluto in 1930.

#### Why Pluto?

The orbits of Neptune and Pluto are in a 3:2 mean-motion resonance. While Neptune revolves three-times three revolutions around the Sun, Pluto revolves twice. In 1965, Cohen and Hubbard [22] integrated the orbits of the five outer planets for 120 kyr and found that Pluto and Neptune are locked in that resonance in a way that the distance between Neptune and Pluto never gets less than 18 au — a first indicator for the long-term stability of Pluto. Later, in 1971, Williams and Benson [112] reported Pluto's argument of perihelion librates such that Pluto is about 8 au above the orbital plane of Neptune whenever it is closer to the Sun, another mechanism preventing close encounters between the two bodies.

During the 1980's more power-full computers became available for studying the long-term orbital evolution, allowing much longer simulation times. In 1986, Applegate et al. [3] performed integrations of the outer solar system ranging more than 100 Myr in the past and 100 Myr into future on a special purpose computer called *Digital Orrery*, which was exclusively designed for studying planetary motion [2]. They were looking for chaotic behavior in the motion of the planets, which is given if two initially very close trajectories exponentially diverge with time. They found no evidence for chaotic motion of Pluto but reported that orbits very similar to that of Pluto clearly show chaotic behavior.

The famous LONGSTOP integrations [17], performed in the late 1980's, revealed some very important insights in the long-term orbital evolution of the outer solar system. The two experiments LONGSTOP 1A and 1B were carried out on a CRAY-1S supercomputer and simulated the orbital evolution of Jupiter, Saturn, Uranus, Neptune and Pluto over time spans of 9.3 Myr and 100 Myr. Examining the output of these simulations, in 1989, Milani et al. [78] concluded that the orbit of Pluto is indeed chaotic. It was also found that due to the locking in various orbital resonances, Pluto seems to be trapped in a chaotic region and thus its orbit is likely to be stable over very long time scales. In 1988, Sussman and Wisdom [102] used the *Digital Orrery* to perform integrations of the outer solar system, covering up to 845 Myr. They confirmed that the orbit of Pluto is chaotic and reported a Lyapunov time of 20 Myr. The Lyapunov time is as a characteristic value describing the chaotic nature of a dynamical system. Calculations beyond the Lyapunov time are speculative.

The chaotic nature of Pluto's orbit makes it very sensitive to small errors and variations of numerical methods. The fact that Pluto's orbit is chaotic but nevertheless stable over very long time scales and that it is studied in such a great detail with many different integration methods makes Pluto a good example for validating the various optimizations introduced with cuWHM as well as the numerical differences arising from the different computing platforms.

### 5.1.2 A fictitious population of Plutos

In order to perform reliable experiments with conclusive statistics, a single orbit would not be sufficient. Hence, a sufficiently large number of fictitious Plutos was generated. These fictitious bodies have similar orbital properties like Pluto but are distributed over a larger volume in phase space. In fact, several such objects have been recently discovered in the Kuiper belt. They are making up a sub-population in the trans-Neptunian region called *Plutinos*. To generate an adequately large population of these objects, a similar approach as in the work of Tiscareno and Malhotra [104], who studied the chaotic diffusion of resonant Kuiper belt objects, was taken. First, 1,000 particles were generated which have the same semi-major axis as Pluto but randomly differ by up to $\pm$ 0.05 and $\pm$ 1° in eccentricity and inclination, respectively, with the introduced deviations being uniformly distributed. Longitude of ascending node ($\varOmega$), argument of perihelion ($\omega$) and mean anomaly $M$ for each object were chosen such that the critical argument for the 3:2 mean-motion resonance between the real Pluto and Neptune is violated by less than 1°. The critical argument is given as

$$\varPhi_{3:2} = 3\lambda_{res} - 2\lambda_N - \varpi_{res} \tag{5.1}$$

where $\lambda = M + \Omega + \omega$, the mean longitudes of the resonant object and Neptune and $\varpi_{res} = \Omega_{res} + \omega_{res}$ the longitude of perihelion of the resonant object. The fictitious Plutinos were then integrated with cuWHM in a system containing the Sun and the four outer planets for 10 Myr using a time step of 0.5 yr. Particles leaving the resonance region were discarded from the simulation. A lower and upper limit of 38.77 and 40.17 au [104] for the particles semi-major axis was applied to determine when a particle has left the resonant region. After 10 Myr, 301 particles remained. Figure 5.1 shows snapshots of the planets and the remaining particles in a heliocentric reference frame at different time steps. The fictitious Plutinos populate two distinct regions but are leaving a gap each. Whenever Neptune approaches either of the regions, it will be located in these gaps and thus does not encounter any of the particles. These 301 particles are trapped in the protecting mechanisms in a similar way like Pluto and are proven to be stable for at least 10 Myr. Together with the real Pluto, they are taken as a population of test particles for validating cuWHM.

### 5.1.3 Validating cuWHM

To validate cuWHM, a similar approach as the one by Sussman and Wisdom for the *Digital Orrery* [102] was chosen. They integrated Pluto as a mass-less particle in a system containing the Sun and the outer planets 3 Myr into the future and, afterwards, 3 Myr backward in time to see how well the *Digital Orrery* reproduces the initial conditions. The same test, but with the 301 fictitious Plutinos instead of only Pluto was used for cuWHM. As the WHM integrator is time reversible, the position of the bodies after the integration should theoretically be exactly the same as before. Truncation errors made during rounding however will cause the reproduced initial positions to differ from the initial positions. And due to the fact that the orbit of Pluto is chaotic, these differences can become very large even for very small numerical deviations.

To measure the accuracy of the WHM integrator, the distances between the particles' initial and reproduced initial positions (hereafter position errors) were examined. Several test runs were preformed in order to quantify the effects of the different optimizations and computing platforms and histograms of the position errors for each run were created. However, when looking at the Histograms alone, the results of the individual runs are almost indistinguishable, making it hard to see which run was more accurate. To better evaluate the differences between individual runs, the median of the position errors for each run was considered as well.

The position errors for most of the fictitious Plutinos were in the order of $10^{-6}$ au but some of them also showed very large errors of up to several tens of au, indicating a very high degree of chaos for those orbits. For better readability, only position errors up to $1 \cdot 10^{-5}$ are shown in the histograms and the number of particles which exceeded this limit is mentioned for each run.

The cubic root function (`cbrt(x)`), used by the Danby drift method, which has 1 ULP error on the GPU is only used for non-elliptic or very eccentric orbits. This means that because the orbits of the fictitious Plutinos have moderate eccentricities, the results should be the same for CPU and GPU when using the Danby drift on both devices. Indeed, the output files produced by GPU and CPU are identical. Hence, no detailed comparison

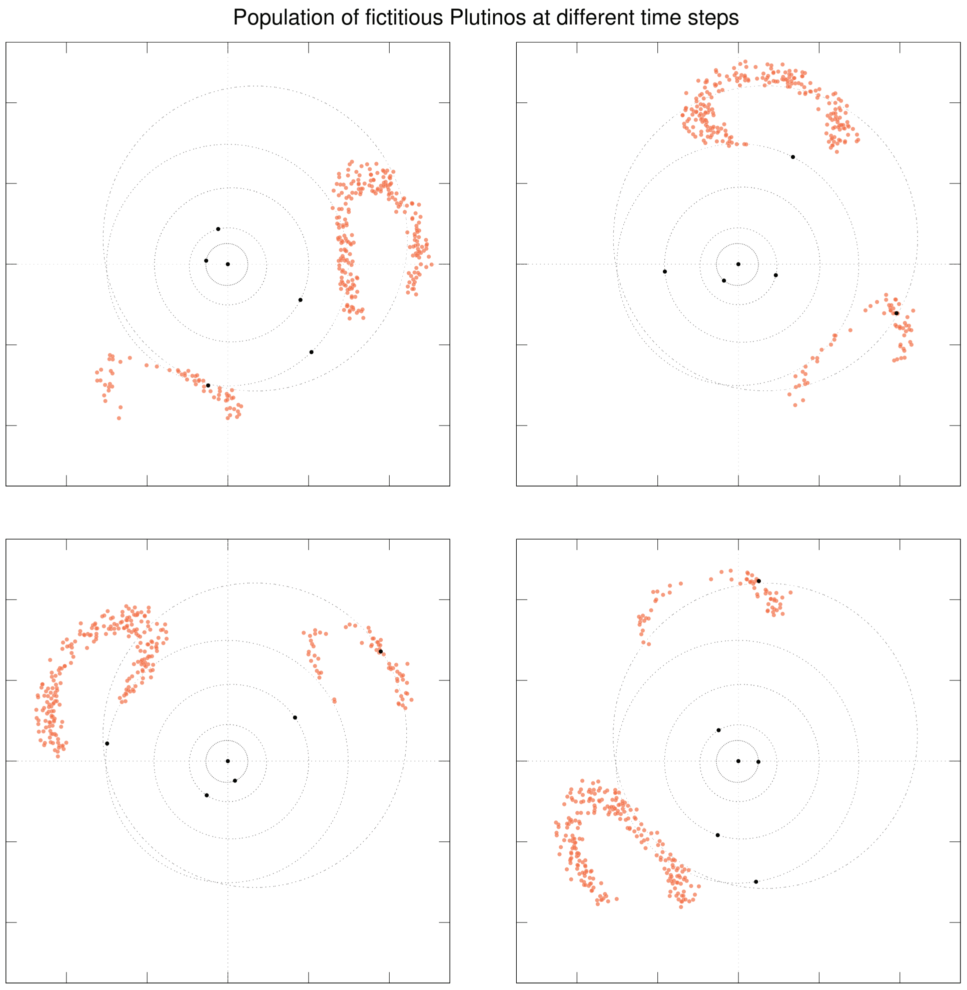

**Figure 5.1:** Snapshots of Jupiter, Saturn, Uranus, Neptune, the real Pluto (black dots) and the population of fictitious Plutinos (orange dots) which are on orbits stable over at least 10 Myr. The Plutinos populate two distinct regions and are forming a gap. When Neptune enters either of the two regions it will be inside the gap and thus no close encounters will occur.

between CPU and GPU is needed for this case. The influence of the error caused by the `cbrt(x)` function is evaluated when validating cuRMVS in Section 5.2.

### Fused multiply-add

As already mentioned, using the FMA nvcc compiler flag ((`fmad`)), the GPU should produce more accurate results because a multiplication and an addition can be performed using a single rounding step only. To test this hypothesis, a run without FMA was compared with a run with FMA enabled. Both runs were using the Danby drift method. Figure 5.2 shows the histograms of the position errors for both runs plotted over each other. It can be seen that the median position error after 6 Myr of integration is almost the same, regardless if FMA is enabled or not. Without FMA, the median error is $2.40 \cdot 10^{-6}$ au while when enabling FMA, the value slightly decreases to $2.37 \cdot 10^{-6}$ au. The number of particles with errors larger than $1 \cdot 10^{-5}$ au is almost the same for both runs. Without FMA, 60 particles produced errors larger than $1 \cdot 10^{-5}$ au while 63 particles exceeded the limit when FMA was enabled.

According to this test, FMA does not significantly influence accuracy. Thus, it can be considered safe to use the FMA option when compiling cuSwift.

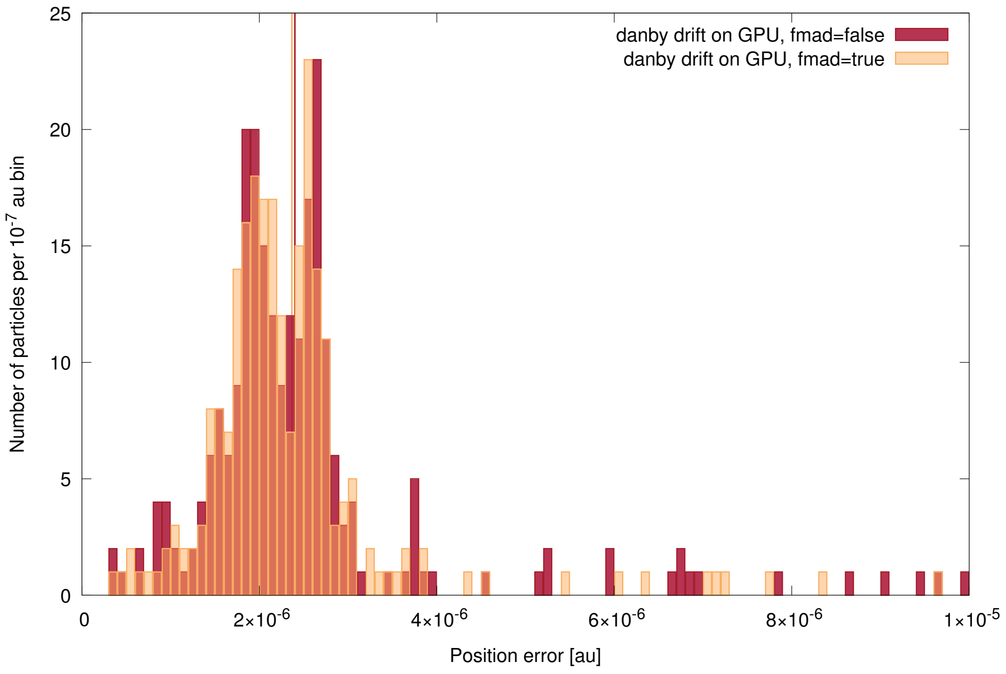

**Figure 5.2:** Histograms of errors of the reproduced initial conditions after integrating 3 Myr into the future and back with and without the FMA optimization. The vertical lines represent the median values of the position errors for each run.

### Drift methods

cuSwift contains two methods for advancing the bodies on their Kepler orbit. It can individually be chosen which method should be applied on CPU and GPU during compilation. For this particular test case, the Danby drift method reproduces exactly the same results, regardless which computing device is used. The Prussing drift method however makes use of `sin(x)`, `cos(x)`, `sinh(x)`, `cosh(x)` and `sincos(x, sptr, cptr)` which have an error of 1 ULP each and the `pow(x,y)` function, which has a maximum error of 2 ULP. Thus, two runs using the Prussing drift method are very likely to produce different results on the different devices. Several combinations must be compared in order to evaluate the different drift methods: First, the Danby drift and Prussing drift must be compared on the CPU in order to ensure they produce comparable results at all. Second, to examine the effect of the errors of the mathematical functions on the GPU, the CPU Prussing drift run must be compared to a GPU run also using the Prussing drift but leaving the FMA optimization disabled. Then, the CPU run using the Danby drift must be compared to a GPU run using the Prussing method in order to evaluate the differences of both methods and devices. Finally, a GPU run using all optimizations (FMA and Prussing drift) must be compared to the CPU run using the Danby drift to evaluate the fully optimized GPU implementation by comparing it to the reference implementation.

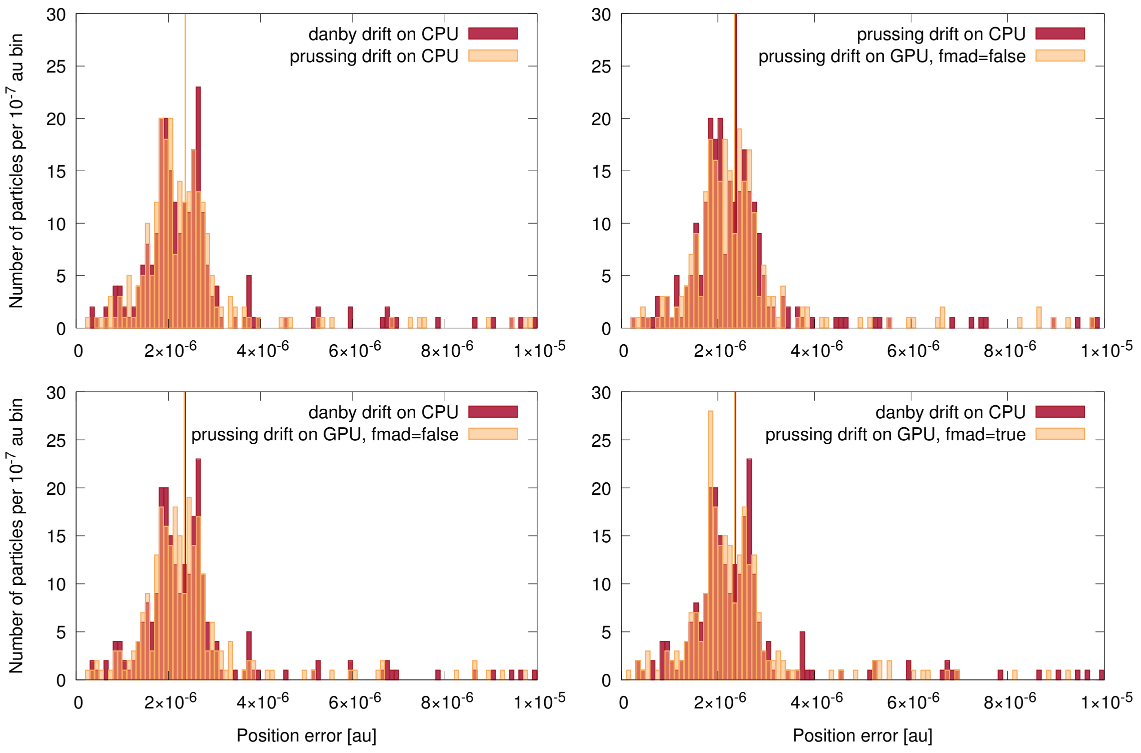

**Figure 5.3:** Same as Figure 5.2 but for evaluating how the different drift methods and FMA affect the position errors.

Figure 5.3 shows the histograms of the position errors for all cases to be discussed. In the upper left panel, the different drift methods are compared on the CPU. Using the Danby drift, 60 particles have position errors larger than $1 \cdot 10^{-5}$ au while when using the Prussing drift, the error for 61 particles exceeded the maximum. The median value of the position errors for both runs is $2.37 \cdot 10^{-6}$ au. This means that the different drift methods are producing almost identical results on the CPU — at least for elliptical orbits with moderate eccentricities. The upper right panel of figure 5.3 shows the differences for the Prussing drift on CPU and GPU. For the GPU run, the position errors of 56 particles were larger than $1 \cdot 10^{-5}$ au and the median value is $2.35 \cdot 10^{-6}$ au. Producing almost identical results as the CPU version, the errors of the mathematical functions on the GPU can be neglected. This implies that the two different drift methods are comparable in accuracy on both devices, as can be seen in the lower left panel of Figure 5.3. Finally, in the last panel of that figure the reference implementation is compared with the fully optimized GPU implementation. With FMA enabled, the median value for the position error is $2.34 \cdot 10^{-6}$

au and 61 particles exceeded the $1 \cdot 10^{-5}$ au limit.

In general, none of the different combinations shows significant deviations, which is very good, as it shows that the results produced with the different drift methods and optimizations are almost identical for both computing platforms. According to this test, it can be considered as safe to use either of the two drift methods in cuWHM. As the Danby drift method implemented in SWIFT/swifter is optimized to be used on the CPU, this method is the default setting for the CPU while the Prussing drift is the default setting for advancing the particles on their Kepler orbit on the GPU.

#### Conclusions

This test does not only evaluate the changes to the original version of WHM, but also demonstrates how remarkably well the method works in general. The position errors for most of the particles after 6 Myr of simulation are at about $2 \cdot 10^{-6}$ au, a distance as far as only 300 km. The errors in the reproduced positions for the real Pluto were less than $2.06 \cdot 10^{-6}$ au. Sussman and Wisdom [102] reported in their test an error in the position of Pluto after 6 Myr of integration of about $1.17 \cdot 10^{-6}$ au for a 12th order integration method on the *Digital Orrery* using a time step of 32.7 days. WHM, a symplectic 2nd order integration method, reaches comparable results when using a time step 5 times longer than the 12th order method employed by the *Digital Orrery*.

## 5.2 Test setup for RMVS

The procedure of testing the accuracy of numerical integration methods for orbital motion described in the last section is working very well as long as the particles do not have their orbits significantly changed due to close encounters with Planets. During close encounters, the smallest deviation in positions and velocities result in tremendously deviating trajectories. Thus, comparing the initial position with the reproduced initial position of a particle integrated forth and back in time would not be meaningful. For evaluating the newly implemented version of the RMVS integrator, a different approach is needed.

### 5.2.1 Monitoring the Jacobi constant in the restricted three body problem

The circular restricted three body problem which consists of two massive bodies on circular, non-inclined orbits around the center of mass and a third mass-less body may seem a huge simplification. However, the advantage of this simplified depiction is that, in contrast to the non restricted problem, there exists an analytic integral of motion: the Jacobi constant $C_J$, which, in a barycentric, co-rotating reference frame, is defined as

$$C_J = n^2(x_{tp}^2 + y_{tp}^2) + 2\left(\frac{Gm_1}{r_1} + \frac{Gm_2}{r_2}\right) - \left(v_{x_{tp}}^2 + v_{y_{tp}}^2 + v_{z_{tp}}^2\right) \tag{5.2}$$

where $n$ is the mean orbital motion, $x_{tp}$ and $y_{tp}$ the x- and y-coordinate of the mass-less particle, $r_1$ and $r_2$ the distances between massive bodies and the particle and $v_{x_{tp}}$, $v_{y_{tp}}$ and $v_{z_{tp}}$ the particles' velocity.

To evaluate cuRMVS, a similar test as used by Levison and Duncan [65] was performed. In order to determine the accuracy of RMVS, they examined the change in the Jacobi constant of planet-crossing test particles in a three body system consisting of the particle, the Sun and a planet with the mass of Jupiter on a circular, non-inclined orbit at 5.2 au.

### 5.2.2 Creating a set of particles frequently involved in close encounters

For the test, a set of test particles which undergo many close encounters is required. In order to create this test set, 500 particles were generated where semi-major axis and eccentricity of the particles were chosen such that their orbits crossing the orbit of Jupiter. Inclination was set to values of less than 30° and the other elements were uniformly distributed over the phase space. These particles were then integrated for 1 kyr in a system containing Jupiter on a circular orbit around the Sun using cuRMVS. The time step was set to 1/10 yr and the Danby drift method was used for advancing the bodies on their orbits. During the integration, the number of close encounters for each particle was monitored. The 10 particles having the highest number of close approaches within the Hill sphere of Jupiter were selected for evaluating cuRMVS. Figure 5.4 shows the initial positions and orbits of the particles chosen for testing. During 1 kyr of integration the particles penetrate Jupiter's Hill sphere between 7 and 22 times.

