The Gravitational Billion Body Problem

Het miljard deeltjes probleem

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 2 september 2014
klokke 16:15 uur

door

Jeroen Bédorf
geboren te Alkmaar
in 1984

Promotiecommissie

Promotor:	Prof. dr. Simon Portegies Zwart	
Overige leden:	Prof. dr. Steve L.W. McMillan	(Drexel University)
	Prof. dr. Henk Sips	(Technische Universiteit Delft)
	Prof. dr. Joost Batenburg	(Centrum Wiskunde & Informatica / Universiteit Leiden)
	Prof. dr. Huub Röttgering	

ISBN: 978-94-6259-260-5

Dit proefschrift werd ondersteund door NWO IsFast grant #643.000.802.

Contents

1 Introduction

If I have seen further it is by standing on the shoulders of giants.
Sir Isaac Newton, February 5, 1676

This quote can be seen as a metaphor for past and current day research. We all benefit from and build on the the work of our predecessors, just as Newton copied the “Standing on the shoulders of giants” part from the 12th century scholar Bernard of Chartres. Today this quote is still commonly used by authors (like Stephen Hawking), groups (The Free Software Movement), movies (Jurassic Park) and websites (Google Scholar). All to remind us that we continue the work of the ones who came before us, and nowhere is this more clear then in the fields of Astronomy and Computer Science. Archeologists keep uncovering artifacts that show that ancient civilizations were already interested in the planets and stars. They were able to track the movement of the heavenly bodies on the sky and used the position of the Sun when designing their temples and religious sites (see for example Stonehenge). And although digital computers were only introduced about 70 years ago they now form an integral part of our daily life. Without the Sun humanity would not have existed and without computers we would not have been able to start exploring the Moon and the planets around us. All these discoveries build upon previous work, from the formulation of gravity by Newton in 1686 to the invention of the transistor by John Bardeen, Walter Brattain, and William Shockley in 1947. Step by step these discoveries and inventions help us to understand more about our Galaxy and the Universe surrounding it.

Computers become faster each year by improving on previous designs or using inventions from physics and material sciences to create smaller and faster chips. Just like the works in this thesis are inspired by previous developments and take these works a step further by applying them to new hardware architectures. This shows how we gradually improve on our ability to use new designs in the computer industry and apply them to help us answer the age long questions we have about the Universe. However, sometimes it helps to step away from small updates and make radical changes, and thereby take big leaps forward in our ability to use new hardware architectures. The computer industry is evolving from performance improvements caused by increasing clock-speed to increasing the number of computational cores. In the Central Processing Unit we went from single to dual core designs 9 years ago. Currently the first 16 core chips have become available. However, a completely different hardware architecture, the Graphics Processing Unit, already contains thousands of computational cores. It is a challenge to develop methods that are able to make efficient use of all these cores. Methods developed for the single and dual core chips are often not applicable when going to many-core chips. This requires a radically different way of thinking and designing methods that will be able to scale to thousands of cores. It does not matter if these cores are on a Graphics Processing Unit or will be on a Central Processing Unit. In both cases you have to design methods that are inherently parallel or you will not be able to benefit from future hardware improvements.

That's what the work in this thesis is about, it's about using current and future computational resources as efficient as possible and thereby allowing us, and other researchers, to help explain the questions about our daily life and our childhood dreams.

Based on:
Jeroen Bédorf and Simon Portegies Zwart
The European Physical Journal Special Topics, Volume 210, 2012, pp.201-216, August 2012.

1.1 Introduction

This thesis is focussed on N-body methods that use direct N-body and hierarchical tree-code algorithms on Graphics Processing Units (GPUs) for astronomical simulations. In this introduction we give an overview of the hardware and software developments, since the 1960s, which formed the preamble of the works in this thesis. We will not cover developments in cosmological simulations. Despite this being one of the computationally most demanding branches of N-body simulations, the GPU usage is negligible. Although GPUs are used for cross-correlation of radio telescope data, we do not discuss this either, for that we refer the reader to the following works, e.g. Clark et al. (2013); Hassan et al. (2013); Fluke (2012) and references therein. Other reviews that cover N-body subjects that we do not discuss here are those by Dehnen and Read (2011) with a focus on methods and algorithms for N-body simulations as well as the work of Yokota and Barba (2012) which especially focus on Fast Multipole Methods and their implementation on GPUs. The introduction is ordered chronologically and makes a distinction between the collisional direct N-body methods (highly accurate and computationally the most expensive) and the collisionless tree-code methods (approximation based and therefore less computational expensive). This is similar to how the thesis chapters are divided with Chapter 2 discussing direct N-body tools, while Chapters 3, 4 and 6 cover collionless hierarchical algorithms.

As this is not the first work that tries to give an overview of the developments in the N-body field we built upon previous work of other authors. For the direct simulations we partially follow the papers discussed by Hut (2010) and Heggie and Hut (2003) as being noteworthy simulations since the 1960s. The papers and the number of bodies used in those simulations are presented in Fig. 1.1 (adapted from Hut (2010); Heggie and Hut (2003)); in the figure we show the number of bodies used and Moore's law which is a rough indication of the speed increase of computer chips (Moore 1965)[1]. Since direct N-body methods scale as $\mathcal{O}(N^2)$ it is understandable that the number of bodies used does not follow Moore's law in Fig. 1.1. However, in Fig. 1.2 we show that the increase in the number of bodies is faster than would be explainable by increase in computer speed alone. In this figure we show the theoretical number of operations, which is N^2 in the naive situation, and the number of transistors which is an indication of the speed of the computer. Both lines are normalized to the 1963 values. If the increase in N was solely based on the increase in computer speed, which doubles every 18 months according to Moore's law, the line would be horizontal and equal to 1. If the line is above 1 than Moore's law alone cannot be the cause of the increase in number of bodies. Therefore the increase must come from changes in hardware and software that are not directly related to the number of transistors. In the figure we tried to indicate (with arrows) what the major reasons for improvements were. The software improvements are covered in more detail in Section 1.3 and in Section 1.5 the improvements in hardware are discussed. With the context set we continue the introduction and start our historical overview in the early 1940s, before the era of digital computers.

---

[1] Technically Moore does not describe the speed of the computer, but the number of transistors. In practice the speed of a computer chip is roughly related to the number of transistors.

Figure 1.1: Number of particles used in collisional simulations over the last 4 decades. The solid line shows Moore's law Moore (1965), the circles publications and the dashed-line a fit through the data points.(Adapted from Hut (2010); Heggie and Hut (2003)).

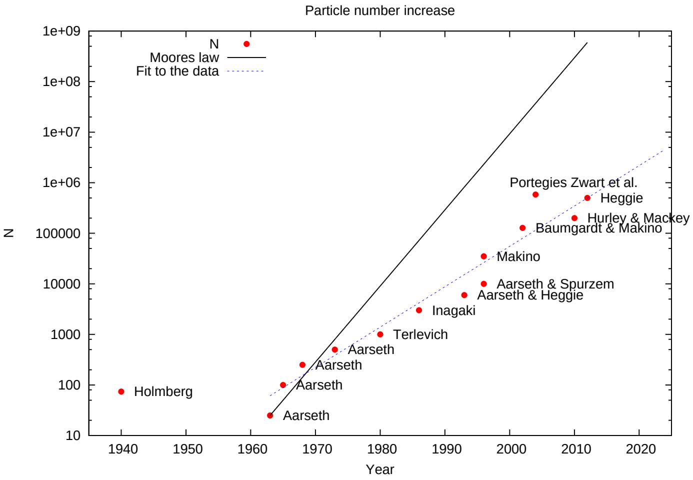
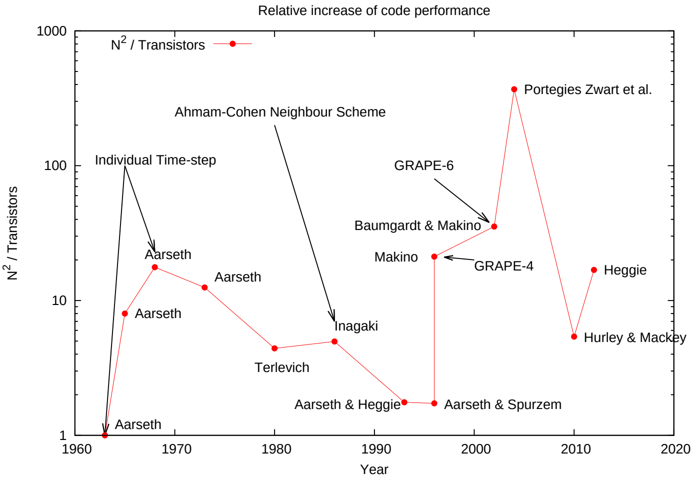


Figure 1.2: The number of theoretical operations, N^2, divided by the number of transistors determined using Moore's law normalized to 1963.



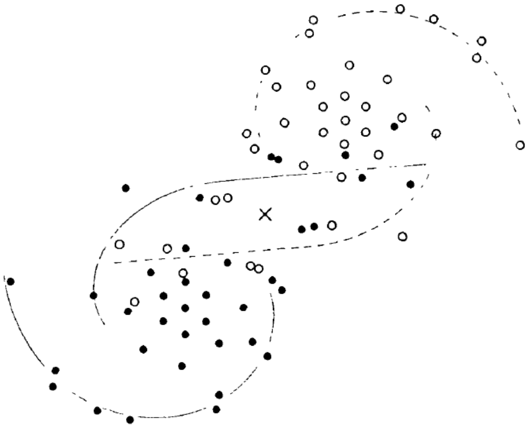


Figure 1.3: Spiral arms formed in the experiments by Holmberg in 1941. Image taken from Holmberg (1941)

1.2 The very beginning

Before the first computer simulations the Swedish astronomer Erik Holmberg, Holmberg (1941), published simulations of two interacting galaxies which were conducted using light bulbs. In his experiment each galaxy was represented by 37 light bulbs. Holmberg then measured the brightness of the light bulbs, which falls off with $\frac{1}{r^2}$, to compute the gravitational forces and let the galaxies evolve. In Fig. 1.3 we show one of his results where spiral arms develop, because of the interactions between two galaxies. This experiment was specifically tailored for one problem, namely gravitational interaction, which made it difficult to repeat using other more general hardware available at the time. So, even though it took a lot of manual labour, it would take almost 20 years before digital computers were powerful enough to perform simulations of comparable size and speed. This is the advantage of tailoring the hardware to the specific problem requirements. 50 years later we see the same advantage with the introduction of the special purpose GRAPE hardware (see Section 1.5).

1.3 1960 - 1986: The Era Of Digital Computers

The introduction of general purpose digital computers in the 1960s made it easier to buy and use a computer to perform simulations of N-body systems. Digital computers were based on transistors instead of vacuum tubes which made them cheaper to produce and maintain. The first computer simulations of astrophysical N-body systems where performed by von Hoerner in 1960, Aarseth in 1963 and van Albada in 1968 (von Hoerner 1960; Aarseth 1963; van Albada 1968). The number of bodies involved in these simulations was still relatively small and comparable to the experiment of Holmberg ($N = 10$ to

100).