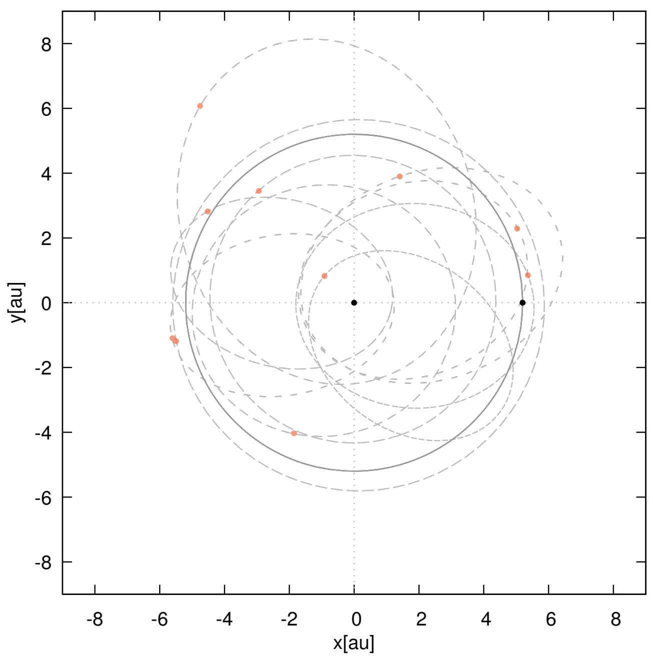

**Figure 5.4:** Initial conditions for the test in heliocentric reference frame: 10 mass-less test particles (orange) and the massive bodies Sun and Jupiter (black) in a circular restricted three body problem. The particles are on Jupiter-crossing orbits and undergo 7 to 22 close encounters during the 1 kyr integration.

### 5.2.3 Validating cuRMVS

To evaluate cuRMVS, runs of the original version of RMVS in SWIFT (swift_rmvs4) and swifter (swifter_rmvs) and the newly implemented version in cuSwift were performed and compared. Further, cuRMVS runs with the different drift methods on the CPU and a GPU run employing the Prussing drift method and FMA were tested as well. In order to compare the different runs, the relative change in the Jacobi constant for each particle was plotted and compared. Each plot contains the results of two runs, plotted on top of each other so that it is easy to see where the value of $C_J$ starts to diverge. There are rather large excursions of the Jacobi constant during close encounters which are not shown in the plots. However, after the encounter, the Jacobi constant always reaches similar values as before [65].

When using the Danby drift method, the CPU version of cuRMVS exactly reproduces the results of swifter_rmvs. Hence, comparison for this case is obsolete. To evaluate the differences between cuRMVS and swift_rmvs4, a run of swift_rmvs4 was compared with a run of the CPU version of cuRMVS using the Danby drift method, which also serves as a reference run for the remaining tests. Then, in order to validate the different drift methods, a CPU run employing the Prussing drift was compared with the reference run. Finally, to evaluate the fully optimized GPU implementation, a run on the GPU using the Prussing drift and FMA was compared with the reference run.

### Conclusions

Figure 5.5 shows the fractional error in the Jacobi constant for the particles using the original implementation of RMVS in SWIFT and the CPU version of cuSwift. Due to slight differences in the way the terms for computing the accelerations of the encountering particles are evaluated in both implementations, the results of cuRMVS (and swifter_rmvs) are not exactly the same as the results obtained with swift_rmvs4. However, according to the plot, the two implementations produce very similar results. For the first $\approx 400$ years, there is no noticeable difference between the two runs. Then, the Jacobi constant for some of the particles starts to diverge but no significant deviations can be observed. For some Particles, the Jacobi constant is the same for both versions over the whole run. This shows that both implementations do not produce identical but very similar results. Thus, the different implementations can be considered equivalent.

Figure 5.6 shows the fractional error in the Jacobi constant for the 10 test particles when employing the different drift methods on the CPU. The particles taken for testing are on very eccentric orbits. Further, when a particle penetrates a planets Hill region, it is stepped on an orbit around the encountered planet rather than an orbit around the Sun, which results in an hyperbolic orbit. This means that in contrast to the test evaluating cuWHM, this test also compares the different subroutines for highly eccentric and hyperbolic orbits in the Danby drift method with the Prussing drift, which uses the same procedure for all forms of Kepler orbits. Again, both runs are indistinguishable for the first $\approx 400$ years and then start to diverge. However, no significant outliers can be seen which implies that the Jacobi constant is conserved to a similar degree for the different drift methods and thus, both methods are equivalent, even for highly eccentric and hyperbolic orbits.

Finally, a run on the GPU using the Prussing drift and FMA was compared to the reference run. As can be seen in Figure 5.7, this run produced very similar results as well. This shows that the errors arising form the numerical differences of the mathematical functions on the GPU do not lead to less accurate results.

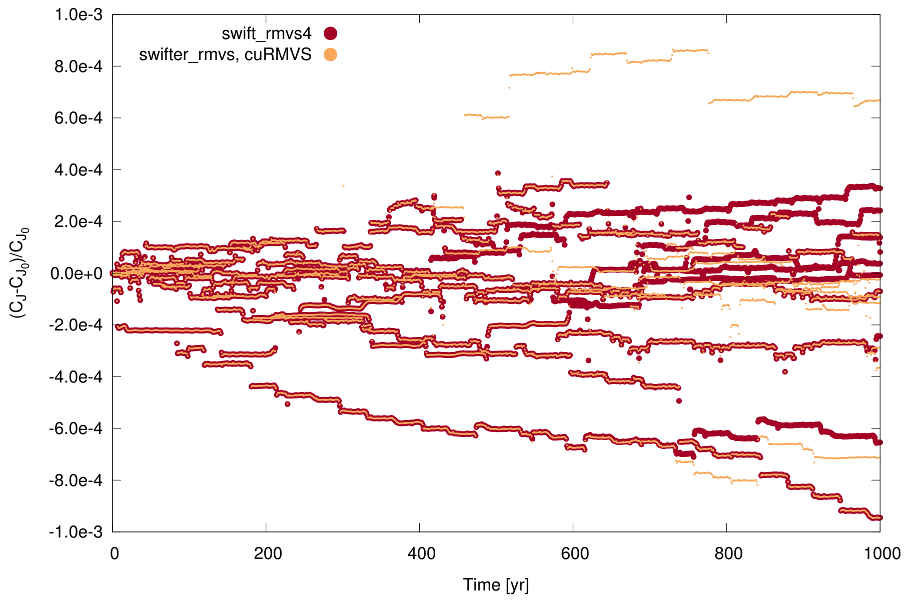

**Figure 5.5:** Fractional error in the Jacobi constant for each particle during 1 kyr of integration using cuRMVS with the Danby drift method and swift_rmvs4.

These tests show that even though it is not possible to exactly reproduce results achieved with swift_rmvs4, there are only minor differences between the tested implementations. For all runs, the fractional error in the Jacobi constant is nearly the same.

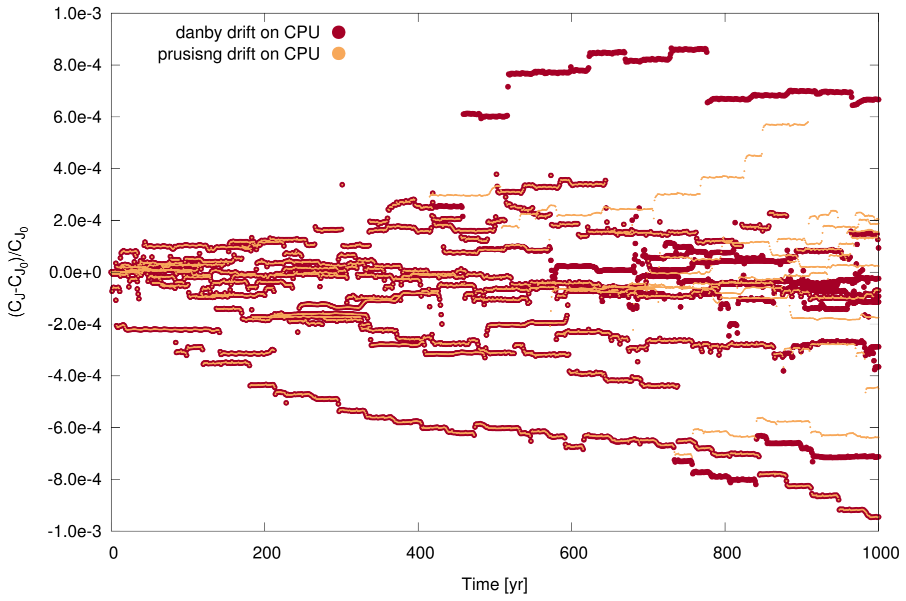

**Figure 5.6:** Same as Figure 5.5 but comparing the different drift methods on the CPU.

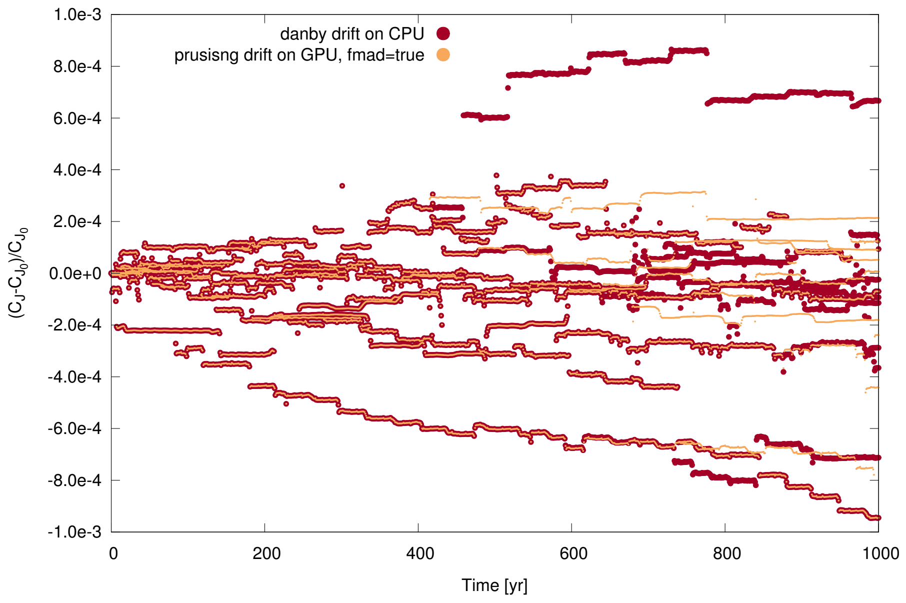

**Figure 5.7:** Same as Figure 5.5 but comparing the different computing platforms and optimizations.

## 5.3 Test setup for the Yarkovsky effect

For testing the implementation of the Yarkovsky effect, the following test setup was considered: Bodies of different sizes, thermal and rotational properties on circular orbits around the Sun with a semi-major axis of 2.25 au were integrated over 1 Myr using WHM plus the Yarkovsky effect with a time step of 1/100 yr. The change in semi-major axis was monitored and compared to the implementation of the Yarkovsky effect in SWIFT by Brož [12]. To exclude secular perturbation effects from the other planets which would alter the orbits of the particles and superimpose the Yarkovsky effect, no planets were included in the integrations.

The physical properties of the particles were chosen by analogy with those in Farinella and Vokrouhlický [41] who studied the influence of the Yarkovsky effect on asteroidal fragments. The results presented here however, differ from the those by Farinella and Vokrouhlický because they also included the effect of random reorientation of the bodys' spin axes due to collisions and integrated the particles over longer time scales. The spin axis was varied from 0° to 90° obliquity ($\gamma$) in 22.5° steps and the spin period (in seconds) was set to 5 times the bodys' radius (in meters). Thermal emissivity ($\varepsilon$) and Bond albedo were set to 0.9 and 0.05, respectively. For the material on the object's surface, a density of 1.3 g/cm$^3$ was assumed. The thermal properties were chosen such that they resemble 4 different types of objects: 3 stone-like asteroids differing in thermal conductivity only and a metal-rich class of higher density objects. The stone-like asteroids were given a bulk density ($\rho_{bulk}$) of 2.5 g/cm$^3$ . The specific heat capacity ($C$) of the surface material was set to 680 J/(kg K) and the thermal conductivity ($k$) was varied from 0.0015 to 0.015 W/(m K) . For the metal-rich objects, $\rho_{bulk} = 8.0$ g/cm$^3$ , $C = 500$ J/(kg K) and $k = 40$ W/(m K) was assumed.

### 5.3.1 Validating the Yarkovsky effect

As the Yarkovsky effect in cuSwift is implemented in the same way as in swift_mvs2_fp_ye_yorp by Brož, accuracy can be verified by comparing runs of the different integrators using the same initial conditions and parameters for the Yarkovsky effect. However, because Brož improved the WHM integration scheme in order to allow larger time steps and because several calculations in Brož' Yarkovsky force routines use single-precision by default, the results of swift_mvs2_fp_ye_yorp and cuSwift differ. In order to have a reference implementation against cuSwift can be tested, the code by Brož was modified such that the integrator uses the default WHM integration scheme. Further, variables declared as single-precision were changed to double-precision and the (`-fdefault-real-8`) gfortran option was set in the makefile. This ensures that constant expressions which are not explicitly marked as double-precision values and would thus be evaluated in single-precision are promoted to double-precision by default.

#### Conclusions

Figure 5.8 shows how the Yarkovsky effect changes the semi-major axis of the particles according to the reference implementation. Except for the metal-rich objects, smaller objects are more affected. The diurnal component of the effect dominates the low $k$ cases while the seasonal component is more important for mid-sized metal-rich objects. For

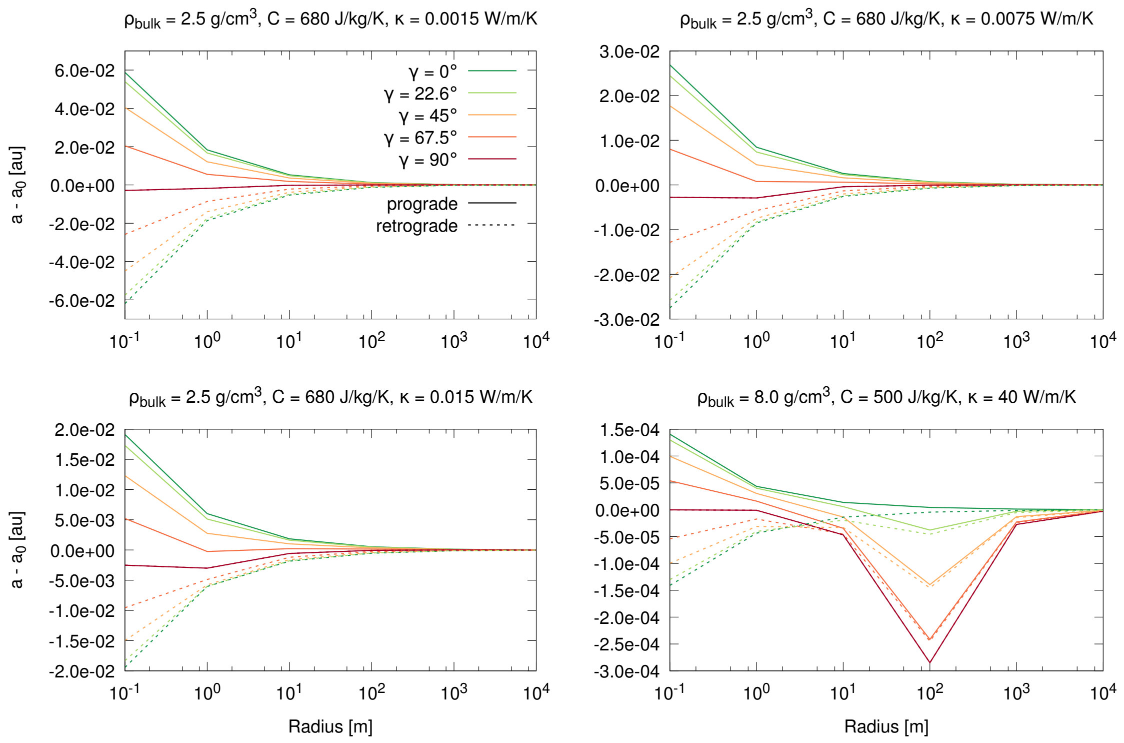

**Figure 5.8:** Change in semi-major axis caused by the Yarkovsky effect after 1 Myr of integration for different physical properties and obliquities using the modified version of swift by Miroslav Brož. The metal and high $k$ types are more affected by the seasonal component while for the low $k$ stony objects the diurnal component is dominant.

prograde rotators, the diurnal component leads to an increased semi-major axis while it causes retrograde rotators to spiral inwards. For the small metal-rich bodies, the seasonal effect vanishes as the seasonal thermal wave completely penetrates the objects, which leads to the heat being more evenly distributed over the whole surface and thus, temperature differences are vanishing. As expected, objects with large obliquities are also dominated by the seasonal component. In general, the metal-rich asteroids are, due to their high bulk density, affected to a smaller degree. For objects having a radius larger than 10 km, the effect is not measurable over the integration interval. Overall, the results are in good agreement with the approximations by Brož [13] and Bottke et al. [9].

The same experiment carried out with the CPU version of cuWHM plus the Yarkovsky force yields almost exactly the same results. For particles having a radius of 10 m or more, there is no difference between cuWHM and the reference implementation. For smaller particles, the relative error does not exceed 1%. This means that the changes to the Yarkovsky routines introduced with cuSwift do not significantly affect the results.

Compared to the method used for orbital integration, the routines for evaluating the Yarkovsky force make rather heavy use of mathematical functions whose GPU implementations do not fully comply to their CPU versions (see table 5.1). This causes the results obtained with CPU and GPU implementation to differ from each other. However, when comparing the GPU version to the reference implementation, it turned out that only high obliquity particles of 1 m radius or smaller are affected. The relative error for those particles is less than 2%.

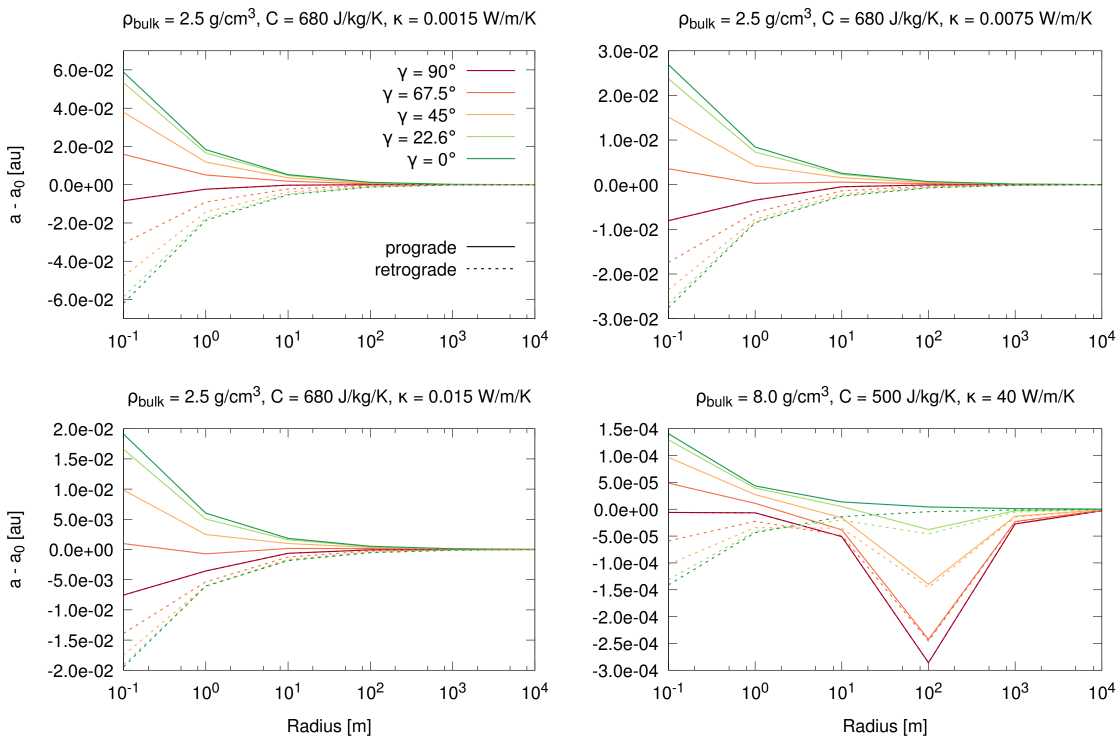

**Figure 5.9:** Same as Figure 5.8 but using the original implementation by Brož. Due to precision issues (see text) the seasonal component of the effect seems to be overestimated for smaller objects causing these bodies to drift inwards.

When comparing cuWHM with the original version by Brož (figure 5.9), some differences are visible: In the original implementation, the seasonal component seems to have a greater influence on the smaller objects. The differences can even better be perceived in Figure 5.10, where the relative error compared to the reference implementation is plotted. Especially for small objects, the seasonal component of the Yarkovsky effect should vanish, as the seasonal thermal wave fully penetrates the body. Depending on the thermal conductivity $k$, at 2.25 au, the seasonal thermal wave would penetrate 4 m into a high $k$ basaltic (bare rock) object and 0.6 m into a low $k$ regolith covered body [13]. For metal-rich objects which have $k$ values orders of magnitudes higher than the penetration depth would be even larger. This results in objects smaller than 1 m in radius should not be affected by the seasonal component as the seasonal wave heats up the whole body. The modified implementation by Brož and cuSwift comply with this assumption while the original implementation (figure 5.9) does not. The largest discrepancies between the two implementations occur for the 90° obliquity metal-like objects whose results differ from the reference implementation by up a factor of 30. For the stone-like objects, the relative error is about 2 for the high obliquity small particles. To examine the reason for these rather large differences between swift_mvs2_fp_ye_yorp and the reference implementation, additional tests have been performed.