During the first decades of the digital computer there were two ways to increase the number of bodies. One method was to buy a faster computer which allowed you to keep using the same software but increase the number of particles. This was an efficient method in the sense that the speed of the computer doubled roughly every 18 months, following Moore's Law (Moore 1965). Another method to increase N was by improving the software either by code optimizations or by algorithmic changes. In direct N-body integrations the number of required interactions scales as N^2 so any improvement to reduce the number of operations is very welcome.

In 1963 Aarseth introduced the individual time-step scheme (Aarseth 1963). To simulate an N-body system, the orbits of particles have to be followed exactly. However, particles isolated in space do not have sudden changes in their orbit and therefore can take longer time-steps than particles in the core of the cluster or which are part of a binary. Particles forming a binary change their positions quickly and therefore require many more time-steps to be tracked accurately. This is the basic idea behind the individual time-step scheme — each particle is assigned a simulation time when it is required to update and recompute the gravitational force. When only a few particles take small time-steps, then for most of the particles the gravitational forces do not have to be computed. The number of operations in a shared time-step scheme is then reduced from N^2 to $N \cdot N_{\text{active}}$ where N_{active} is the number of particles that have to be updated. If N_{active} is sufficiently small then the overhead of keeping track of the required time-steps is negligible compared to the gain in speed by not having to compute the gravitational forces for all bodies in the system.

The Ahmam-Cohen Neighbour Scheme (ACS) introduced in 1973 by Ahmad and Cohen (1973) takes another approach to reduce the number of required computations. In ACS the gravitational force computation is split into two parts. In the first part the force between a particle and its nearest neighbours, N_{nn}, (hence the name) is computed in a way similar to the direct N-body scheme with many small time-steps, because of the fast changing dynamical nearby neighbourhood. In the second part the force from the particles that are further away is updated less frequently, since the changes to that part of the gravitational force are less significant. When $N_{\text{nn}} \ll N$ the number of total interactions is reduced dramatically and thereby the total wall-clock time required for the simulation is reduced.

With the introduction of the digital computer also came the introduction of parallel computing. We can distinguish fine grained and coarse grained parallelisation. The former focuses on tasks that require many communication steps whereas the latter splits the computational domain and distributes it among different processors. These processors can be in the same machine or connected by a network. When connected by a network the communication is slower and therefore only beneficial if the amount of communication is minimal. In the early years of computing the focus was on fine grained parallelism using vector instructions. These instructions helped to increase the number of bodies in the simulations, but still N increased much slower than the theoretical speed of the processors. This is because of the N^2 scaling of direct N-body algorithms. Some of the noteworthy publications were the simulation of open clusters containing 1000 stars by Terlevich (1980) and the simulation of globular clusters using up to 3000 particles by Inagaki (1986) using the (at the time) commonly used `NBODY5` code.

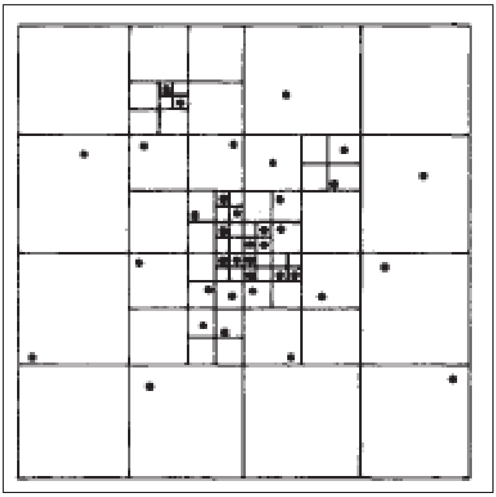


Figure 1.4: Particles grouped together in boxes in the tree-code algorithm. Image taken from Barnes and Hut (1986)

1.4 1986 - 2000 : Advances in software

In 1986 Barnes & Hut introduced their collisionless approximation scheme based on a hierarchical data structure, which became known as the Barnes-Hut tree-code (Barnes and Hut 1986). In this hierarchical data structure (tree) the particles are grouped together in boxes (see for an example Fig. 1.4). These boxes get the combined properties of the underlying particle distribution, like center of mass and total mass. To compute the gravitational force on a particle one does not compute the force between that particle and all other particles in the system, but rather between the particle and a selection of particles and boxes. This selection is determined by traversing the tree-structure and per box deciding if the particle is distant enough or whether the box lies so close that we should use the particles that belong to the box. This decision is made by a 'Multipole Acceptance Criterion' which, in combination with a free parameter (usually refered to as θ), is used to get the required precision. In this way one can either get high precision at high computational costs, by using more particles than boxes, or the other way around, lower precision by using more boxes instead of particles. The resulting code that implements this algorithm generally achieves a scaling of $\mathcal{O}(N \log N)$ instead of the $\mathcal{O}(N^2)$ of direct N-body codes. This allowed for simulations containing more particles by orders of magnitude than in direct N-body simulations, but came at the cost of reduced accuracy. Therefore, if high accuracy was required, researchers kept using collisional methods, for which algorithmic and computational developments continued.