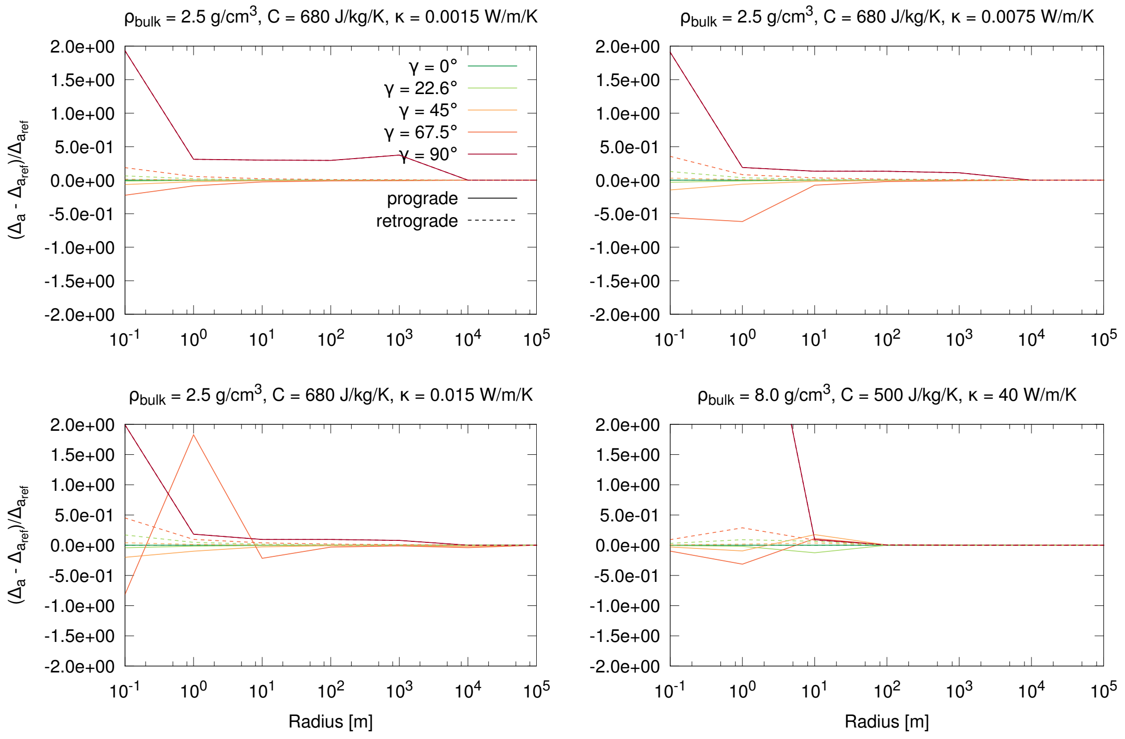

**Figure 5.10:** Relative error of the change in semi-mayor axis of the original implementation. In general, smaller objects are more affected. For the stone-like objects, the error grows up to a factor of two for objects smaller than 10 m in radius. For the 90° obliquity metal-like objects the relative error becomes as large as 30.

In the integration scheme used in swift_mvs2_fp_ye_yorp, the acceleration due to the Yarkovsky force is evaluated three times per time step while for WHM the acceleration is evaluated only once each time step. In order to determine if the error of swift_mvs2_fp_-ye_yorp is caused by round-off errors due to more calls of the Yarkovsky subroutines, the step size for MVS2 was increased by a factor of three. When switching off the Yarkovsky effect and using only the orbital integrators, the results are exactly the same for WHM and MVS2, even though the step size for MVS2 was tree times larger than for WHM. The increased step size for the MVS2 integrator results in the number of Yarkovsky force calculations over the integration interval is the same as for WHM. When enabling the Yarkovsky force, the error of swift_mvs2_fp_ye_yorp was still the same as when using the smaller time step, which means that the difference between swift_mvs2_fp_ye_yorp and cuWHM must be caused by something else.

As already explained, the Yarkovsky routines in swift_mvs2_fp_ye_yorp are partially employing single-precision arithmetic. This is caused by constant expressions in the code which are not explicitly marked as double-precision and thus are by default evaluated in single-precision only when compiling with the gfortran compiler. Additionally, there are variables for complex numbers which are explicitly declared in single-precision. After changing the makefile to promote constant expression to double-precision by default and changing the declaration of the complex numbers to double-precision, the results produced swift_mvs2_fp_ye_yorp and the reference implementation are identical. According to this test, in swift_mvs2_fp_ye_yorp, the seasonal component is rather heavily overestimated. When using the original version, it is recommended to change the code such that all calculations are performed in double precision.

## 5.4 Performance

The main goal of this work was to exploit the computing power of modern GPUs to significantly decrease the calculation times needed for modelling the long-term orbital evolution for minor bodies in the solar system. This section is intended to determine how much faster the newly implemented versions are, compared to the original. Therefore, runs using the same initial conditions and covering the same integration time interval with SWIFT and cuSwift have been carried out. The computer which was used for the test was equipped with an Intel Core i7-4930K CPU and a Nvidia GeForce GTX Titan Black GPU (see Section 3.3.1). The ratio between the execution time needed to cover a certain interval of integration time by the integration methods in SWIFT and cuSwift, also called speedup, is used as a measure to compare the different implementations and computing platforms. It is important to note that this is somewhat unfair, as pointed out by Lee et al. [63]. They studied several numerical methods which have been ported to the GPU and found that while the authors of the GPU implementation claimed a speedup of several orders of magnitudes over the CPU, after applying optimizations needed to fully utilize the compute capabilities of modern CPUs, the actual speedup decreases to an average value of only 2.5. SWIFT was developed during the 90s and thus is not optimized for modern CPUs which means that comparing the execution time needed for certain experiment by SWIFT on an up to date CPU with the execution time for the same experiment needed by cuSwift on the GPU does not tell much about the difference in computing power of CPU and GPU. However, the comparison is still relevant as the original implementation is actually used on modern CPUs. To better understand how much faster the GPU performs over the CPU, the parallel CPU version of cuSwift was also compared with the GPU implementation.

### 5.4.1 Test setup

A typical application for the integrators in cuSwift is to study the orbital evolution of minor bodies located close to or inside orbital resonances which are destabilizing their orbits. The results of these dynamical processes can easily be seen when looking at the distribution of asteroids in the main asteroid belt. Figure 5.11 shows the number of asteroids as a function of semi-major axis per $5 \cdot 10^{-4}$ au bin. Rather than being uniformly distributed, the main belt shows dips which are also known as Kirkwood gaps and which are related to mean motion resonances with Jupiter [114]. The strongest resonances are highlighted in the plot. Jupiter causes the eccentricity of asteroids, located in one of the resonances, to increase such that their orbits are crossing those of the planets which leads to close encounters and eventually to the ejection of the asteroids from the resonance region [79].

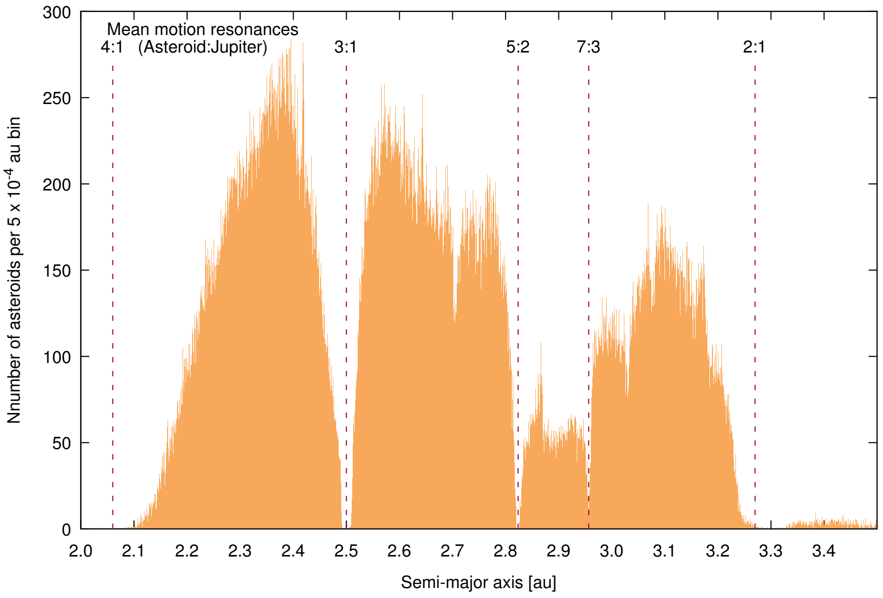

**Figure 5.11:** Number of asteroids in the main asteroid belt as a function of semi-major axis in $5 \cdot 10^{-4}$ au bins. The gaps can be related to mean motion resonances with Jupiter of which the strongest resonances are highlighted in the plot.

While the 4 Asteroid : 1 Jupiter (4A:1J) and 2A:1J mean motion resonances are marking the inner and outer boundary of the main asteroid belt, the 3A:1J resonance is the most effective resonance in destabilizing asteroids within the belt. The eccentricities for asteroids in or close to the 3A:1J resonance can suddenly increase to values up to 0.8 within only a few Myr [113] [121] which means that they do not only cross the orbit of Mars but also the orbits of Earth and Venus. Encounters within the RMVS extended Hill sphere of Jupiter can also occur.

To get a set of initial conditions for determining the speedup of cuSwift, the following scenario was considered: Semi-major axis (2.502 au), eccentricity (0.1366) and inclination (2.501°) of asteroid (27405) Danielfeeny, which is located inside the 3A:1J resonance, were chosen to create 32768 clones by setting the remaining orbital elements randomly distributed between 0 and 360°. The clones were then integrated with cuRMVS in a system of all planets form Earth to Neptune for 20 Myr using a step size of 1/100 yr. Because the particles were started right in the center of the resonance and their orbits were already slightly eccentric, they should encounter the inner planets very quickly. Figure 5.12 shows how semi-major axis and eccentricity of one of the particles develops with time. After about 200 kyr, the eccentricity reaches values large enough to cross the orbit of Mars and after 500 kyr encounters with Earth are possible. Due to a series of encounters with Earth and Mars, the semi-major axis for the particle is decreased. Eventually, it impacted Mars after 747 kyr.

While the WHM integrator, which does not resolve close encounters, will need more or less the same computation time for each time step, the RMVS method will slow down the

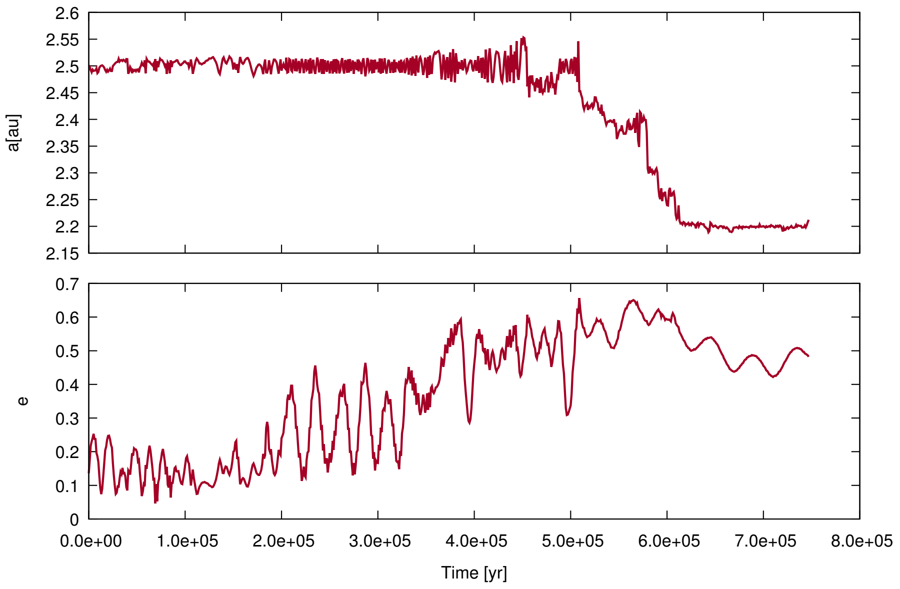

**Figure 5.12:** Semi-major axis and eccentricity for one of the particles used for this test. The eccentricity increases with time causing encounters with the inner planets. After 757 kyr the particle impacted on Mars.

more encounters are occurring per time step as close encounter resolution introduces extra work. To determine the performance of cuWHM and cuRMVS, 1 kyr of the computationally most demanding part (for the RMVS integrator) out of the 20 Myr run was selected, that is where the most encounters occur. In order to identify this part, a histogram of the number of encounters during the first 5 Myr (figure 5.13) was created. Positions and velocities from the bin with the most encounters (between 757 and 758 kyr) were extracted and used as initial conditions for the shorter 1 kyr run, which is then used to compare the execution times of cuSwift and SWIFT. The histogram only shows encounters occurring within the planets Hill spheres, as, due to memory and performance issues, cuRMVS only saves deep encounters and not those occurring in the extended Hill region. The total number of close encounters between 757 and 758 kyr, including the ones within the extended encounter region, was 4,676,460 which corresponds to an average number of 47 encounters per time step. Recording every encounter from the entire 20 Myr run would need several TB of disk space. Also, computations would dramatically slow down due to many disc accesses, which would be needed to write the data for each encounter.

Figure 5.13 also shows the CPU utilization during the integration. Note that 12 logical cores are available on the i7-4930K which means that a usage of 1200% corresponds to the CPU is fully utilized. The parameters for close encounter handling were set such that at least 15 encounters are stepped per CPU core (see Section 4.1.2). This means that during the time where the most encounters are occurring, 3 cores are used in average, resulting in a maximum CPU usage of ≈ 270%. As one core is used to integrate the planet positions, perform memory transfers between host and device and kernel execution, the CPU usage

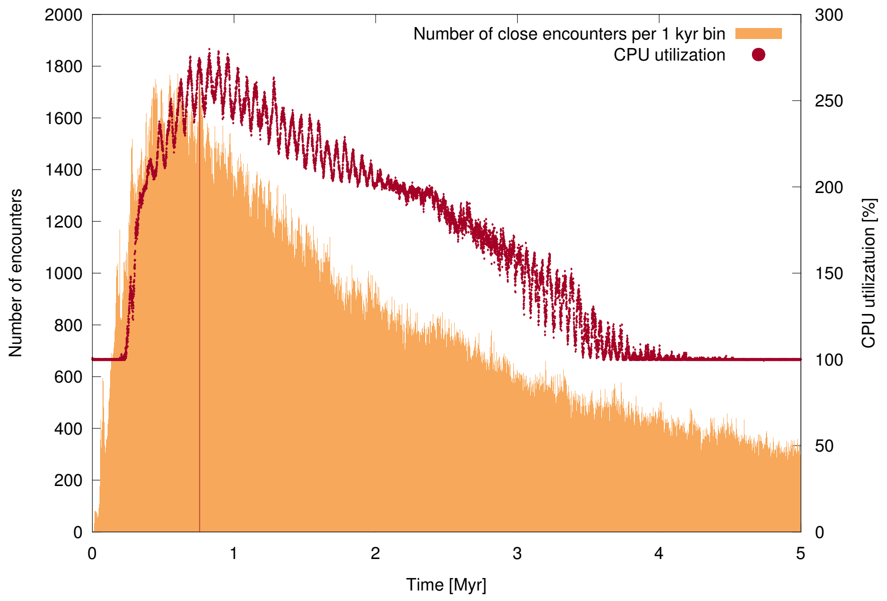

**Figure 5.13:** Number of close encounters during the first 5 Myr of integration in 1 kyr bins. The maximum number of close encounters occurred between 757 and 758 kyr. At this point the CPU usage was about 270% which means that 3 CPU cores were utilized.

never drops below 100%.

Already during the first 757 kyr of integration, several thousand particles were discarded. To create larger numbers of particles which are needed for the test, particles were randomly selected from those leftover after 757 kyr until a sufficient number was reached. For the integrations performed to determine the speedup between SWIFT and cuSwift, a time step of 1/100 yr was used. This results in $10^5$ time steps to cover 1 kyr of integration time.

## 5.4.2 Speedup

### WHM

The left panel of Figure 5.14 shows the calculation time needed for 1 kyr of integration as a function of the number of particles in the simulation for WHM in SWIFT (swift_mvs) and for different runs of cuWHM. The CPU runs show almost perfect linear behavior. If the number of particles is doubled, the execution time increases by a factor of two. The graph of the GPU run differs from the others. Execution time almost does not increase up to 4096 particles. For the first three data points, the relative growth in execution time is less than 10%, although the number of particles increases by a factor of 4. When more particles are included in the simulations, the slope of the curve gets steeper but does not reach a value of 2. The ratio of execution times needed for integrating 131072 and 65536 particles is 1.83. This can be explained by the GPU is not fully utilized when small numbers of particles are integrated. The GK110 chip has 2880 cores and can simultaneously run 30720 threads and as memory latency can be hidden more effectively using even more threads, the slope slowly converges to the value of 2 the more particles are involved.

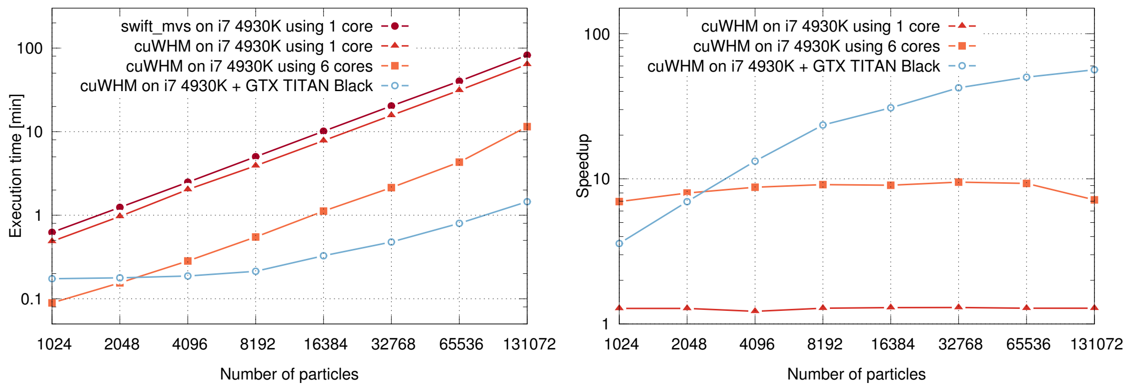

**Figure 5.14:** Left panel: Execution times for 1 kyr of integration with swift_mvs and cuWHM as a function of the number of particles integrated. The CPU runs show almost perfect linear behavior except the multi-core run. Execution time on the GPU almost does not change for small nubers of particles due to poor utilization. Right panel: Speedup of cuWHM over swift_mvs. Using one CPU core, cuWHM is slightly faster than swift_mvs. A speedup up to 9.5 is achieved using all cores. The GPU version performs up to 8 times better than the multi core version and up to 56 times better than swift_mvs for large numbers of particles.

An interesting feature can be seen for the multi-core CPU run of cuWHM: for 131072 particles, the slope gets steeper. A reason for this might be that the amount of data for the particles becomes so large that it can not be kept in the CPU cache while integrating and thus must be swapped to the RAM, introducing latency. The data needed for integrating a single particle is 185 bytes. Some of the data needs to be processed several times each time step. The run for 65336 particles consumes 11.6 MB and thus almost completely fills the 12 MB of last-level cache (LLC) available on the i7-4930K. For larger number of particles, memory must be swapped between CPU and RAM. To verify this hypothesis, the number of LLC loads and load-misses for a 100 yr run using the same initial conditions as for the speedup integrations has been monitored. A LLC-load-miss corresponds to an unsuccessful LLC access due to the requested data was swapped to the RAM. This means that each LLC-load-miss causes a memory transaction between RAM and CPU. Figure 5.15 shows the number of LLC-loads and LLC-load-misses as well as the ratio of misses per loads as a function of the number of particles for the single-core and multi-core version of cuWHM. Even small numbers of particles cause LCC-load-misses. This can be explained by although the computer was not used for any other task while performing the test, the CPU also needs to serve other processes while the integrator is running. This means that cuWHM can not consume the whole cache and thus, data needs to be swapped to the RAM even for runs using 65336 or fewer particles. When comparing the single-core with the multi-core runs it can be seen that for both versions the ratio of LLC-load-misses and the number total LLC-loads is similar. However, as the multi-core version runs about 10 times faster, it also demands higher memory bandwidth. This means means that the reason why the execution time for the multi-core version does not grow linear for runs including 65536 or more particles is that the memory bandwidth between CPU and RAM is insufficient to provide all cores with as fast as they would need it. The computer used for this work was equipped with RAM supporting only 1600 MHz front side bus (FSB) clock speed. As the i7-4930K supports up to 1866 MHz FSB clock speed, the number of load-misses might have a weaker impact when using faster memory. Processors equipped with larger LCC would be capable to integrate more particles without suffering from limited memory bandwidth. However, the problem will still be present for runs using even larger numbers of particles.

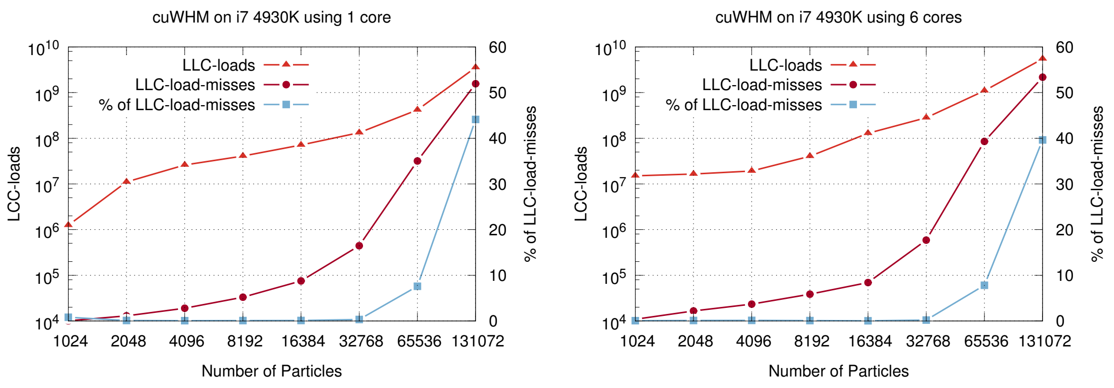

**Figure 5.15:** Number of LLC-loads and LLC-load-misses as well as the ratio between load-misses and loads after 100 yr of integration for the single-core (left panel) and multi-core (right panel) version of cuWHM. Growing numbers of load-misses cause non linear growth of execution time for large large number of particles.

When looking at the speedup of cuWHM over swift_mvs (figure 5.14, right panel) it can be seen that the newly implemented method using only a single CPU core beats the original implementation but is still comparable in speed. Using all available CPU cores, a speedup of almost a factor of 10 is achieved. For large numbers of particles the speedup decreases due to the previously described memory latency issues. The GPU version performs worse than the multi-core CPU version when using small numbers of particles but is 8 times faster when using 131072 particles. When comparing the GPU version of cuWHM to swift_mvs, a speedup of more than 50 is reached for large numbers of particles.

## RMVS

When performing the same experiment with RMVS (figure 5.16), it can be seen how close encounter handling slows down the simulation. For the same interval of integration time, the execution time of swift_rmvs4 increases of about a factor of 3, compared to swift_mvs. However, the execution times of cuRMVS on the CPU are less affected by close encounter handling. The relative growth in execution time of the single-core run and the multi-core run of cuRMVS and cuWHM is about 1.4 and 1.7, respectively. This suggests that the routines for processing the close encounters in cuRMVS are better implemented than in swift_rmvs4. The GPU version of cuRMVS is almost as fast as cuWHM on the GPU. For numbers of particles larger than 16384, the execution time is only 1.2 times longer for cuRMVS than for cuWHM. That is because close encounter processing in cuRMVS is performed in parallel on the CPU while the particles not involved in close encounters are stepped on the GPU (see section 4.1.2) and results in almost no extra time is needed for close encounter handling.

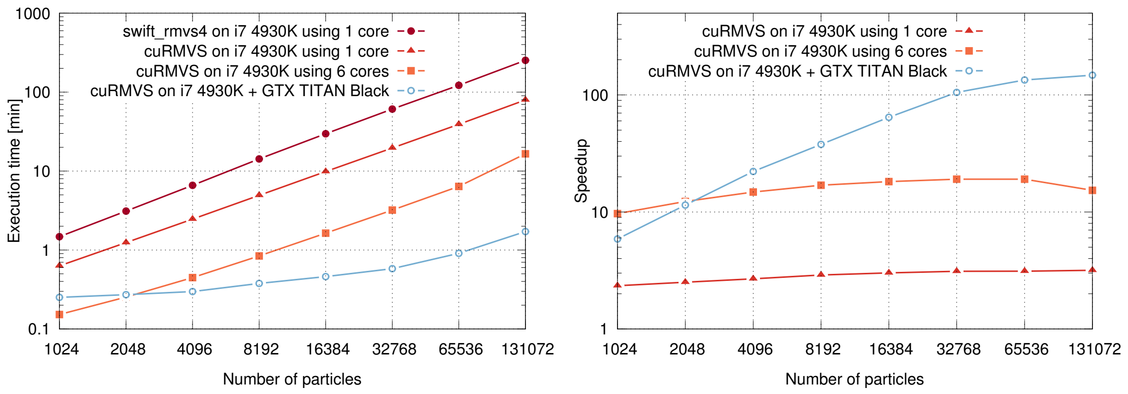

**Figure 5.16:** Left panel: Execution times of swift_rmvs4 and different runs of cuRMVS as a function of the number of particles integrated. Right panel: Speedup of cuRMVS over swift_rmvs4. Using one CPU core cuRMVS is up to 3 times faster than swift_rmvs4. A speedup of up to 20 is achieved using all cores. The GPU version performs up to 10 times better than the multi core version and 150 times better than swift_rmvs4 for large numbers of particles.

Figure 5.17 shows the CPU and GPU calculation as well as the memory transfer time-line for one time step out of the 65536 particles cuRMVS run. During this particular time step, 100 particles were involved in close encounters. Downloading the encountering particles to the CPU, close encounter handling and uploading the new positions, velocities and accelerations of the particles to the GPU is done in parallel while the particles not involved in close encounters are stepped on the GPU. The encounters were processed using 6 out of 12 available CPU threads which means that there are still reserves for processing more encounters on the CPU. Further, as, close encounter handling takes only about half the time the GPU needs for stepping the other particles, the execution time would not increase even when more encounters than in this test need to be resolved.

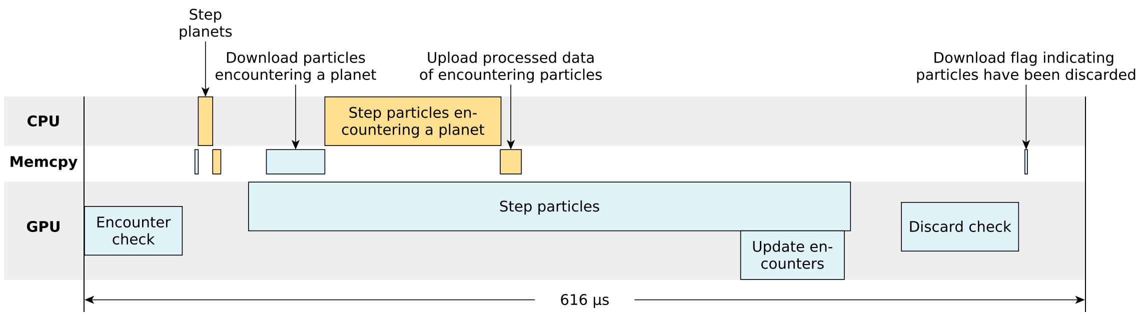
Step planets
Download particles encountering a planet
Upload processed data of encountering particles
Download flag indicating particles have been discarded
CPU
Memcpy
GPU
Step particles encountering a planet
Encounter check
Step particles
Update encounters
Discard check
616 µs

**Figure 5.17:** CPU and GPU time line for a single time step integrating 65536 particles. Close encounter handling is performed on the CPU in parallel to stepping the other particles on the GPU.

Due to parallel processing of encountering and non-encountering particles the speedup of the GPU version of cuRMVS over swift_rmvs4 is significantly higher than that of cuWHM over swift_mvs. Between the multi-core version and the GPU version almost a factor of

10 is reached. Compared to swift_rmvs4, the GPU version of cuRMVS runs up to 150 times faster.

The test setup was chosen such that many encounters per time step occurred and thus marks an upper limit for the speedup which can be achieved with cuRMVS. The speedup over swift_rmvs4 will drop if fewer encounters are occurring. Depending on the application, the actual speedup of cuRMVS over swift_rmvs4 will be at least as high as the speedup of cuWHM over swift_mvs.

### Conclusions

The test performed in this section clearly demonstrates that the newly implemented versions of WHM and RMVS in cuSwift are significantly faster than the reference implementation in SWIFT. To integrate the same number of particles with SWIFT in the same amount of time needed by the GPU version of cuSwift, the particles would need to be split into 50 to 150 chunks and integrations would need to be executed as separate tasks on a single processor core each. For large numbers of particles, up to 25 i7-4930K CPUs would be needed in order to achieve calculation times for SWIFT similar to those obtained with cuSwift on the GeForce GTX Titan Black. However, the disadvantage of splitting the integrations in several sub-processes is that integrating the planets is performed by each process. As during the integrations many particles may be discarded, the ratio of active particles and planets for each individual task decreases with time, which means that the computational overhead, arising from each process needs to integrate the planets, increases, making the over-all process less efficient.

When comparing the CPU version of cuSwift with the GPU version, the advantage of the GPU is considerably less. But still, the Titan Black outperforms the i7-4930K by up to a factor of 10. This factor may reduce when further optimizations like using the Intel Advanced Vector Extensions (AVX), which utilize special SIMD registers located on each CPU core, are applied. However, only runs using up to 65536 particles would benefit from these optimizations as for large numbers of particles, the memory bandwidth available on the CPU places constraints to the upper limit of the speedup which can be achieved. On the other hand, the CPU implementation runs faster for smaller numbers of particles as the GPU can only be fully utilized when large numbers of particles are considered. For the hardware used in this work, the GPU surpasses the CPU when more than about 3k particles are integrated. This number varies, depending on the hardware which is used. To determine which device should be used to perform a certain experiment, tests should be carried out in advance in order to find the faster device for the particular experiment and hardware.

The results from this test can also be used to estimate the time needed for longer integrations: For example, a 1 Gyr cuWHM run of 65336 particles using a time step of 1/10 yr would need abut 55 days on a GeForce GTX Titan Black. Also comparisons with science experiments actually performed can be made: To model the orbital distribution of Near-Earth Objects, Greenstreet et al. [48] performed integrations of about 62000 particles, initially located in unstable regions in the inner solar system. They used swift_rmvs4 with a time step of only 4 h to follow the particles until more than 99% of those having a perihelion distance of smaller than 1.3 au were discarded. The total integration time was

200 Myr which corresponded to as much as 300 core years of computation time using the fastest core available in 2009 [48]. As the 20 Myr run, used for setting up the benchmark for cuSwift, is very similar to the integrations performed by Greenstreet et al., it can be used to estimate the calculation time cuRMVS would need for performing the same experiment. Integrating 32768 particles for 20 Myr with cuRMVS took about 3.5 days. During the simulation, 97% of the particles were discarded. Compared to this run, Greenstreet et al. used 2 times as many particles and integrated them over an interval 10 times longer with a time step about 20 times shorter. This means that performing the same experiment with cuRMVS would take about 3.8 years on a single GeForce GTX Titan Black — certainly to long to use a single GPU only but still almost 80 times shorter than the fastest core available in 2009.

# CHAPTER 6

# Impact of the Yarkovsky effect on the Jupiter Trojan Asteroids

This chapter describes the first scientific experiment carried out with cuSwift. It is designed to study the effect of the Yarkovsky force on the Jupiter Trojan asteroids. The purpose of the experiment is to find out if and how the Yarkovsky effect influences long-term stability of the Jupiter Trojan asteroids. First, an introduction to the subjects of interest is given. Then the setup for the experiments is explained and finally the results are presented and discussed.

## 6.1 Jupiter's Trojan asteroids

Jupiter shares its orbit with a large number of asteroids which populate the regions around the $L_4$ and $L_5$ Lagrangian points in the Sun-Jupiter system (see Figure 6.1). These asteroids are in a 1:1 mean motion resonance with Jupiter and have similar orbital elements as the gas giant. The first of them, (588) Achilles, was discovered by Max Wolf in 1906. During the following years, more objects in the vicinity of the $L_4$ and $L_5$ Lagrangian points were found and Johann Palisa, who carried out many observations of these asteroids to better constrain their orbits, suggested to name those around $L_4$ after Greek heroes while those around $L_5$ should be named after heroes of Troy [89]. More recently, asteroids were also detected on the Lagrangian points of Mars [43], Earth [23] Uranus [1] and Neptune [101] which led to the term Trojan asteroid became a more general synonym for minor bodies in the 1:1 mean motion resonance with a planet. Even though five Lagrangian points exist the three-body problem, only $L_4$ and $L_5$ provide long-term stability. Up until now, Jupiter is the planet with the largest known population of Trojan asteroids. As of October 2017, the Minor Planet Center lists 6704 Trojans for Jupiter, 17 for Neptune, nine for Mars and one for Earth and Uranus, respectively. No Trojan asteroids have been discovered on the orbit of Saturn so far. While studies suggest that there are approximately as many Jupiter Trojans as main belt asteroids [119], it is not expected for the terrestrial planets to possess a large number of Trojans because the extent of the region around around their $L_4$ and $L_5$ Lagrangian points providing orbital stability is rather small. Also for Saturn and Uranus, only a few Trojans are expected as their $L_4$ and $L_5$ Lagrangian points are perturbed by secular resonances with the other giant planets and thus do not provide long-term stability [55][4]. Neptune, on the other hand, although not many have been detected so far, may host a very large population of Trojans [87] which might even outnumber the Jupiter Trojans.

It is still unclear how these objects were emplaced into their current orbits. Because planets clear their neighborhood during formation, it was not expected to find such large

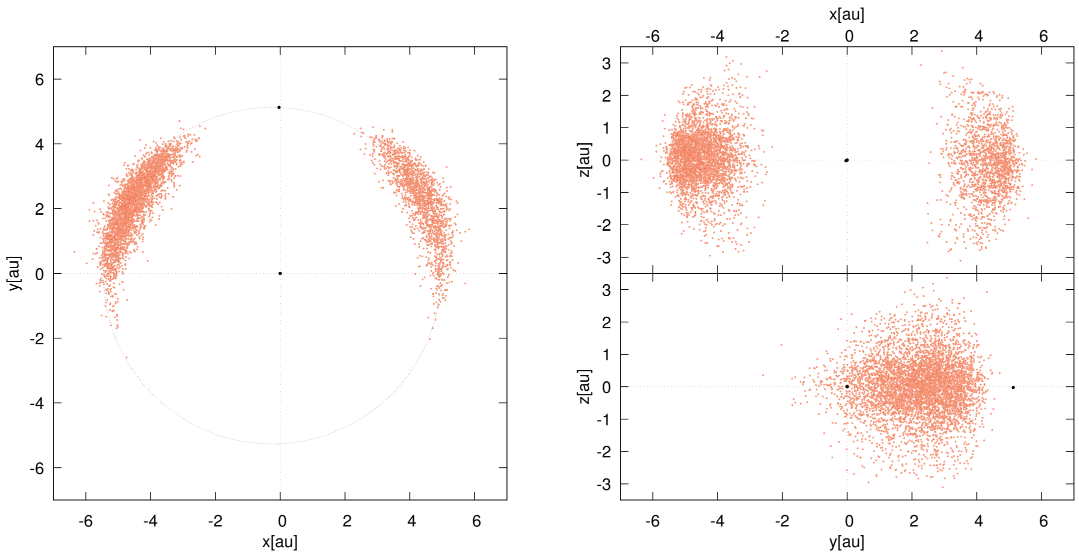

**Figure 6.1:** Positions of all known numbered and multi-opposition Jupiter Trojans in the heliocentric reference frame at epoch 2460816.5 JD. The leading swarm (around $L_4$) contains about twice as many objects than the trailing swarm (around $L_5$).

numbers of asteroids sharing a 1:1 mean motion resonance with Jupiter. It is also excluded that asteroids migrate to the Lagrangian points for example by the help of gas drag or collisions and get trapped in the resonance, producing a distribution of Trojans which matches the one currently observed [83]. Another peculiarity of the Jupiter Trojans is that both swarms differ in numbers. As of October 2017, the Minor Planet Center lists 4271 objects in the leading (around $L_4$) but only 2433 in the trailing cloud (around $L_5$). It is confirmed that this asymmetry is not only due to an observational bias. The ratio between $L_4$ and $L_5$ Trojans should lie between 1.2 and 1.8 [88].

Many analytic and numerical studies were carried out to shed light on the mysteries of the Jupiter Trojans. It was found that a large part of the currently known population is indeed stable over the age of the solar system [39] and detailed maps describing the stable regions around the Lagrangian points have been created [66] [106]. Although gravitational stability should be similar for both Lagrangian points [71], it was recently found in long-term integrations of the known Trojan population that the escape rate after 4.5 Gyr for $L_4$ Trojans is about 23% while it is about 28.3% for the $L_5$ swarm[32]. However, the difference in escape rate alone can not explain the asymmetry in the Trojan clouds which is observed today. A currently widely accepted explanation for the origin of Jupiter Trojans is that they were already present in their current orbits by the time the gas giants formed and were captured in resonance with Jupiter as it migrated to its current orbit [88]. This scenario is consistent with the famous *Nice model* [105] which describes the orbital evolution of the planets at the time of the early solar system. According to the Nice model, the migration of Jupiter took place rather quickly. The process of capturing the Jupiter Trojans is therefore also referred to as *jump capture*. The scenario describes the asymmetry between

$L_4$ and $L_5$ by an additional ice giant which, due to several close encounters with Jupiter, was eventually ejected from the solar system. During this phase of orbital chaos, the hypothetical ice giant may also have moved through one of the Trojan clouds wiping out a large number of objects.

Only very little is known about the physical properties of the Jupiter Trojans. Using infrared observations carried out with the Wide-field Infrared Survey Explorer (WISE) space telescope, a low mean geometric albedo of $0.07 \pm 0.03$ was determined for a sample of about 3000 objects [45]. Densities are known for two Jupiter Trojans only, namely for (624) Hektor and (617) Patroclus. Both objects are binaries, which allows a determination of mass and density by studying their mutual orbital configuration and size which can be derived for example from adaptive optics or radar observations. For (617) Patroclus, several studies report low bulk densities of values ranging from 0.8 g/cm$^3$ [69] to 1.3 g/cm$^3$ [75] with 0.88 g/cm$^3$ [14] being the most recent result. Densities found for (624) Hektor vary from 1.0 g/cm$^3$ [68] to 2.48 g/cm$^3$ [61]. Thermal properties are known for a few objects only. In general, it is expected that Jupiter Trojans have very low thermal inertia of about 5 J/(K m$^2$ s$^{1/2}$) [34]. Using thermophysical modelling from infrared observation data, a thermal inertia of 6 J/(K m$^2$ s$^{1/2}$) has been determined for (624) Hektor [51] while for (1173) Anchises a higher value in the range of 25 to 100 J/(K m$^2$ s$^{1/2}$) was reported [54]. By examining temperature changes on the surface of (617) Patroclus during shadowing events caused by its moon Menoetius, a thermal inertia of about 25 J/(K m$^2$ s$^{1/2}$) was determined [85].

The stability regions around the $L_4$ and $L_5$ Lagrangian points of the Sun-Jupiter system are well known. The main properties for determining the orbital stability of Jupiter Trojans are their libration amplitude $D$ which defines the maximum excursion in mean longitude $\lambda$

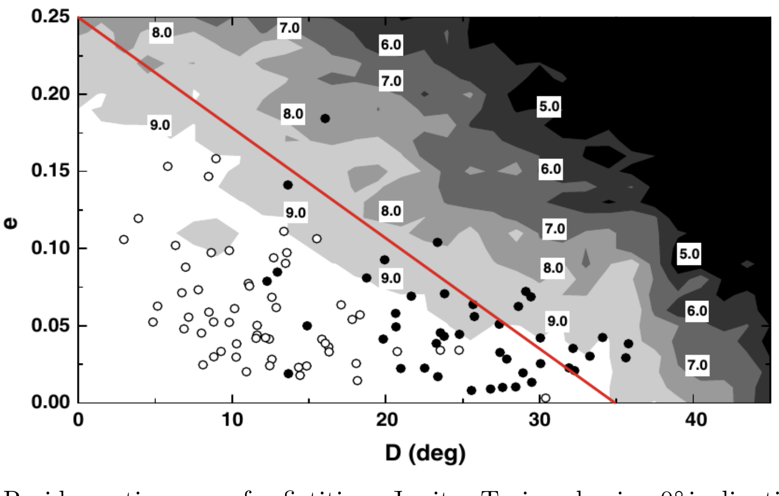

**Figure 6.2:** Residence time map for fictitious Jupiter Trojans having 0° inclination (modified from Tsiganis et al. [106]) Each color level corresponds to an order of magnitude of residence time in years. Open dots correspond to stable, closed dots to chaotic orbits of real Trojans.

with respect to Jupiter [39]

$$\lambda - \lambda_J = \pm\pi/3 + D\cos\theta + \mathcal{O}(D^2) \tag{6.1}$$

as well as their eccentricity and inclination. In 6.1, $\theta$ is related to the phase of libration. For $L_4$ Trojans, the difference in mean longitude is positive while it is negative for $L_5$ Trojans. The most recent study characterizing the residence time of objects around the Lagrangian points of Jupiter was done by Tsiganis et al. [106] in 2005. They used swift_mvs to integrate a synthetic population of Jupiter Trojans for 1 Gyr and monitored the escape time from the Lagrangian points as a function of libration amplitude, eccentricity and inclination. Figure 6.2 shows the expected residence times for fictitious Jupiter Trojans with 0° inclination computed by Tsiganis et al. In general, the residence time increases with decreasing libration amplitude and eccentricity. Inclination affects the long-term behavior to a lesser degree (see Figures 2-5 in Tsiganis et al. [106]).

## 6.2 Does the Yarkovsky force affect Jupiter Trojans?

As explained in the last section, many studies have been carried out to answer the questions about the origin and dynamical evolution of Jupiter Trojans. However, none of those studies incorporated the Yarkovsky effect. Some authors argue that due to the large heliocentric distance of Jupiter Trojans, the Yarkovsky force is too small to remove them from the stable region inside the resonance [119] [32]. Others concluded that at least for the smaller bodies, the Yarkovsky force becomes important for their long-term orbital evolution [106] [58]. This study is intended to answer the question as to whether the Yarkovsky force affects Jupiter Trojans and whether it is able to significantly influence their orbits on time scales of hundreds of Myr.