The individual time-step method reduced the total number of executed gravitational force computations, since particles are only updated when required. However, if you would use the individual time-step method in a predictor-corrector integration scheme[2] you

---

[2] In a predictor-corrector scheme the positions are updated in multiple steps. First you predict the new positions using the original computed gravitational forces, then you compute the new gravitational forces using these positions and then you apply a correction to the predicted positions.

would have to predict all N particles to the new time while only computing the gravitational force for one particle. This prediction step results in a large overhead and the possibilities to parallelise the algorithm are limited, since the gravitational force is computed for only one particle. In a shared time-step method, there are N particles for which the force is computed, which can then be divided over multiple processors. A solution came in the form of the block time-step method in which particles with similar time-steps are grouped together. These groups are then updated using a time-step that was suitable for each particle in the group. Since multiple particles are updated at the same time, the number of prediction steps is reduced and the amount of parallel work is increased, see McMillan (1986). This is an example of i-particle parallelisation in which all j-particles are copied to all nodes and each node works on a subset of the i-particles. The i-particles are the sinks and the j-particles are the sources for the gravitational forces. In hindsight, it might have been more efficient to use j-particle parallelisation in which each processing node would get a part of the total particle set. The i-particles that have to be updated during a time-step are then broadcast to each node. The nodes then compute the gravitational force on those i-particles using their subset of j-particles and finally in a reduction step these partial forces are combined. With the introduction of the GRAPE hardware a few years later (see below), it turned out that this i-particle parallelisation was ideal for special purpose hardware. We will see more about this i-particle parallelisation in combination with GPUs in Chapter 2.

Other noteworthy publications in this time period are the following works. Aarseth and Heggie (1993) published the results of a 6000 body simulation containing primordial binaries and unequal mass particles. With this simulation it was possible to improve on previous results where only equal mass particles were used. The differences in the unequal mass and equal mass simulations were small, but too large to be ignored, which shows the critical importance of binaries and initial mass functions even though they are computationally expensive. Spurzem and Aarseth (1996) performed a simulation of a star cluster with 10^4 particles. The simulation was executed on a CRAY machine. This is one of the last simulations in our review that was executed without any special purpose hardware, since in the same year, Jun Makino presented his work which used three times more particles and was executed on GRAPE hardware (Makino 1996).

1.5 2000 - 2006: The Era Of The GRAPE

1.5.1 GRAPE

The introduction of the special purpose GRAvity PipE (GRAPE) hardware caused a breakthrough in direct-summation N-body simulations (see the book by Makino and Taiji (1998)). The GRAPE chips have the gravitational force calculations implemented in hardware which results in a speed-up of two orders of magnitude compared to the standard software implementations. The GRAPE chips were introduced in the early 1990s (Fukushige et al. 1991), but it would take a few years and development cycles before they were widely accepted and being used in production simulations. The GRAPE chips are placed on a PCI-expansion card that can be installed in any general purpose (desktop)

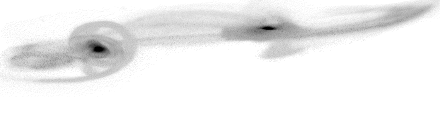
computer. The GRAPE came with a set of software libraries that made it relatively easy to add GRAPE support to existing simulation software like NBODY4 (Aarseth 1999) and Starlab (Portegies Zwart et al. 2001). The block time-stepping scheme introduced a few years earlier turned out to be ideal for this hardware and when multiple GRAPE chips were used one could combine this in $i - j$-particle parallelisation.

In the early 90s the large computational cost of direct N-body simulations had caused researchers to start using collisionless codes like the Barnes-Hut tree-code (Section 1.5.3) in order to do large N simulations. The introduction of the GRAPE combined with the availability of ready-to-use software caused the opposite effect, since suddenly it was possible to do collisional simulations at the same speed as collisionless simulations. The last generation of the fixed function GRAPE hardware was the GRAPE-6 chip. These were the most commonly used GRAPE chips, and when placed on a GRAPE-6Af extension board they reached a peak performance of $\sim$131 GFLOPs and enough memory to store 128k particles.

The GRAPE hardware is designed to offer large amounts of fine grained parallelism, since the on-chip communication is fast and specifically designed for the gravity computations. Supercomputers, on the other hand, are designed for coarse grained parallelisation thereby offering a large amount of computational cores connected using fast networks. But the communication times are still orders of magnitude slower than on-chip communication networks. This means that supercomputers are rarely used for direct N-body simulations and are much more suitable for collisionless simulations which require more memory and usually take larger time-steps (see Section 1.5.3). It would require hundreds of normal processor cores to reach the same performance as the GRAPE offers on one extension board, and that is even without taking into account the required communication time. If this is taken into account then the execution time on supercomputers is unrealistically high for high precision (e.g. many small time-steps with few active particles) direct N-body simulations

One of the limitations of the GRAPE is its fixed function pipeline and because of this it can not be used for anything other than gravity computations. For example in the ACS (Ahmad and Cohen 1973) the force computation is split into a near and a far force. The GRAPE cards are suitable to speed-up the computation of the far force, but the near force has to be computed on the host since the GRAPE has no facility to compute the force using only a certain number of neighbours. An alternative like Field Programmable Gate Arrays (FPGAs) allows for more flexibility while still offering high performance at low energy cost, since the hardware can be programmed to match the required computations. The programming is complex, but the benefits can be high since the required power is usually much less than a general purpose CPU cluster offering similar performance. An example of FPGA cards are the MPRACE cards (Spurzem et al. 2007), which are designed to speed-up the computation of the neighbour forces and thereby eliminating the need to compute the near force on the host computer which would become a bottleneck if only GRAPE cards would be used.