The first step to answer this question is to verify that the Yarkovsky force is strong

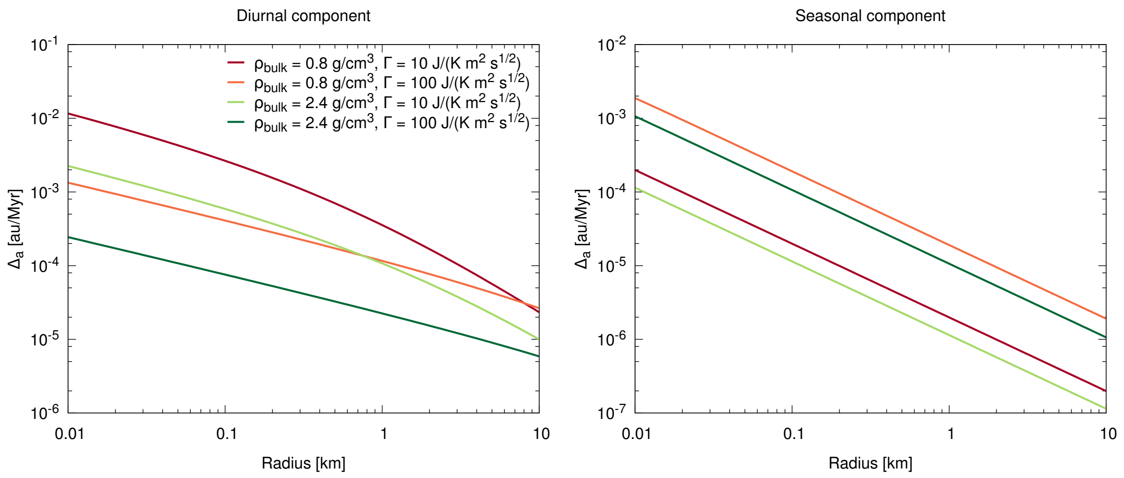

**Figure 6.3:** Upper bounds for semi-major axis drift rates dependent on the object radius for bodies in the Trojan region caused by the two components of the Yarkovsky force.

enough to produce a measurable effect on the Trojans at all. Using the physical properties derived for Trojans, an estimation of the drift rate in semi-major axis due to the two components of the Yarkovsky effect as a function of their size can be made [10]:

$$\left(\frac{\mathrm{d}a}{\mathrm{d}t}\right)_{diurnal} = -\frac{8\alpha}{9}\frac{\Phi}{n}F_\omega(R',\Theta)\cos\gamma + \mathcal{O}(e) \tag{6.2}$$

and

$$\left(\frac{\mathrm{d}a}{\mathrm{d}t}\right)_{seasonal} = \frac{4\alpha}{9}\frac{\Phi}{n}F_n(R',\Theta)\sin^2\gamma + \mathcal{O}(e). \tag{6.3}$$

$\alpha$ is related to the body's albedo as $\alpha = 1 - A$ with $A$ being a higher degree albedo, which for a rough approximation can be set to the Bond albedo [108], $\gamma$ is the obliquity of the body's spin axis and $\Phi$ represents the radiation pressure coefficient for spherical objects. The function $F(R',\Theta)$ is the same for both components of the Yarkovsky effect. The subscripts $\omega$ and $n$ are corresponding to the diurnal and orbital frequency, respectively. $F$ depends on $R'$, the Radius, scaled by the penetration depth of the thermal wave and the thermal parameter $\Theta$ and both are further depending on the corresponding frequency of the thermal wave (see Section 2.4). Orbital eccentricity (e) is neglected for this estimation.

Figure 6.3 shows how much the Yarkovsky force changes the semi-major axis of objects on a circular orbit at a distance of 5.2 au from the sun over 1 Myr for different thermal inertia and densities of the object. In order to get the largest possible drift rate, the obliquity for the diurnal and seasonal component were set to 0° and 90°, respectively. Physical properties of the objects were set to values expected for Jupiter Trojans. Because Trojans are assumed to have low densities and low thermal inertia, the drift rates are comparable to those of main belt objects. Assuming random obliquity, Bottke et al. [10] reported drift rates for asteroids in the main belt of about $1 \cdot 10^{-3}$ to $3 \cdot 10^{-3}$ au/Myr. For Trojans, changes in the semi-major axis have a strong influence on the libration amplitude. To quantify this influence, the libration amplitude in mean longitude ($D$, Equation 6.1) can be converted to the libration in semi-major axis ($d$)

$$a - a_J = d\sin\theta + \mathcal{O}(d^2) \tag{6.4}$$

using the approximation [39]

$$d = \sqrt{3\mu}\, a_j D \approx 0.2783D \tag{6.5}$$

where $\mu$ is the ratio of Jupiter's mass to the total mass of the system. According to the estimation of the drift rates due to the diurnal Yarkovsky effect (Figure 6.3), after 100 Myr, the maximum displacement in semi-major axis of a 100 m, low density Trojan is about 0.2 au. This would result in a change of libration amplitude of about 40° and therefore would significantly influence the dynamical lifetime of that object. However, gravitational interactions with Jupiter and the other planets also lead to secular changes in the orbital elements which might counteract the Yarkovsky effect. Consequently, it is necessary to perform long-term orbit integrations in order to evaluate both, gravitationally and Yarkovsky driven orbital evolution of the Trojans.

### 6.2.1 A Fictitious, long-lived population of Jupiter Trojans

In order to obtain a sufficiently large set of Jupiter Trojans with precisely known long-term survivability, a synthetic population of 16384 Trojans was created for each Trojan cloud. The limits for libration amplitude and eccentricity were chosen such that only orbits within the region which ensures a residence time of at least 100 Myr were generated, that is, below the red line in Figure 6.2. Inclination with respect to Jupiter was uniformly distributed between 0 and 30°.

This fictitious population was then used for a first test run to determine whether the Yarkovsky force does have an impact on the long-term orbital evolution of the Jupiter Trojans at all. The objects were integrated for 100 Myr with and without considering the Yarkovsky effect, with the rate of particles escaping the Trojan region being monitored for each run. The integrations were performed with cuWHM using a time step of 1/10 yr with all giant planets accounted for. Particles were considered as leaving the Trojan region and discarded if their heliocentric distances became less than 3 au, exceeded 8 au or if they approached Jupiter closer than twice the planet's Hill radius.

#### Without considering the Yarkovsky effect

Even though according to the way in which the population was constructed all particles should stay on Trojan orbits over the entire integration time span, during the 100 Myr of integration, 2% (674) of the 32768 particles left the Trojan region. Several explanations exist for this phenomenon. First of all, the residence time maps computed by Tsiganis et al. are based on statistical experiments which are inherently underlying fluctuations. Moreover, the libration amplitudes of the fictitious population were approximated by neglecting the higher order terms of Equation 6.1 and an approximation (Equation 6.5) was used to convert the libration in longitude ($D$) to the libration in semi-major axis ($d$) for creating the fictitious population. Finally, the condition used here for a particle escaping the Trojan region was a bit more restrictive than the one used by Tsiganis et al. They considered a particle leaving the Trojan region only if its distance to Jupiter becomes less than two Hill radii but did not use limits in heliocentric distance in order to examine also the high eccentricity regime. However, as highly eccentric Trojan orbits are not long-lived and would leave the Trojan cloud after a short period of time anyway, those particles can be discarded for this experiment.

Figure 6.4 shows the initial values for eccentricity and inclination over libration amplitude of all particles. The colored dots refer to particles which escaped from the Trojan region during the 100 Myr of integration. As expected, orbits with a higher eccentricity are more likely to be removed. Further, because the size of the region providing orbital stability tends to decrease with growing inclination, most of the ejected particles have high inclinations. The rate of escaping particles is about the same for both Lagrangian points (see left panel of Figure 6.5).

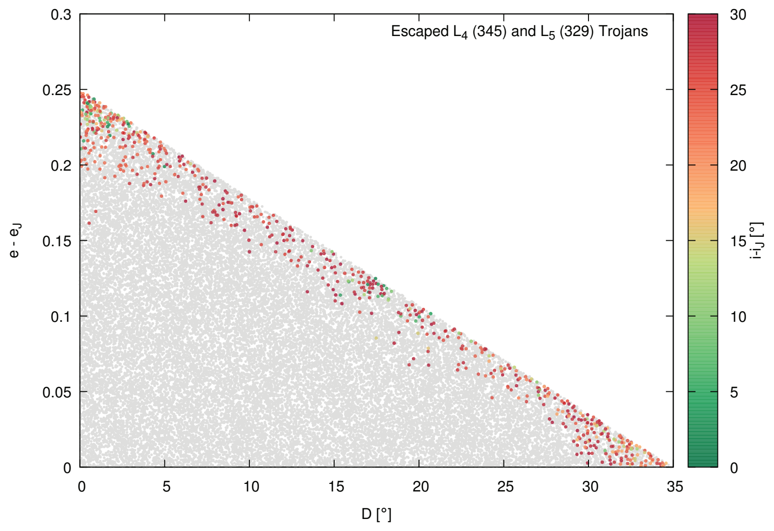

**Figure 6.4:** Initial conditions for the synthetic population of Jupiter Trojans. The gray dots refer to particles which remain inside the Trojan region for 100 Myr of integration (without considering the Yarkovsky effect) while the colored dots indicate ejected particles. Orbits having higher eccentricities and inclinations are more likely to escape the Trojan clouds.

### When considering the Yarkovsky effect

To see how the Yarkovsky force affects the long-term evolution of the fictitious population, the objects were given thermal properties leading to a strong Yarkovsky effect but were still within the range expected for Jupiter Trojans. The bulk density was set to 0.8 g/cm$^3$ and a low thermal inertia of 10 J/(K m$^2$ s$^{1/2}$) was chosen for the surface material. All objects were 10 m in radius and, in order to have a strong diurnal component of the Yarkovsky effect, their rotation periods were set to 150 s, about three times as long as can be assumed for objects of this size [41]. The particle's spin axis orientation was uniformly distributed over the celestial sphere, resulting in an equal ratio of prograde and retrograde rotators. Bolometric Bond albedo and infrared emissivity were set to 0.027 and 0.9, respectively [45]. Using these settings for the Yarkovsky effect, the orbits of the same fictitious population as used for the non-Yarkovsky run were integrated for 100 Myr.

Compared to the non-Yarkovsky run, the number of escaping particles increased by a factor of 1.7 (see left panel of Figure 6.5). The initial conditions of the escaped particles in D-e-i space (right panel of Figure 6.5) reveal that, in contrast to the non-Yarkovsky run, objects initially located deeper inside the region providing long lifetimes were also affected. This means that the Yarkovsky effect acts as a transport mechanism driving the particles to regions providing shorter residence times. Another interesting fact is that inclination seems to be more affected than eccentricity. While the average eccentricity of the ejected particles in both runs was 0.130, the average inclination decreased from 23.29° to 20.73° when incorporating the Yarkovsky effect.

This comparison clearly shows that the Yarkovsky effect can destabilize small Jupiter Trojans and causes more of them to leave the Trojan region. However, size, rotational and

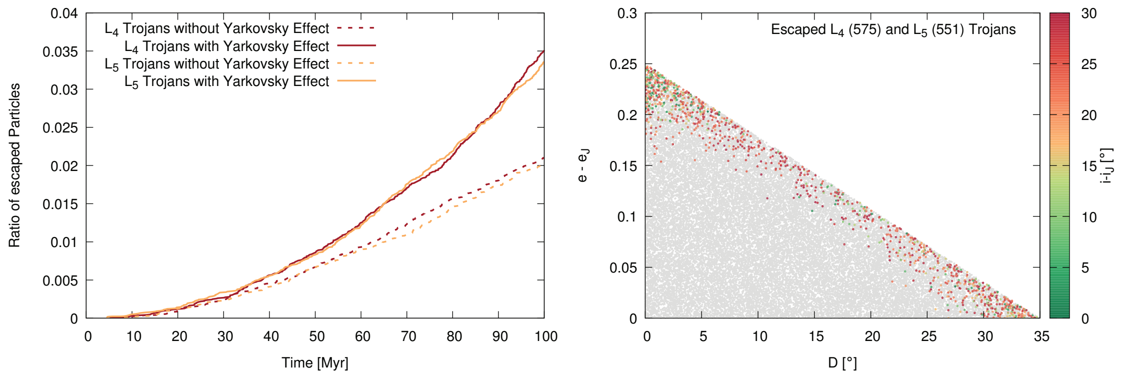

**Figure 6.5:** Left panel: Cumulative number of particles escaping the Trojan region around $L_4$ and $L_5$ over 100 Myr of integration with and without considering the Yarkovsky effect. Right panel: Same as Figure 6.4 for the Yarkovsky run. Compared to the non-Yarkovsky run, particles initially located deeper inside the resonance region were also affected.

physical properties of the particles in this first run were intentionally chosen such that they resulted in a very strong Yarkovsky force. To see how the effect acts on the real Trojans, a more realistic setup needed to be considered.

### 6.2.2 A more realistic population

After it was verified that the Yarkovsky effect can indeed influence Trojan orbits, the real population of Jupiter Trojans was studied. However, the population of currently known objects would be an insufficient sample for this experiment. Also, for some objects only very few observations from a single opposition are available, which means that their orbits may not be known with sufficient accuracy. Therefore, when investigating long-term evolution, only the orbital elements of the numbered and unnumbered multi-opposition Trojans should be used, which makes the set of available orbits even smaller. In order to create a sufficiently large sample of objects for each cloud, the orbits of the currently observed Trojans were cloned by varying the orbital elements. Because especially for Trojans, the niches in orbital elements space which provide long-term stability are very narrow, it has to be ensured that cloning does not change the orbits long-term dynamical behavior. On that account, the variations in orbital elements for each clone were computed based on the accuracy of the astrometric positions from which the orbit of the real object is calculated. Each individual observation contains an error which then propagates into uncertainties in the resulting orbit. These uncertainties are published in form of a covariance matrix for each asteroid on the AstDyS website[1]. Using that covariance matrix, the uncertainty $\varDelta\mathbf{q}$ in the orbital elements vector $\mathbf{q}$ can be expressed as

$$\varDelta\mathbf{q} = \sum_{i=1}^{6} \xi_i \sqrt{\lambda_i}\mathbf{X}_i, \tag{6.6}$$

1 http://hamilton.dm.unipi.it/astdys2/index.php?pc=4

where $\xi_i$ are Gaussian distributed random numbers with a standard deviation of unity and zero mean, $\lambda_i$ are the eigenvalues of the covariance matrix and $\mathbf{X}_i$ are the normalized eigenvectors. This cloning technique was used to generate a total number of 32768 Trojans. Because only precisely determined orbits are chosen for cloning and each clone is a realization of its parent orbit within the $1\sigma$ range of the observational data, the long-term orbital evolution of the cloned population should be the same as for the real one.

The clones were generated from the known numbered and multi-opposition Trojan orbits as of August 2017, consisting of 3642 $L_4$ and 1925 $L_5$ objects. The cloned population then contains the real orbits plus four to five clones for $L_4$ Trojans and seven to eight clones for $L_5$ Trojans, respectively. The sub-populations of the two clouds were further divided into 16 groups of particles having different physical properties and sizes each. The radii were set to 10 m, 100 m, 1 km and 10 km and four different categories of physical properties which are the parameters for the Yarkovsky force were introduced. The upper and lower limits of the values which have actually been reported for the physical properties of Trojans (see Table 6.1) where chosen to define the categories. This subdivision resulted in 1024 particles for each category, size and cloud. The spin axis orientation was uniformly distributed over the celestial sphere and the spin period (in seconds) was set to 5 times the body's radius (in meters) [41]. Bolometric Bond albedo and infrared emissivity for all objects were 0.027 and 0.9, respectively.

In order to determine whether the Yarkovsky effect influences the long-term orbital behavior of this population, again a run without and a run with considering the Yarkovsky effect was performed.

### Estimating the libration amplitude

As already explained, the libration amplitude is an important indicator for the lifetime of Trojans. Hence, it would be interesting to see how the Yarkovsky effect influences the libration amplitudes of the cloned orbits. However, in contrast to the fictitious population where the libration amplitude was a parameter for creating the orbital elements, it is unknown for the objects of the cloned population used for this experiment. In order to analyze the results, the libration amplitude for each object needs to be determined. A convenient way for doing this is to apply Frequency map analysis (FMA) on the output of the numerical integration to identify periodic variations in the orbital elements and the resonant argument [73]. For numbered and multi-opposition Jupiter Trojans this is frequently done and the latest set of proper elements and libration amplitudes, computed by

**Table 6.1:** Combinations of physical properties used for the Yarkovsky effect.

| Category | $\rho$ [g/cm$^3$] | $\Gamma$ [J/(K m$^2$ s$^{1/2}$)] | $A_{bond}$ | $\varepsilon$ |
|---|---|---|---|---|
| 1 | 0.8 | 10 | 0.027 | 0.9 |
| 2 | 0.8 | 100 | | |
| 3 | 2.5 | 10 | | |
| 4 | 2.5 | 100 | | |

Knezevic and Milani in June 2017, is available at the AstDyS website[1]. For the population of cloned Jupiter Trojans used for this work however, the libration amplitudes are unknown. The procedure of computing high precision proper elements which are needed to compute libration frequencies and amplitudes is rather complicated and time-consuming [77]. In order to get a good estimation of the libration amplitude for the cloned objects used in this work, a less sophisticated but much faster approach, similar to the one introduced by Marzari et al. [73], was employed. The libration amplitudes were calculated from a shorter 200 kyr integration. To remove the short-periodic variations from the orbital elements which would superimpose the libration, a digital low-pass filter was used. The left panel of Figure 6.6 shows the parameters and the response of the filter. The parameters were chosen such that periods shorter than the libration period lie in the stop band while the libration period (around 145 years or longer) lies in the pass band and is very little affected by the filter. Using the filtered elements, the libration in semi-major axis is computed as the mean of the maximum excursion in semi-major axis ($d$) in 2 kyr intervals over the 200 kyr of integration. Finally, the libration amplitude $D$ is calculated using Equation 6.5.

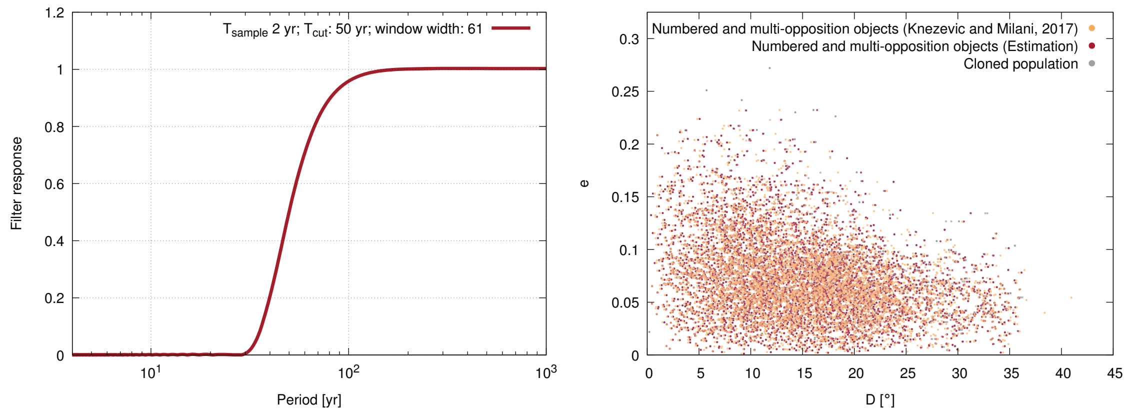

**Figure 6.6:** Left panel: Response of the filter used to eliminate short periodic perturbations from the orbital elements. Right panel: Libration amplitudes for numbered and multi-opposition Jupiter Trojans published by Knezevic and Milani (orange dots) and computed with the method used in this work (red dots). The results are in good agreement, except for a few objects having a large libration amplitude or period. The gray dots represent the libration amplitudes for the population of clones.

In order to test this procedure, the libration amplitudes of the numbered and multi-opposition Trojans were calculated and compared with the values available on the AstDyS website. The right panel of Figure 6.6 shows the libration amplitudes for the objects published at the AstDyS website and those calculated using the method described above. The estimated amplitudes are in good agreement with the ones calculated by Knezevic and Milani. The standard deviation is 0.6°and only a few objects with very large libration amplitudes or very long libration periods show larger errors. The gray dots in Figure 6.6 are representing the libration amplitudes for the cloned population created for this

1 http://hamilton.dm.unipi.it/~astdys2/propsynth/tro.syn

work. Most of them cluster around the real objects (red dots) as the differences in orbital elements between the parent body and clones are very small. However, there are also clones who appear to have no corresponding parent body. This is because the same population of objects as the one used by Knezevic and Milani was used to verify the procedure of calculating the libration amplitude but a more recent population was used to generate the clones. Because there are more numbered and multi-opposition objects in August 2017 then there were in June 2017, not all parent bodies for the cloned population are represented in the plot. It can be seen that in general the distribution of clones in e-D space matches the known orbits very well and thus the cloned population should have very similar long-term orbital behavior.

### Finding an appropriate integration interval

On the one hand, under the gravitational influence alone, the orbits of a large number of Jupiter Trojan asteroids can survive the age of the solar system (about 4.5 Gyr). This means that for studying the real population, an integration interval larger than that of 100 Myr used for the fictitious population should be considered. On the other hand, the lifetime of asteroids is also limited by mutual collisions which are not resolved in cuSwift. Thus, when choosing an integration interval it has to be ensured the objects can sustain the total simulation time and won't be destroyed by mutual collisions. In order to find a reasonable value for the integration interval, the collisional lifetime of the Jupiter Trojans has to be estimated.

The collisional lifetime of a minor body is defined as the mean time until the object is destroyed by a catastrophic disruption, defined as a collision with the size of the largest remnant being less or equal to half the size of the original body [46]. The important parameters for the collisional lifetime are size and density of the colliding bodies as well as the intrinsic collision probability $P_i$, impact velocity $v_i$ and size distribution of the collisional interacting population. While the bodies' physical parameters and impact velocity are needed to determine $Q_D^*$, the energy per unit mass causing a catastrophic disruption of the target body, the intrinsic collision probability which defines the flux of impactors per area and time can be combined with the objects size distribution to estimate how frequently catastrophic disruption events occur.