With the increasing availability of the GRAPE hardware at different institutes came the possibility to combine multiple GRAPE clusters for large simulations. A prime example of this is the work by Harfst et al. (2007) who used two parallel supercomputers which were equipped with GRAPE-6A cards. They showed that for direct N-body sim-



ulations it was possible to reach a parallel efficiency of more than 60%, and reached over 3 Teraflop (TFLOP) of computational speed when integrating 2×10^6 particles. Though the number of GRAPE devices was increasing it was still only a very small fraction of the number of "normal" PCs that was available. In order to use those machines efficiently one had to combine them and run the code in parallel. An example of this is the parallelisation of the N-body integrator in the `Starlab` package (Portegies Zwart et al. 2008). This work showed that it was possible to run parallel N-body simulations over multiple computers, although it was difficult to get good enough scaling in order to compete with the GRAPE hardware. This was also observed in earlier work by Gualandris et al. (2007), who developed different parallel schemes for N-body simulations thereby observing that the communication time would become a bottleneck for simulations of galaxy sized systems. Another approach is the work by Dorband et al. (2003) in which they implemented a parallel scheme that uses non-blocking communication. They called this a systolic algorithm, since the data rhythmically passes through a network of processors. Using this method they were able to simulate 10^6 particles using direct N-body methods.

1.5.2 Multi-Core Processors and Vector Instructions

In the 2000s it became clear that parallelisation had become one of the requirements to be able to continue increasing the number of particles. The clock speed of the CPU came near the physical limits of the used components and the chips used so much energy that the produced heat became a serious problem. This forced hardware manufacturers to shift their focus from increasing the clockspeed to increasing the number of CPU cores and to increase the amount of parallelisation inside the CPU cores with the introduction of special vector instructions. The Streaming SIMD Extensions (SSE) vector instructions in modern day processors promised to give a performance boost for optimized code. However, this optimization step required deep technical knowledge of the processor architecture. With the introduction of the Phantom GRAPE library by Nitadori et al. (2006) it became possible to benefit from these instructions without having to write the code yourself. As the name suggests, the library is compatible with software written for GRAPE hardware, but instead executes the code on the host processor using the special vector instructions for increased performance. Recently this is extended with the new Advanced Vector eXtensions (AVX) which allow for even higher performance on the latest generation of CPUs (Tanikawa et al. (2012, 2013)). Since at the same time the number of cores in the processors is increasing it helps that these libraries are able to use all the available CPU cores using multithreading. This has the potential to double the performance of the library when going, for example, from a single core to a double core processor. The actual performance gain depends on the type of simulation that is being executed. Often a combination of fine grained (vector instructions) and coarse grained (multi-core or even multiple-machine) parallelisation is used for direct N-body simulations with individual or block time-steps . The best solution depends on properties of the simulation, for example the total number of particles or the time-steps. When the number of particles that have to take a gravity step is small it is more efficient to not use the external network, but rather let all the work be handled by a single machine or single core. On the other hand if the number of particles taking a gravity step is large it could be more efficient to distribute the

work over multiple machines. Therefore the most optimal choice highly depends on the number of particles and the required accuracy / time-step.

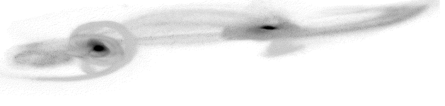
1.5.3 Collisionless methods

Collisionless simulations scale as $\mathcal{O}(N \log N)$ when using tree methods or with $\mathcal{O}(N)$ when using the Fast Multipole Moment (FMM) method (Dehnen 2002; Yokota and Barba 2012; Yokota et al. 2011). Because of this scaling it is possible to perform large collisionless simulations ($N > 10^7$) while collisional methods are generally limited to $N = 10^5 \sim 10^6$. These large simulations require a combination of memory and computational resources and are therefore often executed on supercomputers. On the other hand collisional simulations are usually only limited by computational power. In the collionless simulations the gravity computation still forms a major part of the total computation time and it is therefore beneficial to execute these computations using GRAPE hardware. By modifying the tree-walk and using a special version of the GRAPE chip Fukushige et al. (1991) were able to execute the computation of the gravitational force of the Barnes & Hut tree-code algorithm using the GRAPE hardware, thereby benefiting from both the fast tree-code algorithm and the efficiency of the GRAPE. Since only the computation of the gravitational forces was accelerated, the limited amount of memory on the GRAPE hardware did not pose a serious limitation. Over the years this technique has been applied on various generations of the GRAPE hardware, for example by Athanassoula et al. (1998) and Kawai et al. (2000).

1.6 2006 - Today: The Era Of Commercial High Performance Processing Units

1.6.1 Collisional methods

In 2001 programmable Graphics Processing Units (GPUs) were introduced. However, it would take another 7 years before GPUs were powerful enough to be a viable alternative to the single function GRAPE that dominated the N-body simulation field over the previous decade. The GPU was originally designed to improve the rendering speed of computer games. However, over the years these cards became progressively faster and, more importantly, they became programmable (see for more details also Section 1.7). At first one had to use programming languages which were specially designed for the creation of visual effects (e.g. `Cg` and `OpenGL`). The first use of the GPU for N-body simulations was by Nyland et al. (2004) who used `Cg`. Their implementation was mostly a proof-of-concept and lacked advanced time-stepping and higher order integrations which made it unsuitable for production quality N-body simulations. Programming GPUs became somewhat easier with the introduction of the `BrookGPU` (Buck et al. 2004a) programming language. This language is designed to be a high level programming language and compiles to the `Cg` language. Elsen et al. (2006) presented an N-body implementation using the BrookGPU language. Around the same time Portegies Zwart et al. published an implementation of a higher order N-body integration code with block time-steps written in `Cg` (Portegies



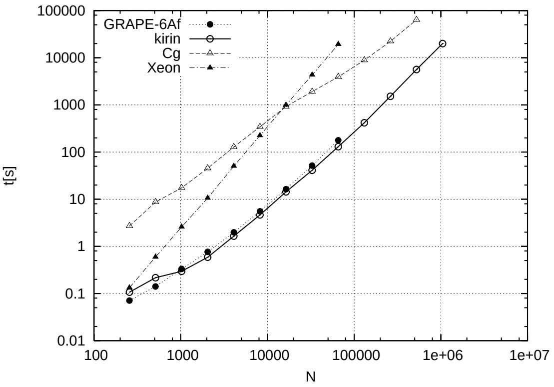


Figure 1.5: Performance comparison of N-body implementations. The CUDA GPU implementation `kirin` is represented by the solid line (open circles). The GRAPE is represented as the dotted line (bullets). The Cg GPU implementation is represented as the dashed line (open triangles). The dashed-dotted line (closed triangles) represent the results on the host computer. Figure taken from Belleman et al. (2008).