The average impact velocities and the intrinsic collision probability for Jupiter Trojans have been determined in several studies [72][31]. However, providing a good estimation for $Q_D^*$ is not easy. By simulating collisions using hydrocodes and performing high velocity impact experiments, several methods for modelling $Q_D^*$ were introduced. The most recent study for finding an appropriate value of $Q_D^*$ for Jupiter Trojans was carried out by Wong et al. [116]. They simulated the collisional evolution of the Jupiter Trojans by adjusting the size dependent scaling law for $Q_D^*$ of main belt asteroids proposed by Durda et al. [36] in order to fit it to the low diameter end of the debiased size-frequency distribution of Trojans down to about 10 km. As in this work much smaller objects are studied, this approach is not applicable. Instead, a scaling law introduced by Benz and Asphaug [5] which is valid down to cm sized objects is used to determine $Q^*_D$:

$$Q^*_D = Q_0 \left(\frac{R_{tar}}{1\,\mathrm{cm}}\right)^a + B\rho \left(\frac{R_{tar}}{1\,\mathrm{cm}}\right)^b, \tag{6.7}$$

where $R_{tar}$ and $\rho$ denote radius and density of the target body, respectively. The advantage of this model is that it is capable of describing $Q^*_D$ in the strength regime (small objects) where fragmentation of the target body is the dominant factor as well as in the gravity dominated regime (large objects) where the target body has to be fragmented and the fragments must also be dispersed in order to cause a catastrophic disruption. Benz and Asphaug used smooth particle hydrodynamics (SPH) to simulate collisions of different materials and impact velocities and provided $Q^*_D$ fits for each parameter set. For icy material and an impact velocity of $v_i = 3$ km/s, the conditions closest to those in this work, the remaining parameters in Equation 6.7 are $Q_0 = 1.6$ J/g, B = 1.2 $\cdot 10^{-7}$ J cm$^3$/g$^2$, a = -0.39 and b = 1.26. To obtain a lower limit for the collisional lifetime, the density of the weakest bodies (0.8 g/cm$^3$) was used to evaluate $Q^*_D$. Knowing $Q^*_D$, the radius for an impactor $R_{dis}$, capable of disrupting a target body of a certain radius $R_{tar}$ can be calculated as [11]

$$R_{dis} = \left(\frac{2Q^*_D}{v_i^2}\right)^{1/3} R_{tar}. \tag{6.8}$$

Finally, given the size distribution of the collisionally interacting bodies, the collisional lifetime ($\tau_{dis}$) of an object with radius $R_{tar}$ is [42]

$$\tau_{dis} = \frac{1}{P_i R_{tar}^2 N(> R_{dis})}, \tag{6.9}$$

where $N(> R_{dis})$ is the number of objects with radii $R > R_{dis}$. $P_i$ was set to the highest value reported, which is 7.79 $10^{-18}$ yr$^{-1}$km$^{-2}$ for $L_4$ vs $L_4$ Trojans [28]. Other populations like short period comets or Hildas also collisionally interact with Jupiter Trojans [30]. However, the intrinsic impact probability between these populations and the Trojans is orders of magnitudes lower than the Trojan's mutual impact probability and thus their influence on the collisional lifetime of Trojans is negligible for this experiment.

Unfortunately, there is only very little knowledge about the size distribution of $R < 10$km Jupiter Trojans which is needed to calculate $N(\geq R_{dis})$. Usually, the size distribution of asteroids can be expressed in form of a power-law distribution:

$$N(> R) = C \cdot R^b, \tag{6.10}$$

where $N(> R)$ is the cumulative number of asteroids with radii $> R$, $C$ is constant and $b$ denotes the slope of the cumulative size distribution. By observing and debiasing $L_4$ Trojans using the 2.2 m telescope at the University of Hawaii, Jewitt et al. [57] found the size distribution of objects larger than 2.2 km in radius follows a broken power law.

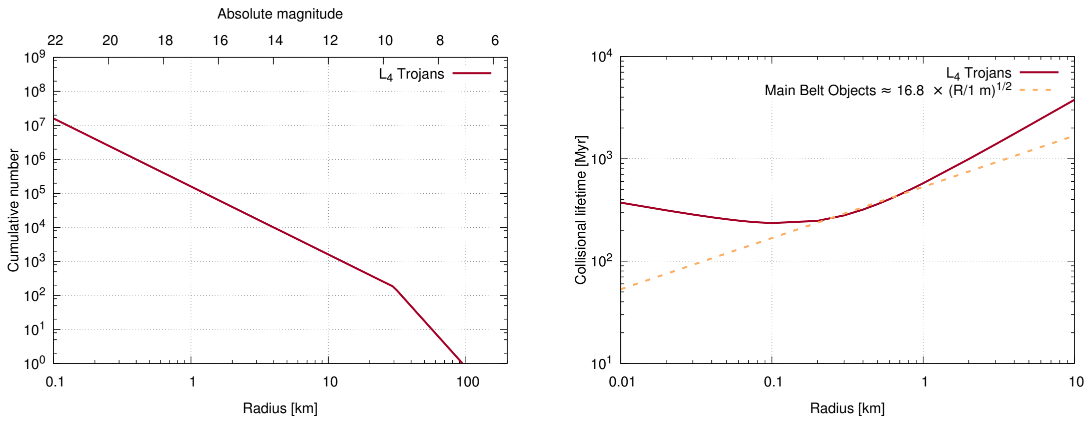

**Figure 6.7:** Left panel: Cumulative size distribution of $L_4$ Trojans according to Jewitt et al. [57]. Right panel: Collisional lifetime of $L_4$ as well as main belt objects.

According to their work, the cumulative size distribution of $L_4$ Trojans is

$$N(> R) = 1.6 \cdot 10^5 \left(\frac{1\,\mathrm{km}}{R}\right)^{2.0\pm0.3} \tag{6.11}$$

for $2.2\,\mathrm{km} \leq R \leq 20\,\mathrm{km}$ and

$$N(> R) = 7.8 \cdot 10^8 \left(\frac{1\,\mathrm{km}}{R}\right)^{4.5\pm0.9} \tag{6.12}$$

for $R \geq 42\,\mathrm{km}$ objects. For smaller Trojans, very large telescopes are required for detection. The latest approach to estimate the size distribution of small Trojans was done by Yoshida and Nakamura [119][118]. They used Trojan asteroids serendipitously detected within the Subaru main belt asteroid survey, performed with an 8.2 m telescope to determine their size distribution down to $R \approx 1\,\mathrm{km}$. For $1.0\,\mathrm{km} \leq R \leq 2.5\,\mathrm{km}$ $L_4$ Trojans they found a shallower power-law slope of $1.3 \pm 0.1$ while objects with $2.5\,\mathrm{km} \leq R \leq 5.0\,\mathrm{km}$ showed a slope of $2.4 \pm 0.1$, a value similar to the one reported by Jewitt et al. It is well known that collisional cascades can introduce variations in the power-law slopes [38] which means that it is not clear the slope found by Yoshida and Nakamura is representative down to $R < 1\,\mathrm{m}$ objects which are considered in this experiment (see Table 6.2). For this reason, the size distribution found by Jewitt et al. is used to calculate $N(> R_{tar})$ and $N(> R_{dis})$, although it is based on observations of much larger objects.

Putting it all together, an estimation of the collisional lifetime of Jupiter Trojans can be made. For the size ranges considered in this experiment, the collisional lifetimes are given in Table 6.2. Figure 6.7 shows the cumulative size distribution used to estimate the number of targets and impactors as well as the resulting estimation for the collisional lifetime of Trojans and main belt objects [41] for comparison. Although this estimation is calculated based on the size distribution of $L_4$ Trojans, it represents a lower limit for the collisional lifetime of Jupiter Trojans in general, as the $L_5$ Trojans are expected to be fewer in numbers and thus should have even longer collisional lifetimes than $L_4$ Trojans. As can be seen in Figure 6.7, the collisional lifetime for Trojans considered in this experiment ranges from several 100 Myr up to several Gyr. On that account, the integration interval for investigating the influence of the Yarkovsky effect was set to 500 Myr.

**Table 6.2:** Radii of Target and Projectile capable of causing a catastrophic disruption of the target body as well as the number projectiles in the $L_4$ cloud and the resulting collisional lifetime for the target body.

| $R_{tar}$ [m] | $R_{dis\,L_4}$ [m] | $N(R_{dis})_{L_4}$ | $\tau L_4$ [Myr] |
|---|---|---|---|
| 10 | 0.22 | $3.44 \cdot 10^{12}$ | $3.72 \cdot 10^{2}$ |
| 100 | 1.71 | $5.45 \cdot 10^{10}$ | $2.75 \cdot 10^{2}$ |
| 1000 | 26.8 | $2.22 \cdot 10^{8}$ | $5.77 \cdot 10^{2}$ |
| 10000 | 685 | $3.41 \cdot 10^{5}$ | $4.39 \cdot 10^{3}$ |

It is important to note that the collisional lifetime of small Trojans calculated by de Elía and Brunini [38] is significantly shorter than the estimation presented here. Their results however correspond to a primordial population of Trojans which they assumed to be orders of magnitudes larger than the current one. Their aim was to reproduce the population which is observable today and to find out down to which size the Trojans were not significantly altered by collisions since the formation of the solar system. They also placed their initial population inside the dynamically stable regions of the resonance. Assuming the jump capture scenario for the origin of Jupiter Trojans, there is no reason to assume that the primordial population occupied the stable regions only. This means that the dynamical lifetime of many objects of the primordial population may be significantly shorter, resulting in the collisional lifetime of the remaining long-time stable part of the population is increased.

### Run time of the integrations

The main integrations of 32768 orbits over 500 Myr without and with considering the Yarkovsky effect for this work were performed with cuWHM on a PC equipped with two Nvidia GeForce GTX Titan Black GPUs. The simulation took about 20 days in total. The shorter 200 kyr integrations for estimating the libration amplitudes took as few as 1.5 hours on a Laptop equipped with a quad core 2.7 GHz Intel i7-6820HQ CPU. For comparison, in the numerical experiment for computing the libration amplitudes of Trojans, performed by Marzari et al. in 2003, 1000 fictitious Trojans were integrated with swift_mvs for 2.5 Myr [73]. They reported the integrations took about four days on a computer equipped with a 2 GHz Intel Pentium 4 processor. Taking these values to estimate the run time for simulating $\approx$65k particles over 500 Myr in 2003 yields about 140 yr, 2600 times longer than the computation time for the simulations performed in this work. This means that the hardware improvements achieved within a decade led to the computation time for the same algorithm to be decreased by more than three orders of magnitude.

## 6.3 Results and discussion

Figure 6.8 shows the ratio of particles escaping from the Trojan clouds during the simulation. The transparent areas represent the 1 $\sigma$ confidence region, assuming the number of ejected particles is following a Poisson distribution. During the Yarkovsky run significantly more particles escaped from the Trojan region as during the non-Yarkovsky run. However, the effect is not as big as for the fictitious population. The dashed vertical line in Figure 6.8 represents the end of the integration interval which was originally defined. However, due to some findings made while analyzing the results after 500 Myr, it was decided to extend the integration interval by another 500 Myr The next paragraphs give more detailed analysis on how the Yarkovsky effect influences the different dynamical properties of the Trojan population.

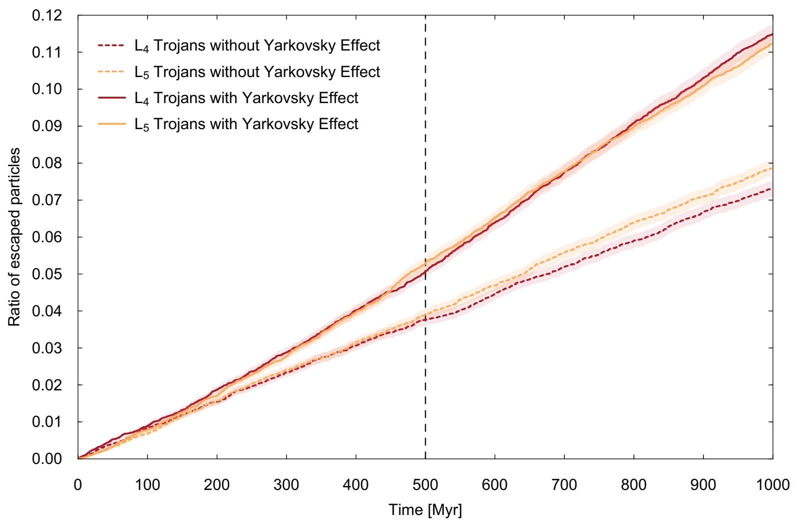

**Figure 6.8:** Ratio of ejected particles with and without considering the Yarkovsky effect during the 500 Myr plus another 500 Myr of extended integration. The impact of the Yarkovsky effect is not as big as for the fictitious population. Still there were significantly more particles ejected during the Yarkovsky run.

### Influence of the physical properties

The escape rate for each category and size was separately monitored to determine the impact of the Yarkovsky effect on the different physical properties of the objects. As can be seen in Figure 6.9, smaller objects are more likely to be ejected from the Trojan region. The colored transparent areas in the plots represent the 1 $\sigma$ confidence region for each category. The gray area in each plot is the confidence region for the non-Yarkovsky run. Because the number of particles for each category and size in the Yarkovsky run was 2048 while the number of particles in the non-Yarkovsky run was 32768, the 1 $\sigma$ confidence region for the non-Yarkovsky run is accordingly narrower than those corresponding to the sub-populations of the Yarkovsky run.

While the escape rates of the objects of $R = 1$ and $R = 10$ km seem not to be affected by the Yarkovsky force, for the 10 m radius, low thermal inertia and low density objects,

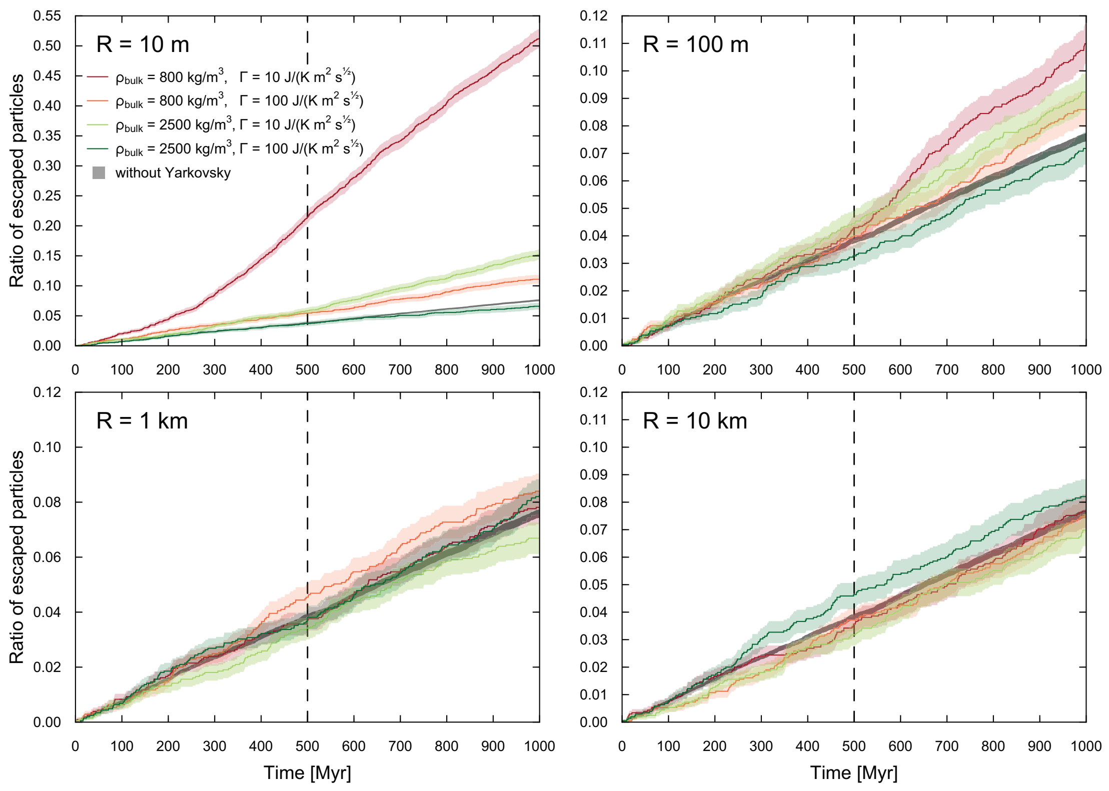

**Figure 6.9:** Ratio of ejected particles for the different sizes and physical properties of the particles. The transparent areas represent the 1 $\sigma$ confidence region, assuming the number of ejected particles is following a Poisson distribution. The gray area represents the 1 $\sigma$ confidence region for the ratio without considering the Yarkovsky effect. In general, smaller particles are more likely to be ejected. Objects of 1 and 10 km radius are practically not affected.

during the first 500 Myr of the Yarkovsky run more than 5 times as many particles as during the non-Yarkovsky run were ejected. Also the ratio of escaped particles with low thermal inertia and high density as well as the one for the high thermal inertia and low density are clearly above the confidence region of the non-Yarkovsky run. This result matches very well the expected behavior discussed in the preliminary considerations in the beginning of Section 6.2. An interesting feature for the $R = 10$ m low density, low thermal inertia particles is that the rate of ejected particles per unit time increases after about 250 Myr. If the initial orbital distribution were in a steady state, which can be assumed for an evolved population like the Jupiter Trojans, one would expect the escape rates to be constant over time for each size and category of physical properties. A possible explanation for the rate change is that the initial orbital distribution of the test particles for this experiment reflects the distribution of the currently known Jupiter Trojans. This distribution is based on objects much larger than 1 km in radius which are less influenced by the Yarkovsky effect. As most of them have very long dynamical lifetimes (as few as 5% of all particles were ejected during the non-Yarkovsky run), they are on very stable orbits.

The reason for the suddenly increasing escape rate would be that it takes some time until the Yarkovsky force moves the much smaller particles considered in this experiment from their stable initial orbits to less stable regions where they are eventually ejected. For small particles this should happen faster than for larger ones because they are more strongly affected by the Yarkovsky force. To test this hypothesis, the simulations were continued for another 500 Myr. Indeed the same effect is visible for the $R = 100$ m objects. Except for the high thermal inertia and high density objects, the escape rate increases after about 500 Myr of integration. This implies that the initial conditions for this experiment may not represent the actual orbital distribution for Trojans smaller than 1 km in radius and suggests that the actual number of escaped particles would be higher. Therefore, all further analysis was done considering the extended integration interval of 1 Gyr.

### Ratio of escaped $L_4$ and $L_5$ Trojans

The observed asymmetry of the Trojan clouds is based on the known population, which consists of objects having radii of tens to hundreds km. Because these bodies are not influenced by the Yarkovsky effect, the latter can not explain the observed asymmetry. However, it is still meaningful to examine whether the effect may influence the ratio of escaping particles between $L_5$ and $L_4$ for smaller objects. In Figure 6.8 the escape ratios are separately monitored for the two clouds. For the non-Yarkovsky run, the ratio between escaped particles from $L_5$ and $L_4$ after 1 Gyr was $1.07 \pm 0.04$, again assuming the number of ejected particles is Poisson distributed. This ratio is lower than the value of $1.22 \pm 0.08$ reported by Di Sisto et al. [32] for the numbered Jupiter Trojans after 4.5 Gyr of integration. Di Sisto et al. concluded that the ratio of escapes between $L_5$ and $L_4$ is very sensitive to the orbital distribution. They also could not find any difference in escape rate from either of the Trojan clouds for a fictitious population containing 18200 particles. For this reason Di Sisto et al. decided to consider only numbered objects when studying the real population because of the superior precision of their orbits. When considering only clones coming from numbered Trojans, during the non-Yarkovsky run performed in this work, the ratio between escaped $L_5$ and $L_4$ increases to $1.12 \pm 0.04$ which is higher but not significantly different from the ratio including the multi-opposition objects. This suggests that the less accurate orbits of the multi-opposition objects do not significantly influence the results.

When enabling the Yarkovsky effect, the escape rates from $L_4$ and $L_5$ seem to be more or less the same. No trend of more particles escaping from any of the two clouds is visible. Looking at the escape rates for each size (see Table 6.3) suggests that the ratio of escape rate between $L_4$ and $L_4$ decreases with size, however the error on the ratios is too large to allow a significant conclusion. Still, this further supports the hypothesis that the Yarkovsky effect changes the orbital distribution of small Trojans and that the difference in escape rate is very sensitive to the initial orbital distribution.

**Table 6.3:** Ratio between escaped $L_5$ and $L_4$ particles for each size during the Yarkovsky run.

| 10 m | 100 m | 1 km | 10km |
|---|---|---|---|
| 0.91±0.08 | 0.97±0.10 | 1.08±0.11 | 1.09±0.11 |

### Implications on the libration amplitude

To better understand how the Yarkovsky effect changes the Trojans' long-term orbital evolution, the initial libration amplitudes and eccentricities of the particles ejected during both runs were compared. In general, orbits having a low libration amplitude, eccentricity and inclination tend to be more stable. As can be seen in Figure 6.10, objects having high eccentricity or libration amplitude were ejected early during the simulation while those having lower eccentricity or libration amplitude stayed for a longer time span. The figure also shows that more particles were ejected which were initially located in the low libration amplitude regime during the Yarkovsky run as during the non-Yarkovsky run. The average initial libration amplitude for the particles ejected during the non-Yarkovsky run was 21.8° for $L_4$ and 20.9° for $L_5$ Trojans. For the Yarkovsky run, these values slightly decreased to 20.1 and 19.5° for $L_4$ and $L_5$ Trojans, respectively. Although the difference does not seem to be very significant, this indicates that the Yarkovsky effect causes objects which initially were on stable orbits to leave the Trojan region. If this is indeed true, it should also reflect in the distribution of the libration amplitudes at the end of the integration. If the low amplitude particles which escaped during the Yarkovsky run were removed because their libration amplitudes became too large, one would expect more objects with higher amplitudes at the end of the integration time span than there were in the beginning. Thus, the libration amplitudes of the surviving particles at the end of the integration were computed and compared with their initial values. Indeed, as can be seen in the histograms in Figure 6.11, there were more particles with large libration amplitude after 1 Gyr. The smallest bodies show the largest abundancy of high amplitude objects. For larger particles, the difference becomes much smaller but is still visible. This means that even objects of 1 km or larger in radius are affected by the Yarkovsky force and that even more of them would leave the Trojan region if the integration time span would be further increased.