Zwart et al. 2007). Although these publications showed the applicability and power of the GPU, the actual programming was still a complicated endeavor. This changed with the introduction of the Compute Unified Device Architecture (CUDA) programming language by NVIDIA in early 2007. The language and compatible GPUs were specifically designed to let the power of the GPU be harvested in areas other than computer graphics. Shortly after the public release of CUDA efficient implementations of the N-body problem were presented by Hamada and Iitaka (2007); Belleman et al. (2008); Nyland et al. (2007). Belleman et al. (2008) showed that it was possible to use the GPU with CUDA for high order N-body methods and block-time steps with an efficiency comparable to the, until then, unbeaten GRAPE hardware (see Fig 1.5).

As an alternative to the proprietary CUDA language, the Khronos group[3] introduced in 2009 the OpenCL programming language. This language is designed to create parallel applications similar to the way CUDA is used for GPUs, with the difference that programs written in OpenCL also work on systems with only CPUs. The idea behind this is that the developer has only to write and maintain one program. In reality, however, the developer will have to write code that is optimized for one platform (GPU or CPU) in order to get the highest performance out of that platform. Because CUDA was released a couple of years earlier, has more advanced features and has more supported libraries than OpenCL are the reasons CUDA is currently more commonly used than OpenCL. However, there is no reason this cannot change in the future with updated libraries that offer OpenCL support (e.g. `Sapporo2` see below and Chapter 2).

With GRAPE having been around for over 15 years, most of the production quality astrophysical N-body simulation codes were using GRAPEs when NVIDIA released CUDA. Sapporo,the GRAPE compatible library, made the shift from the GRAPE framework to the GPU easy Gaburov et al. (2009). This library uses double-single precision[4] and on-device execution of the prediction step, just like the GRAPE-6Af hardware. Because GPUs are cheaper to buy, have more onboard memory, higher computational speed and

---

[3] The Khronos Group is a group of companies that develops open standards for accelerating graphics, media and parallel computations on a variety of platforms.

[4] This technique gives precision up to the 14th significant bit while using single precision arithmetic.

the option to reprogram them to your specific needs, nowadays more GPUs than GRAPEs are used for N-body simulations. Even though the GRAPE, because of its dedicated design, requires less power than the GPU. The GRAPE-DR (Greatly Reduced Array of Processor Elements with Data Reduction; Makino et al. (2007)) is different from the earlier generation GRAPE chips since it does not have the gravitational computations programmed in hardware, but rather consists of a set of programmable processors. This design is similar to how the GPU is built up, but uses less power since it does not have the overhead of graphics tasks and visual output that GPUs have. At the time of its release in 2007 the GRAPE-DR was about two times faster for direct N-body simulations than the GPU.

Nitadori and Aarseth (2012) describe their optimisations to simulation codes NBODY6 and NBODY7 (Aarseth 2012) to make use of the GPU. They have tested which parts of the code are most suitable to be executed on the GPU and came to the conclusion that it was most efficient to execute the so-called 'regular force' on the GPU. This step involves around 99 percent of the number of particles. On the other hand the local force is executed on the host using vector instructions, since using the GPU for this step resulted in a communication overhead which is too large[5]. In this division the bulk of the work is executed on the GPU and the part of the algorithm that requires high precision, complex operations or irregular memory operations is executed on the host machine possibly with the use of special vector instructions. This is a trend we see in many fields where the GPU is used. However, some authors overcome the communication overhead by implementing more methods on the GPU besides the force computation, see for example Spurzem et al. (2011). The two most recent large N simulations have been performed by Hurley and Mackey (2010) ($N = 10^5$) and Heggie ($N = 5 \times 10^5$; private communication, not yet published) who both used NBODY6 in combination with one or more GPUs.

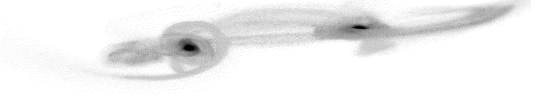
A subfield of direct N-body methods are simulations that are used to determine planetary stability, which usually involve only a few particles. This severely limits the amount of parallelism and therefore a different approach has to be taken than that used in large N simulations. `Swarm-NG` (Dindar et al. 2013) is an example of an implementation that takes a different approach. This is a software package for integrating ensembles of few-body planetary systems in parallel with a GPU. `Swarm-NG` is specifically designed for low-N systems. Instead of breaking up one problem into parallel tasks, `Swarm-NG` integrates thousands of few-body systems in parallel. This makes it especially suited for Monte Carlo-type simulations where the same problem is run multiple times with varying initial conditions.

1.6.2 Collisionless methods

The first result of a tree-code accelerated by a GPU was presented in Belleman et al. (2008). The results did show a speed-up compared to the CPU results, however the speed-up was smaller than observed when using direct N-body methods. This is of course understandable since there are fewer force computations that can benefit from the GPU compared

---

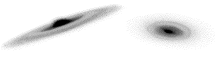
[5] This is similar to how the MPRACE project did the division between the GRAPE and MPRACE cards.



to direct N-body methods. Another limiting factor is the amount of communication required between the GPU and CPU in the standard GRAPE tree-code implementations. The high computational speed of the GPU means that communication can become a bottleneck. In the award-winning papers by Hamada et al. (2009c) and Hamada and Nitadori (2010) this overhead is reduced by combining the results of multiple tree-walks (executed on the CPU), which are then transferred to the GPU in a single data transfer. Instead of receiving one set of interaction lists the GPU now receives multiple sets, which increases the amount of parallel work to be executed and improves the overall efficiency of the GPU. However, even with this method the tree-walk itself is executed on the CPU and only the force computation is executed on the GPU.

1.7 Graphics Processing Units

Since the GPU plays such a pivotal role in this thesis we give a short overview of what it is and what sets it apart from the CPU. The GPU is a computer chip that is placed, together with a certain amount of memory, on a circuit board. This board can then be plugged into the PCI-Express bus. The chip, memory and board are as a set generally referred to as the graphics card. The GPU is specifically designed to deliver high performance for parallel tasks. It was originally designed this way so that it could render the frames of computer games as fast as possible. For these frames the GPU has to compute, for each pixel independently, which colour should be sent to the display. On an average computer display there are over a million pixels which have to be updated thirty times per second to create the feeling of smooth gameplay. To compute the correct colour for each pixel the GPU has to perform a series of (relatively) simple operations. The input for these operations are data-objects stored in the GPU memory and which represent objects in the 3D game world. The game programmer writes a set of programs, usually called shaders, that combine the effects of various light sources and material choices into a final pixel colour. These shaders can be compared to the kernels that we write for scientific applications. The kernels use some data as input, perform a set of computational operations and write the result back to GPU memory. Each of these shaders performs the same operations for a different pixel, which allows for the massive amounts of parallelism. In order to manage all these threads the GPU contains a large amount of compute cores. These cores are placed on a set of multi-processors and the performance of the GPU depends on the number of multi-processors and inherently the number of compute cores. See Fig. 1.6 for an example of such a multi-processor as used in the NVIDIA `Kepler` chips. In order to feed all these compute cores with data there is a high bandwidth memory bus (>100GB/s) that connects the multiprocessors with the onboard memory. This way the data streams from memory, through the compute chips and then back to the memory buffers. To manage all this parallelism the GPU uses a programming model which is commonly referred to as Single Instruction, Multiple-Thread (SIMT). In this model you write a single program that performs the same set of operations for each thread that is launched on the GPU. However, since the input data can depend on the unique identifier of each thread you can use different input data per thread. Usually the threads are grouped together in groups of a certain size. On the NVIDIA hardware these groups are called *warps* and contain 32



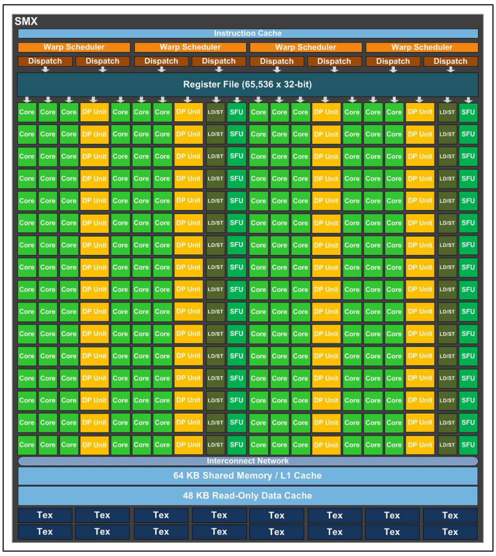
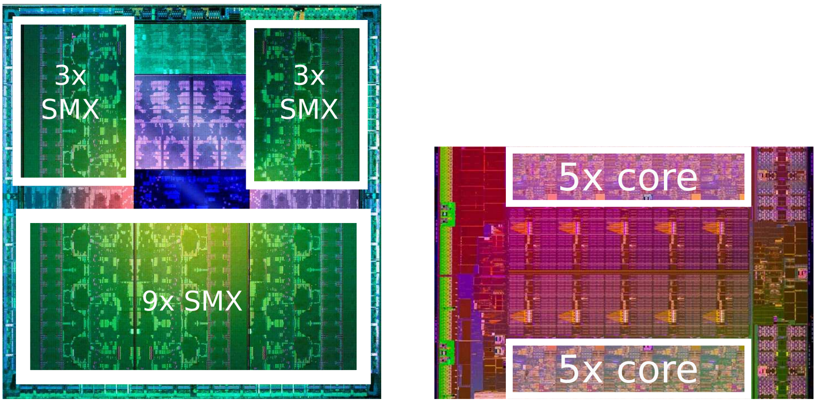


Figure 1.6: Kepler SMX: 192 single precision CUDA cores, 64 double precision units, 32 special function units (SFU), and 32 load/store units (LD/ST). Figure taken from (NVIDIA (2013b))

threads. On the AMD hardware the groups are called *wavefronts* and contain 64 threads. In Chapter 2 we will see how these groups and their sizes can be important for the efficiency of the code.

In general the compute units on the GPU are much less complex than those on the CPU. The compute units on the CPU are, for example, able to execute vector instructions, use out-of-order execution and perform branch prediction. These abilities make the CPU very fast when executing serial operations, but these abilities also take up extra space on the chip. Because all this advanced logic is implemented in hardware there is less space available for the actual compute units. For the GPU this is the other way around. The compute units are relatively simple, do not contain advanced branch prediction or perform out-of-order execution nor do they support vector instructions. Another difference is that GPUs contain much smaller cache sizes (both L1 and L2) than modern day CPUs. As a result, the cores take up less space and is it possible to add more of them on the same chip. See Fig. 1.7 for an example of the differences between a GPU and CPU chip. So, even though the clock speed of these cores is lower than the clock speed of the CPU, the fact that there are more of them allows the GPU to reach higher peak-performance. Therefore, if you have a problem that can be efficiently parallelised it might be possible to execute it efficiently on the GPU. However, if the problem can not be parallelised you will not gain any performance and will only slow down the execution when using a GPU. It is therefore a challenge to subdivide your problems, data and algorithms in as many (independent) parts as possible to create an effective GPU implementation.