Surprisingly, the Yarkovsky effect did not increase the libration amplitudes for all objects: The upper left panel of Figure 6.11 clearly shows that for the smallest bodies, there also

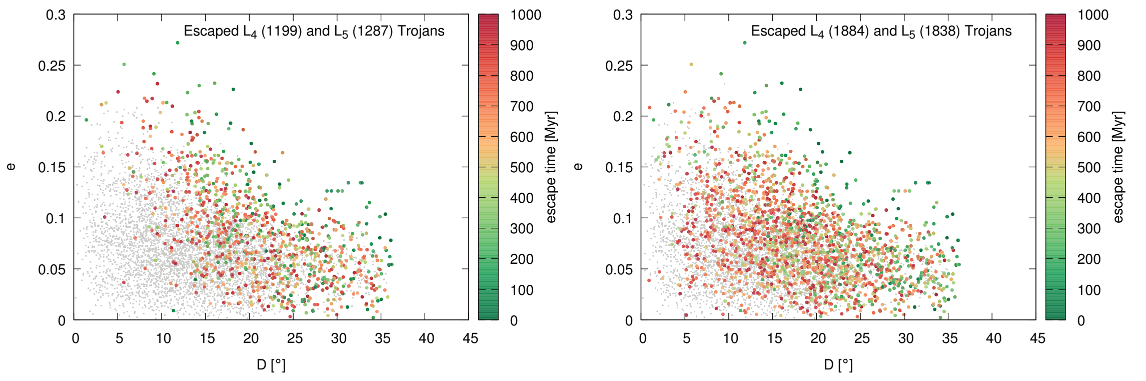

**Figure 6.10:** Initial amplitude and escape time for the escaped particles during the non-Yarkovsky run (left panel) and the Yarkovsky run (right panel). In general, particles having high e and D are ejected earlier. During the Yarkovsky run, more particles in the lower D regime are affected.

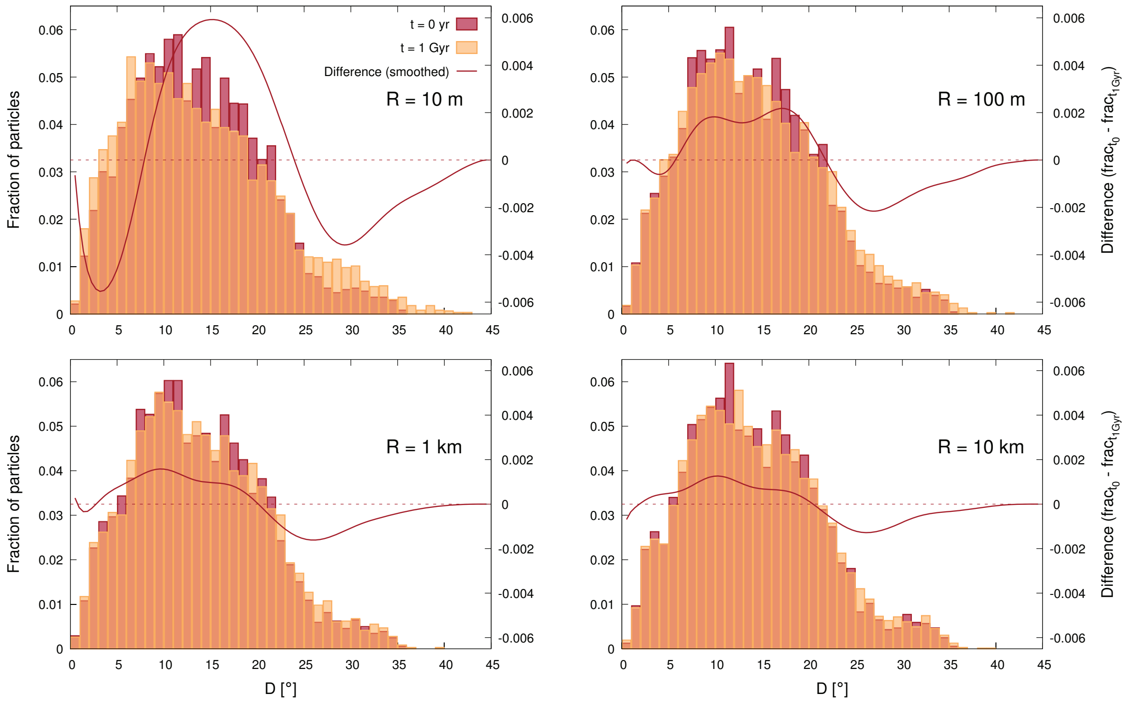

**Figure 6.11:** Comparison of the distributions of libration amplitudes of particles which remained in the Trojan region over the complete integration time-span in the Yarkovsky-run at t=0 and t=1 Gyr. There seems to be an excess of large amplitude orbits for objects of all sizes. For the smallest objects, there are also more low amplitude orbits after 1 Gyr than at the beginning which means that the Yarkovsky effect increases the dynamical lifetime of these objects.

seems to be an abundancy of low amplitude objects after 1 Gyr of integration which means that the Yarkovsky effect tends to increase the dynamical lifetime for some of the small objects. The same trend was also present when separately looking at either of the Trojan clouds suggesting that both clouds are affected in the same way. As mentioned in Section 2.4, the drift in semi-major axis caused by the diurnal component of the Yarkovsky force depends on the sense of the object's rotation. The Yarkovsky effect leads to an increase of the semi-major axis of prograde rotators while it decreases the semi-major axis of retrograde rotators. To see if the sense of rotation causes the feature observed in Figure 6.11, the change in libration amplitude was analyzed separately for prograde and retrograde rotators. Figure 6.12 shows the distribution of libration amplitudes for prograde vs retrograde rotators after 1 Gyr of integration. For particles with R<1 km, it can clearly be seen that there are more low amplitude objects rotating in prograde direction while there are more high amplitude objects rotating in retrograde direction. Comparing the two distributions for each radius with the Kolmogorov-Smirnoff non-parametric test revealed that the libration amplitudes of the 10 and 100 m radius objects are clearly not following the same distribution with a confidence level of more than 99% while the test could not distinguish between prograde and retrograde rotating objects of 1 and 10 km radius. The trend was also visible among the ejected particles. In total, there were 16492 prograde and 16276 retrograde objects of which 1556 (9.4%) and 2166 (13.3%) were ejected, respectively.

If the Yarkovsky effect indeed causes the retrograde rotators to be ejected while prograde rotators are stabilized, this should reflect in the obliquity distribution of the small Jupiter Trojans. Unfortunately, as of August 2017, according to the Asteroid Lightcurve Database [110], available at the minorplanet.info website[1], the spin axis orientations of only four rather large Jupiter Trojans are known. Therefore, the influence of the Yarkovsky effect on the obliquity distribution of Jupiter Trojans effect remains to be confirmed by observations in the future.

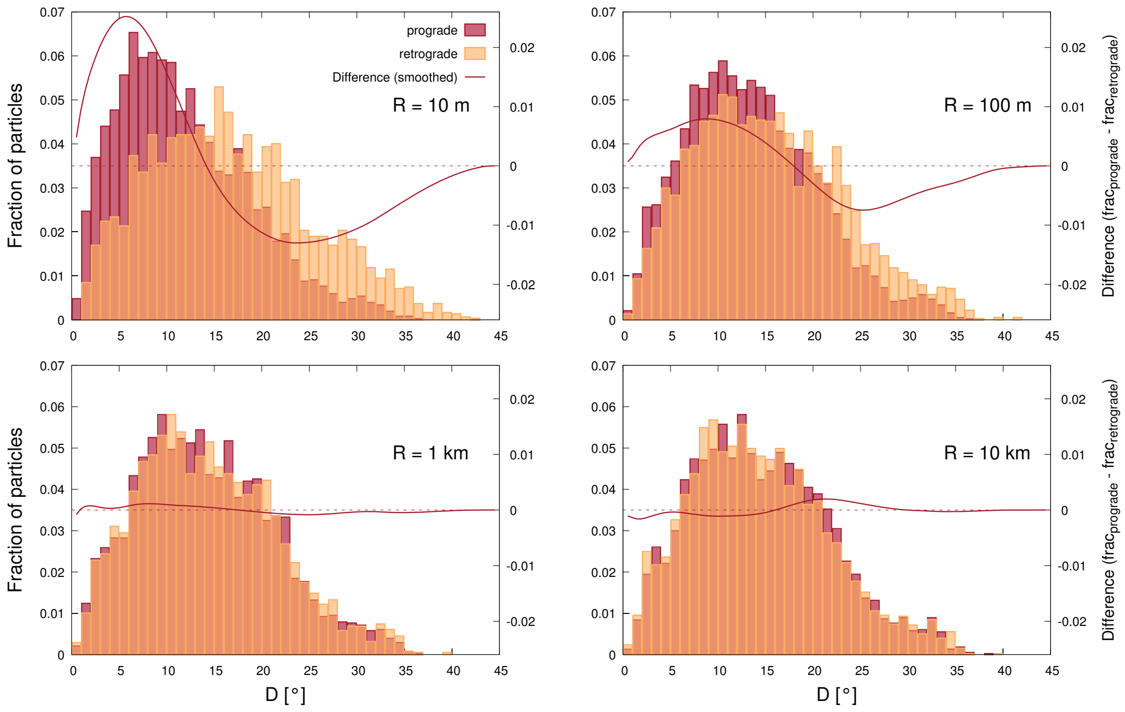

**Figure 6.12:** Libration amplitudes of retrograde vs prograde rotating objects which survived the 1 Gyr of integration incorporating the Yarkovsky effect. For the 10 and 100 m radius objects, there are more prograde objects with small libration amplitudes while there are more retrograde objects with large libration amplitudes.

## Conclusions

Although the integrations only covered a fraction of the estimated age of the Jupiter Trojans, about 4.5 Gyr, some important results were obtained. It was proven that at least the small objects of less than 1 km in radius are indeed influenced by the Yarkovsky effect and that prograde rotators migrate to regions around the Lagrangian points which provide longer residence times while retrograde rotators are more likely to be ejected from the Trojan clouds. The experiment also revealed that the difference in particles escaping

1 http://www.minorplanet.info/lightcurvedatabase.html

from L$_4$ and L$_5$ is caused by the different orbital distributions of each group. Due to the influence of the Yarkovsky effect, objects smaller than 1 km in radius should exhibit a distinct orbital distribution from the one currently observed. For small Trojans, this leads to the escape rates from both clouds being equalized. The fact that, due to the Yarkovsky effect, the residence time in Trojan orbits decreases with decreasing object radius leads to further implications on the size-frequency distribution of these objects: there should exist less Trojans with $R < 1$ km than predicted by purely dynamical and collisional models. However, exact statements can only be made after the physical properties of the Trojans are better constrained.

### Caveats

Some things should be mentioned to assess the significance of the results of this work. First of all, the thermal properties were set to values obtained from rather large objects and might be different for smaller bodies considered in this experiment. Secondly, the particles were assumed to be perfect spheres, as for most studies modelling the dynamical evolution of a huge amount of bodies under the Yarkovsky effect. This simplification might have an influence on the magnitude of the Yarkovsky effect, especially for small bodies which can have very irregular shapes. Also the YORP effect, which can change the rotational properties over long time scales, was not considered within this study. Last but not least, the smaller the objects are, the more they rise in numbers and consequently, the more frequent are collisions which, when not being disruptive, change the objects spin axis orientation and spin rate resulting in a random Yarkovsky-drift direction over very long time scales.

# CHAPTER 7

# Future work

The aim of this work was to implement a software package with tools for studying long-term orbital evolution of thousands of objects in the solar system which can efficiently utilize modern computer hardware and to conduct a scientific experiment to demonstrate its possibilities and benefits. So far, this package contains three integration methods and is also capable of taking into account the non-gravitational Yarkovsky force. Still, there are a lot more things to do in order to extend and improve cuSwift.

## 7.1 Improve and Implement more non-gravitational effects

### 7.1.1 Improve the Yarkovsky effect and include YORP Effect

In the current version of cuSwift, two different densities can be assigned to each body in order to account for different properties of the material on the asteroid's surface and interior. However, it is known that besides density also thermal inertia of the surface regolith changes with depth [94] [52] and temperature [29]. Changing the routines for calculating the Yarkovsky force in order to take into account these changes would result in more a more realistic implementation of the effect.

### 7.1.2 Include YORP Effect

The Yarkovsky-O'Keefe-Radzievskii-Paddack (YORP) effect is, like the Yarkovsky effect, a non-gravitational force acting on small bodies in the solar system. It can alter, on long time scales the objects spin rates and axis which has implications on the magnitude and direction of the semi-major axis drift caused by the Yarkovsky effect [16]. The implementation of MVS by Brož [13] also considers the YORP effect. These routines could easily be added in cuSwift as well.

### 7.1.3 Acceleration due to cometary activity

It is well known that the acceleration caused by cometary activity can lead to secular changes of the orbits of comets [70]. Compared to the age of the solar system, the active short-period comets entered the inner system very recently. For example studies of the dynamical evolution of comet 67P/Churyumov-Gerasimenko, target of the ongoing Rosetta mission, suggest that its semi-major axis decreased from 10 AU to 4.5 AU in the last 10 kyr [67]. Close encounters with Jupiter, especially those in 1923 and 1959, eventually caused the perihelion distance of 67P to decrease to 1.3 AU. Living on planet crossing orbits, one might think the orbital evolution of the short period comets is dominated solely by close encounters. However, in contrast to the Yarkovsky and YORP effects, which alter the orbital and rotational properties of asteroids within time-scales of millions of years, cometary activity has a much greater impact. The rotational period of comet 67P/Churyumov-Gerasimenko, decreased as much as almost 3% over a single orbit around the sun [84]. For modelling the dynamical evolution of active comets, the non-gravitational forces caused by cometary activity might paly an important role. Adding a method which models these effects to cuSwift would enable it to be used for studying dynamical evolution of active comets too.

### 7.1.4 Collisions and Fragmentation

Another important feature to be included in cuSwift is the treatment of collisions among the test particles. In general, there are two ways of doing this.

One way would be to track the distances between all particles during the integration. A collision occurs when the distance between two particles becomes smaller than the sum of their radii. Depending on direction and speed of the impact as well as the compositions of the colliding bodies, it can be determined how the trajectory of the surviving bodies changes (if they do not get completely destroyed) and how many new objects should be included due to fragmentation [64]. The advantage of this method would be that each individual collision would be properly resolved. On the other hand, the enormous advantage of the test particles not belonging to the $n^2$ part of the $n$-body problem is lost because the mutual distances of all particles must be evaluated at each time step. Further, as the distances the particles travel each time step is much larger than their radii, it needs to be ensured no collisions occurring between two time steps are missed. This means that computation time would significantly increase and results in the number of particles which can be considered in order to get reasonable execution times dramatically decreases.

Another method would be not to resolve each single collision but to use a stochastic method. The collision probabilities for bodies of different sizes as well as the average directions and velocities are well known for the different regions in the solar system [28]. This knowledge can be used to introduce random collisions for each object. Like for the first method, the outcome is determined by the impact geometry and the physical properties of the colliding body. However, The parameters describing the collision are randomly generated according to their probabilities. This method does not resolve the actual collisions among the particles but is computationally much less expensive, as it does not introduce the $\mathrm{n}^2$ complexity for the test particles.

## 7.2 Implement additional Integration Methods

Until now, cuSwift contains three integration methods. There are however, many more algorithms which could be implemented to the package to study different phenomena such as planetary accretion [35], the behavior of rubble pile asteroids [97] or for performing highly accurate experiments for simulating spacecraft trajectories or making impact and occultation predictions.

So far, cuSwift can utilize only a single GPU. Adding support simultaneous usage of several GPUs would make it possible to efficiently run cuSwift on large-scaled supercomputers which can contain many GPUs. This would result in a much larger number of particles can be included in the simulation.

# CHAPTER 8

## Bibliography

[1] M. Alexandersen, B. Gladman, S. Greenstreet, J. J. Kavelaars, J.-M. Petit, and S. Gwyn. ‘A Uranian Trojan and the Frequency of Temporary Giant-Planet Co-Orbitals’. In: *Science* 341.6149 (2013), pp. 994–997 (cit. on p. 75).

[2] J. H. Applegate, M. R. Douglas, Y. Gursel, P. Hunter, C. L. Seitz, and G. J. Sussman. ‘A Digital Orrery’. In: *IEEE Transactions on Computers* 34.9 (1985), pp. 822–831 (cit. on p. 51).

[3] J. H. Applegate, M. R. Douglas, Y. Gursel, G. J. Sussman, and J. Wisdom. ‘The outer solar system for 200 million years’. In: *The Astronomical Journal* 92.1 (1986), pp. 176–194 (cit. on p. 50).

[4] H. Baudisch and R. Dvorak. ‘Where are the Saturn Trojans’. In: *Proceedings, Journees 'Systeme de reference spatio-temporels'* (2011), pp. 225–228 (cit. on p. 75).

[5] W. Benz and E. Asphaug. ‘Catastrophic Disruptions Revisited’. In: *Icarus* 142 (1999), pp. 5–20 (cit. on p. 85).

[6] D. Bockelée-Morvan, J. Crovisier, M. J. Mumma, and H. A. Weaver. ‘The composition of cometary volatiles’. In: *Comets II.* 2004, pp. 391–423 (cit. on p. 6).

[7] W. F. Bottke, A. Morbidelli, R. Jedicke, J.-M. Petit, H. F. Levison, P. Michel, and T. S. Metcalfe. ‘Debiased Orbital and Absolute Magnitude Distribution of the Near-Earth Objects’. In: *Icarus* 156.2 (2002), pp. 399–433 (cit. on pp. e, 6, 28).

[8] W. F. Bottke, D. Vokrouhlický, M. Brož, D. Nesvorný, and A. Morbidelli. ‘Dynamical Spreading of Asteroid Families by the Yarkovsky Effect’. In: *Science* 294.5547 (2001), pp. 1693–1696 (cit. on p. 28).

[9] W. F. Bottke, D. Vokrouhlický, D. P. Rubincam, and M. Brož. ‘The effect of Yarkovsky thermal forces on the dynamical evolution of asteroids and meteoroids’. In: *Asteroids III.* 2002, pp. 395–408 (cit. on pp. 25, 27, 62).

[10] W. F. Bottke, D. Vokrouhlický, D. P. Rubincam, and D. Nesvorný. ‘The Yarkovsky and Yorp Effects: Implications for Asteroid Dynamics’. In: *Annual Review of Earth and Planetary Sciences* 34 (2006), pp. 157–191 (cit. on pp. 26, 28, 79).

[11] W. Bottke, D. Durda, D. Nesvorny, R. Jedicke, A. Morbidelli, D. Vokrouhlický, and H. Levison. ‘Linking the collisional history of the main asteroid belt to its dynamical excitation and depletion’. en. In: *Icarus* 179.1 (2005), pp. 63–94 (cit. on p. 86).

[12] M. Brož. ‘A faster version of the SWIFT-MVS integrator and implementation of the Yarkovsky force’. In: *Asteroids, Comets, Meteors (ACM) — Berlin, Germany* (2002) (cit. on pp. 39, 40, 47, 61).

[13] M. Brož. ‘Yarkovsky Effect and the Dynamics of the Solar System’. PhD thesis. Praha: Charles University, Faculty of Mathematics and Physics Astronomical Institute, 2006 (cit. on pp. e, 25, 47, 62, 63, 97).

[14] M. W. Buie et al. ‘Size and Shape from Stellar Occultation Observations of the Double Jupiter Trojan Patroclus and Menoetius’. In: *The Astronomical Journal* 149.3 (2015), p. 113 (cit. on p. 77).

[15] J. A. Burns, P. L. Lamy, and S. Soter. ‘Radiation forces on small particles in the solar system’. In: *Icarus* 40.1 (1979), pp. 1–48 (cit. on p. 29).

[16] D. Čapek and D. Vokrouhlický. ‘The YORP effect with finite thermal conductivity’. In: *Icarus* 172.2 (2004), pp. 526–536 (cit. on p. 97).

[17] M. Carpino, A. Milani, and A. M. Nobili. ‘Long-term numerical integrations and synthetic theories for the motion of the outer planets’. In: *Astronomy & Astrophysics* 181 (1987), pp. 182–194 (cit. on p. 51).

[18] J. E. Chambers. ‘A hybrid symplectic integrator that permits close encounters between massive bodies’. In: *Monthly Notices of the Royal Astronomical Society* 304.4 (1999), pp. 793–799 (cit. on p. 20).

[19] J. E. Chambers. ‘N-Body Integrators for Planets in Binary Star Systems’. In: *Planets in Binary Star Systems*. Vol. 366. 2010, pp. 239–263 (cit. on p. 20).

[20] C. R. Chapman and D. R. Davis. ‘Asteroid collisional evolution - Evidence for a much larger early population’. In: *Science* 190.4214 (1975), pp. 553–556 (cit. on p. 29).

[21] S. R. Chesley, S. J. Ostro, D. Vokrouhlický, D. Čapek, J. D. Giorgini, M. C. Nolan, J. L. Margot, A. A. Hine, L. A. M. Benner, and A. B. Chamberlin. ‘Direct Detection of the Yarkovsky Effect by Radar Ranging to Asteroid 6489 Golevka’. In: *Science* 302.5651 (2003), pp. 1739–1742 (cit. on p. 25).

[22] C. J. Cohen and E. C. Hubbard. ‘Libration of the close approaches of Pluto to Neptune’. In: *The Astronomical Journal* 70 (1965), p. 10 (cit. on p. 50).

[23] M. Connors, P. Wiegert, and C. Veillet. ‘Earth’s Trojan asteroid’. In: *Nature* 475.7357 (2011), pp. 481–483 (cit. on p. 75).

[24] NVIDIA Corporation. *CUDA C Best Practices Guide*. `http://docs.nvidia.com/cuda/cuda-c-best-practices-guide`. Accessed: 2015-04-08. 2015 (cit. on p. 33).

[25] NVIDIA Corporation. *CUDA C Programming Guide*. `http://docs.nvidia.com/cuda/cuda-c-programming-guide`. Accessed: 2015-04-08. 2015 (cit. on pp. 33, 43, 48, 49).

[26] NVIDIA Corporation. *GeForce 256 The World’s First GPU*. `http://www.nvidia.com/page/geforce256.html`. Accessed: 2014-06-09. 1999 (cit. on p. 31).

[27] J. M. A. Danby. *Fundamentals of celestial mechanics*. 1988 (cit. on pp. 19, 43).

[28] D. R. Davis, D. D. Durda, F. Marzari, A. Campo Bagatin, and R. Gil-Hutton. 'Collisional evolution of small-body populations'. In: *Asteroids III*. 2002, pp. 545–558 (cit. on pp. 86, 98).

[29] M. Delbo, M. Mueller, J. P. Emery, B. Rozitis, and M. T. Capria. 'Asteroid thermophysical modeling'. In: *Asteroids IV* (2015), pp. 107–128 (cit. on p. 97).

[30] A. Dell'Oro, F. Marzari, P. Paolicchi, and V. Vanzani. 'Updated collisional probabilities of minor body populations'. In: *Astronomy & Astrophysics* 366.3 (2001), pp. 1053–1060 (cit. on p. 86).

[31] A. dell'Oro, F. PAolicchi, P. Marzari, E. Dotto, and V. Vanzani. 'Trojan collision probability: a statistical approach'. In: *Astronomy & Astrophysics* 339 (1998), pp. 272–277 (cit. on p. 85).

[32] R. P. Di Sisto, X. S. Ramos, and C. Beaugé. 'Giga-year evolution of Jupiter Trojans and the asymmetry problem'. In: *Icarus* 243 (2014), pp. 287–295 (cit. on pp. 76, 78, 91).

[33] F. Diacu. 'The solution of the n-body problem'. In: *The Mathematical Intelligencer* 18.3 (1996), pp. 66–70 (cit. on p. 10).

[34] E. Dotto, J. P. Emery, M. A. Barucci, A. Morbidelli, and D. P. Cruikshank. 'De Troianis: the Trojans in the planetary system'. In: *The Solar System Beyond Neptune*. 2008, pp. 383–395 (cit. on p. 77).

[35] M. J. Duncan, H. F. Levison, and M. H. Lee. 'A Multiple Time Step Symplectic Algorithm for Integrating Close Encounters'. In: *The Astronomical Journal* 116.4 (1998), p. 2067 (cit. on pp. 20, 98).

[36] D. D. Durda, R. Greenberg, and R. Jedicke. 'Collisional Models and Scaling Laws: A New Interpretation of the Shape of the Main-Belt Asteroid Size Distribution'. In: *Icarus* 135 (1998), pp. 431–440 (cit. on p. 85).

[37] J. Durech, B. Carry, M. Delbo, M. Kaasalainen, and M. Viikinkoski. 'Asteroid Models from Multiple Data Sources'. In: *ArXiv e-prints* (2015) (cit. on p. 25).

[38] G. C. de Elía and A. Brunini. 'Collisional and dynamical evolution of the $L_4$ Trojan asteroids'. In: *Astronomy & Astrophysics* 475.1 (2007), pp. 375–389 (cit. on pp. 87, 88).

[39] B. Erdi. 'Long periodic perturbations of Trojan asteroids'. In: *Celestial Mechanics* 43 (1988), pp. 303–308 (cit. on pp. 76, 78, 79).

[40] E. Everhart. 'An efficient Integrator that uses Gauss-Radau Spacings'. In: *Dynamics of Comets: Their Origin and Evolution*. 1985, pp. 185–202 (cit. on pp. e, 39).

[41] P. Farinella and D. Vokrouhlický. 'Semimajor Axis Mobility of Asteroidal Fragments'. In: *Science* 283 (1999), p. 1507 (cit. on pp. 28, 29, 61, 81, 83, 87).

[42] P. Farinella, D. Vokrouhlický, and W. K. Hartmann. 'Meteorite Delivery via Yarkovsky Orbital Drift'. In: *Icarus* 132 (1998), pp. 378–387 (cit. on p. 86).

[43] C. de la Fuente Marcos and R. de la Fuente Marcos. ‘Three new stable L5 Mars Trojans’. In: *Monthly Notices of the Royal Astronomical Society: Letters* 432 (2013), pp. 31–35 (cit. on p. 75).

[44] J. Fung and S. Mann. ‘OpenVIDIA: parallel GPU computer vision’. In: *Proceedings of the 13th annual ACM international conference on Multimedia*. 2005, pp. 849–852 (cit. on p. 32).

[45] T. Grav et al. ‘WISE/NEOWISE Observations of the Hilda Population: Preliminary Results’. In: *The Astrophysical Journal* 744.2 (2012), p. 197 (cit. on pp. 77, 81).

[46] R. Greenberg, J. F. Wacker, W. K. Hartmann, and C. R. Chapman. ‘Planetesimals to planets: Numerical simulation of collisional evolution’. In: *Icarus* 35.1 (1978), pp. 1–26 (cit. on p. 85).

[47] S. Greenstreet, B. Gladman, H. Ngo, M. Granvik, and S. Larson. ‘Production of Near-Earth Asteroids on Retrograde Orbits’. In: *The Astrophysical Journal Letters* 749.2 (2012), p. L39 (cit. on p. 22).

[48] S. Greenstreet, H. Ngo, and B. Gladman. ‘The orbital distribution of Near-Earth Objects inside Earth’s orbit’. In: *Icarus* 217.1 (2012), pp. 355–366 (cit. on pp. e, 22, 72, 73).

[49] E. Hairer, C. Lubich, and G. Wanner. ‘Geometric numerical integration illustrated by the Störmer-Verlet method’. In: *Acta Numerica* 12 (2003), pp. 399–450 (cit. on pp. 15, 17).

[50] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration — Structure-Preserving Algorithms for Ordinary Differential Equations*. 2002 (cit. on p. 15).

[51] J. Hanuš, M. Delbó, J. Ďurech, and V. Alí-Lagoa. ‘Thermophysical modeling of asteroids from WISE thermal infrared data — Significance of the shape model and the pole orientation uncertainties’. In: *Icarus* 256 (2015), pp. 101–116 (cit. on p. 77).

[52] Alan W. Harris and Line Drube. ‘THERMAL TOMOGRAPHY OF ASTEROID SURFACE STRUCTURE’. In: *The Astrophysical Journal* 832.2 (2016), p. 127 (cit. on p. 97).

[53] W. K. Hartmann, P. Farinella, D. Vokrouhlický, S. J. Weidenschilling, A. Morbidelli, F. Marzari, D. R. Davis, and E. Ryan. ‘Reviewing the Yarkovsky effect: New light on the delivery of stone and iron meteorites from the asteroid belt’. In: *Meteoritics & Planetary Science* 34.S4 (1999), A161–A167 (cit. on p. 25).

[54] J. Horner, T. G. Müller, and P. S. Lykawka. ‘(1173) Anchises — thermophysical and dynamical studies of a dynamically unstable Jovian Trojan’. In: *Monthly Notices of the Royal Astronomical Society* 423.3 (2012), pp. 2587–2596 (cit. on p. 77).

[55] X. Y. Hou, D. J. Scheeres, and L. Liu. ‘Saturn Trojans: a dynamical point of view’. In: *Monthly Notices of the Royal Astronomical Society* 437.2 (2014), pp. 1420–1433 (cit. on p. 75).

[56] IBM. *IBM 5175 personal computer professional graphics display and personal computer professional graphics controller family.* Announcement Letter Number 184-112 http://www-01.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_ca/2/897/ENUS184-112/index.html&lang=en&request_locale=en. Accessed: 2014-06-09. 1984 (cit. on p. 31).

[57] David C. Jewitt, Chadwick A. Trujillo, and Jane X. Luu. 'Population and size distribution of small jovian Trojan asteroids'. In: *The Astronomical Journal* 120.2 (2000), p. 1140 (cit. on pp. 86, 87).

[58] O. Karlsson. 'Creation, detection, and evolution of Jupiter Trojan families'. In: *Astronomische Nachrichten* 332.6 (2011), pp. 562–579 (cit. on p. 78).

[59] D. Kirk and W. Hwu. *Programming massively parallel processors: a hands-on approach.* 2010 (cit. on pp. 33, 35).

[60] G. A. Krasinsky, E. V. Pitjeva, M. V. Vasilyev, and E. I. Yagudina. 'Hidden Mass in the Asteroid Belt'. In: *Icarus* 158.1 (2002), pp. 98–105 (cit. on p. 20).

[61] P. Lacerda and D. C. Jewitt. 'Densities of Solar System objects from their rotational light curves'. In: *The Astronomical Journal* 133.4 (2007), p. 1393 (cit. on p. 77).

[62] J. Larsen and T. Spahr. 'Observational Selection Effects in Asteroid Surveys and Estimates of Asteroid Population Sizes'. In: *Asteroids III* (2002), p. 71 (cit. on p. 3).

[63] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. 'Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU'. In: *ACM SIGARCH Computer Architecture News* 38.3 (2010), pp. 451–460 (cit. on p. 65).

[64] Z. M. Leinhardt and S. T. Stewart. 'COLLISIONS BETWEEN GRAVITY-DOMINATED BODIES. I. OUTCOME REGIMES AND SCALING LAWS'. In: *The Astrophysical Journal* 745.1 (2012), p. 79 (cit. on p. 98).

[65] H. F. Levison and M. J. Duncan. 'The long-term dynamical behavior of short-period comets'. In: *Icarus* 108 (1994), pp. 18–36 (cit. on pp. e, 21, 22, 39, 56, 58).

[66] H. F. Levison, E. M. Shoemaker, and C. S. Shoemaker. 'Dynamical evolution of Jupiter's Trojan asteroids'. In: *Nature* 385 (1997), pp. 42–44 (cit. on p. 76).

[67] L. Maquet. 'The recent dynamical history of comet 67P/Churyumov-Gerasimenko'. In: *Astronomy & Astrophysics* 579 (2015), A78 (cit. on p. 97).

[68] F. Marchis, J. Durech, J. Castillo-Rogez, F. Vachier, M. Cuk, J. Berthier, M. H. Wong, P. Kalas, G. Duchene, M. A. van Dam, H. Hamanowa, and M. Viikinkoski. 'The Puzzling Mutual Orbit of the Binary Trojan Asteroid (624) Hektor'. In: *The Astrophysical Journal Letters* 783.2 (2014), p. L37 (cit. on p. 77).

[69] F. Marchis et al. 'A low density of 0.8 g/cm-3 for the Trojan binary asteroid 617 Patroclus'. In: *Nature* 439.7076 (2006), pp. 565–567 (cit. on p. 77).

[70] B. G. Marsden. 'Comets and Nongravitational Forces'. In: *The Astronomical Journal* 73 (1968), p. 367 (cit. on p. 97).

[71] F. Marzari. ‘On the Instability of Jupiter’s Trojans’. In: *Icarus* 159.2 (2002), pp. 328–338 (cit. on p. 76).

[72] F. Marzari, H. Scholl, and P. Farinella. ‘Collision rates and impact velocities in the Trojan asteroid swarms’. In: *Icarus* 119.1 (1996), pp. 192–201 (cit. on p. 85).

[73] F. Marzari, P. Tricarico, and H. Scholl. ‘Stability of Jupiter Trojans investigated using frequency map analysis: the MATROS project’. In: *Monthly Notices of the Royal Astronomical Society* 345.4 (2003), pp. 1091–1100 (cit. on pp. 83, 84, 88).

[74] G. Mei and H. Tian. ‘Performance Impact of Data Layout on the GPU-accelerated IDW Interpolation’. In: *arXiv preprint arXiv:1402.4986* (2014) (cit. on p. 41).

[75] W. J. Merline, S. J. Weidenschilling, D. D. Durda, J. L. Margot, P. Pravec, and A. D. Storrs. ‘Asteroids do have satellites’. In: *Asteroids III* (2002), pp. 289–312 (cit. on p. 77).

[76] S. Mikkola and D. Merritt. ‘Algorithmic regularization with velocity-dependent forces’. In: *Monthly Notices of the Royal Astronomical Society* 372.1 (2006), pp. 219–223 (cit. on p. 17).

[77] A. Milani. ‘The Trojan asteroid belt: Proper elements, stability, chaos and families’. In: *Celestial Mechanics and Dynamical Astronomy* 57 (1993), pp. 59–94 (cit. on p. 84).

[78] A. Milani, A. M. Nobili, and M. Carpino. ‘Dynamics of Pluto’. In: *Icarus* 82.1 (1989), pp. 200–217 (cit. on p. 51).

[79] M. Moons and A. Morbidelli. ‘Secular resonances inside mean-motion commensurabilities: the 4/1, 3/1, 5/2 and 7/3 cases.’ In: *Icarus* 114 (1995), pp. 33–50 (cit. on p. 65).

[80] A. Moore and A. C. Quillen. ‘QYMSYM: A GPU-accelerated hybrid symplectic integrator that permits close encounters’. In: *New Astronomy* 16.7 (2011), pp. 445–455 (cit. on p. 43).

[81] A. Morbidelli. ‘Comets and their reservoirs: current dynamics and primordial evolution’. In: *Trans-Neptunian objects and comets.* 2008, pp. 79–163 (cit. on p. 5).

[82] A. Morbidelli, R. Brasser, R. Gomes, H. F. Levison, and K. Tsiganis. ‘Evidence from the asteroid belt for a violent past evolution of Jupiter’s orbit’. In: *The Astronomical Journal* 140.5 (Nov. 1, 2010), pp. 1391–1401 (cit. on p. e).

[83] A. Morbidelli, H. F. Levison, K. Tsiganis, and R. Gomes. ‘Chaotic capture of Jupiter’s Trojan asteroids in the early Solar System’. In: *Nature* 435.7041 (2005), pp. 462–465 (cit. on p. 76).

[84] S. Mottola et al. ‘The rotation state of 67P/Churyumov-Gerasimenko from approach observations with the OSIRIS cameras on Rosetta’. In: *Astronomy & Astrophysics* 569 (2014), p. L2 (cit. on p. 98).

[85] M. Müller, F. Marchis, J. P. Emery, A. W. Harris, S. Mottola, D. Hestroffer, J. Berthier, and M. di Martino. 'Eclipsing binary Trojan asteroid Patroclus: Thermal inertia from Spitzer observations'. In: *Icarus* 205.2 (2010), pp. 505–515 (cit. on p. 77).

[86] E. Mysen. 'An analytical model for YORP and Yarkovsky effects with a physical thermal lag'. In: *Astronomy & Astrophysics* 484.2 (2008), pp. 563–573 (cit. on p. 25).

[87] D. Nesvorný. 'How Long-Lived Are the Hypothetical Trojan Populations of Saturn, Uranus, and Neptune?' In: *Icarus* 160.2 (2002), pp. 271–288 (cit. on p. 75).

[88] D. Nesvorný, D. Vokrouhlický, and A. Morbidelli. 'Capture of Trojans by Jumping Jupiter'. In: *The Astrophysical Journal* 768.1 (2013), p. 45 (cit. on p. 76).

[89] S. B. Nicholson. 'The Trojan Asteroids'. In: *Leaflet of the Astronomical Society of the Pacific* 8 (1961), p. 239 (cit. on p. 75).

[90] D. Okunbor and R. D. Skeel. 'Explicit canonical methods for Hamiltonian systems'. In: *Mathematics of computation* 59.200 (1992), pp. 439–455 (cit. on p. 15).

[91] E. J. Öpik. 'Collision Probabilities with the Planets and the Distribution of Interplanetary Matter'. English. In: *Proceedings of the Royal Irish Academy. Section A: Mathematical and Physical Sciences* 54 (1951), pp. 165–199 (cit. on p. 25).

[92] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. 'A Survey of General-Purpose Computation on Graphics Hardware'. In: *Computer Graphics Forum* 26.1 (2007), pp. 80–113 (cit. on p. 32).

[93] J.-M. Petit et al. 'The Canada-France Ecliptic Plane Survey — Full Data Release: The Orbital Structure of the Kuiper Belt'. In: *The Astronomical Journal* 142.4 (2011), p. 131 (cit. on p. 5).

[94] C. Coke Pilbeam and J.R. Vaišnys. 'Contact thermal conductivity in lunar aggregates'. In: *Journal of Geophysical Research* 78.23 (1973), pp. 5233–5236 (cit. on p. 97).

[95] J. E. Prussing and B. A. Conway. *Orbital Mechanics.* 1993 (cit. on pp. 19, 43).

[96] W. Qiu-Dong. 'The global solution of the N-body problem'. English. In: *Celestial Mechanics and Dynamical Astronomy* 50.1 (1990), pp. 73–88 (cit. on p. 10).

[97] D. C. Richardson, P. Michel, K. J. Walsh, and K. W. Flynn. 'Numerical simulations of asteroids modelled as gravitational aggregates with cohesion'. In: *Planetary and Space Science* 57.2 (2009), pp. 183–192 (cit. on p. 98).

[98] D. Rubincam. 'Radiative Spin-up and Spin-down of Small Asteroids'. In: *Icarus* 148.1 (2000), pp. 2–11 (cit. on p. 28).

[99] R. D. Ruth. 'A Can0nical Integrati0n Technique'. In: *Nuclear Science, IEEE Transactions on* 30.4 (1983), pp. 2669–2671 (cit. on p. 15).

[100] J. Sanders. *CUDA by example: an introduction to general-purpose GPU programming.* 2011 (cit. on p. 33).

[101] S. S. Sheppard and C. A. Trujillo. ‘Detection of a trailing (L5) Neptune Trojan’. In: *Science* 329.5997 (2010), pp. 1304–1304 (cit. on p. 75).

[102] G. J. Sussman and J. Wisdom. ‘Numerical evidence that the motion of Pluto is chaotic’. In: *Science* 241 (1988), pp. 433–437 (cit. on pp. 51, 52, 56).

[103] P. Swings. ‘Cometary Spectra’. In: *Quarterly Journal of the Royal Astronomical Society* 6 (1965), p. 28 (cit. on p. 6).

[104] M. S. Tiscareno and R. Malhotra. ‘Chaotic Diffusion of Resonant Kuiper Belt Objects’. In: *The Astronomical Journal* 138.3 (2009), p. 827 (cit. on pp. 51, 52).

[105] K. Tsiganis, R. Gomes, A. Morbidelli, and H. F. Levison. ‘Origin of the orbital architecture of the giant planets of the Solar System’. In: *Nature* 435.7041 (2005), pp. 459–461 (cit. on p. 76).

[106] K. Tsiganis, H. Varvoglis, and R. Dvorak. ‘Chaotic Diffusion and Effective Stability of Jupiter Trojans’. In: *A Comparison of the Dynamical Evolution of Planetary Systems: Proceedings of the Sixth Alexander von Humboldt Colloquium on Celestial Mechanics*. 2005, pp. 71–87 (cit. on pp. 76–78).

[107] D. Vokrouhlický. ‘Diurnal Yarkovsky effect as a source of mobility of meter-sized asteroidal fragments. I. Linear theory’. In: *Astronomy & Astrophysics* 335 (1998), pp. 1093–1100 (cit. on pp. 26, 47).

[108] D. Vokrouhlický and W. F. Bottke. ‘The Yarkovsky thermal force on small asteroids and their fragments: Choosing the right albedo’. In: *Astronomy & Astrophysics* 371.1 (2001), pp. 350–353 (cit. on p. 79).

[109] D. Vokrouhlický and P. Farinella. ‘The Yarkovsky seasonal effect on asteroidal fragments: A nonlinearized theory for spherical bodies’. In: *The Astronomical Journal* 118.6 (1999), p. 3049 (cit. on pp. 27, 47).

[110] B. D. Warner, A. W. Harris, and P. Pravec. ‘The asteroid lightcurve database’. In: *Icarus* 202.1 (2009), pp. 134–146 (cit. on p. 94).

[111] N. Whitehead and A. Fit-Florea. ‘Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs’. In: *nVidia technical white paper* (2011) (cit. on p. 49).

[112] J. G. Williams and G. S. Benson. ‘Resonances in the Neptune-Pluto System’. In: *The Astrophysical Journal* 76 (1971), p. 167 (cit. on p. 50).

[113] J. Wisdom. ‘Chaotic behavior and the origin of the 3/1 Kirkwood gap’. In: *Icarus* 56 (1983), pp. 51–74 (cit. on p. 66).

[114] J. Wisdom. ‘The origin of the Kirkwood gaps - A mapping for asteroidal motion near the 3/1 commensurability’. In: *The Astrophysical Journal* 87 (1982), pp. 577–593 (cit. on p. 65).

[115] J. Wisdom and M. Holman. ‘Symplectic maps for the n-body problem’. In: *The Astronomical Journal* 102 (1991), p. 1528 (cit. on pp. e, 18, 19, 21, 39).

[116] I. Wong, M. E. Brown, and J. P. Emery. ‘The Differing Magnitude Distributions of the Two Jupiter Trojan Color Populations’. In: *The Astronomical Journal* 148.6 (2014), p. 112 (cit. on p. 85).

[117] D. K. Yeomans, P. W. Chodas, G. Sitarski, S. Szutowicz, and M. Królikowska. ‘Cometary orbit determination and nongravitational forces’. In: *Comets II* 1 (2004), pp. 137–151 (cit. on p. 29).

[118] F. Yoshida and T. Nakamura. ‘A Comparative Study of Size Distributions for Small L4 and L5 Jovian Trojans’. In: *Publications of the Astronomical Society of Japan* 60.2 (2008), p. 297 (cit. on p. 87).

[119] F. Yoshida and T. Nakamura. ‘Size distribution of faint Jovian L4 Trojan asteroids’. In: *The Astronomical Journal* 130.6 (2005), p. 2900 (cit. on pp. 75, 78, 87).

[120] H. Yoshida. ‘Recent progress in the theory and application of symplectic integrators’. English. In: *Celestial Mechanics and Dynamical Astronomy* 56.1-2 (1993), pp. 27–43 (cit. on p. 15).

[121] M. Yoshikawa. ‘Motions of asteroids at the Kirkwood gaps. I - On the 3:1 resonance with Jupiter’. In: *Icarus* 87 (1990), pp. 78–102 (cit. on p. 66).

# List of Figures

# List of Tables