Over the years the GPU is slowly evolving from a single function compute unit into a more advanced device. First it was only possible to execute single precision computations, but with the addition of IEEE-754 double precision compute units it became possible to use the GPU for codes that require double precision computations. An other example of new features is the way in which the compute work is launched. Traditionally the CPU



(a) `Kepler` **(b)** `Xeon`

Figure 1.7: Photos of two different chip architectures. The left picture shows the NVIDIA `Kepler GK110` chip, which contains 7.1 billion transistors and has a size of 561mm^2. The right picture shows the Intel `Xeon E5-2600 V2` chip, which contains 4.3 billion transistors and has a size of 542mm^2. The NVIDIA chip contains 15 SMX units each with 192 cores (see Fig. 1.6) and the Intel chip contains 10 cores. The marked areas roughly indicate which parts of the chip are used for compute and which parts are used for caches and control. Note the figures are not to scale and do not use the same scale.

tells the GPU to execute compute kernels, but in the latest generation of NVIDIA GPUs (`Kepler`) the GPU can launch its own kernels. This allows for new GPU implementations that are based on recursion. In the individual chapters of this thesis we will highlight some further features of the GPU that are specific to the methods and algorithms used in those chapters.

1.8 Thesis Overview

The previous paragraphs detailed the developments that eventually led to the works in this thesis. In this thesis we continue these developments and improve on previous work by either expanding existing abilities or radically redesigning algorithms in the quest for more performance and higher efficiency when running N-body simulations. The main aim of increasing the performance is to enable N-body simulations which are not feasible with current day simulation codes. A faster code therefore allows us to pursue new scientific topics and arrive at answers to open questions. Simulations that took months or even years to complete will instead finish in a couple of days or a few weeks at most. This short time-to-solution allows us to execute more simulations and thereby cover a wider parameter space. This in turn gives a better sense of the effect these parameters have and thereby improves the research quality. An example of this can be found in Chapter 5.

Currently, many models in astrophysics are necessarily simulated with a number of bodies that is much smaller than the actual number of bodies in star clusters and galaxies. This can provide a good approximation of what a simulation with a larger model would be like, however, it can be unclear if the result is due to a physical cause or if it is an artificial

consequence of the low resolution and simplified simulation. Or, for example, when the small-scale structure as seen in observations cannot be resolved in simulations, because of limited resolution, it is impossible to make a direct comparison between theory and observation. With a more efficient simulation code one can opt to simulate more detailed models instead of executing simulations faster. The simulation would take the same amount of time as before, but can contain orders of magnitude more particles. With such a large number of particles it is, for example, possible to do a galaxy simulation where the digital masses of the particles are comparable to those of the stars in a physical galaxy. A prime example of this can be found in Chapter 6 where we describe a method that is able to simulate the Milky Way Galaxy on a star-by-star basis.

1.8.1 Chapter 2 - Sapporo2

In this chapter we present the new direct N-body library `Sapporo2`. This is a new version of the commonly used `Sapporo` library. In this version we have expanded the abilities of the library to include support for more hardware devices, different orders of integration and adjustable numerical precision. We present the performance of the library on hardware accelerators (GPUs) of different vendors and how the numerical accuracy influences the execution time. Finally we present performance difference between using multiple CPU cores and a single GPU. Because the library exploits the high performance of GPUs it enables us to increase the number of bodies used in simulations and thereby allowing for more realistic models. Direct N-body methods are generally used when accuracy is an important requirement. Therefore, having the option to use double precision or using a sixth-order integrator instead of a fourth-order means that `Sapporo2` can be used for a wider range of problems. For example, the large mass ratios in models with supermassive black-holes have to be solved with a sixth order integrator, otherwise the lack of precision would result in (near) zero time-steps.

1.8.2 Chapter 3 - OctGrav

In this chapter we introduce the `Octgrav` simulation code. This code uses the Barnes-Hut tree-code algorithm and the GPU to accelerate the execution. Previous tree-code methods have used the GPU to accelerate the gravity computation (see Section 1.6.2). However, in this new code we take it a step further and also execute the tree-walk on the GPU. In previous codes this was a serious bottleneck, since this data-intensive operation had to be executed on the CPU, which has an order of magnitude lower bandwidth than the GPU. Apart from the lower performance of the CPU, the resultant lists had to be transferred over the relatively slow PCI-Express bus. By executing this operation on the GPU we remove the need for data-transfer and benefit from the high on-device bandwidth of the GPU. In this chapter we show the performance gain by moving the tree-walk to the GPU and we identify which new bottlenecks surface after removing the traditional ones.

1.8.3 Chapter 4 - Bonsai

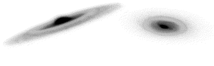
By removing the traditional bottlenecks of the tree-code algorithm in the previous chapter it became clear that the new `Octgrav` code was limited by other bottlenecks. The construction of the hierarchical data-structure, particle sorting or even just the prediction of the new particle positions took only a few percent of the execution time in the original CPU algorithm, but after speeding up the other parts in the GPU version the fast methods in the old CPU version suddenly started taking up a major fraction of the execution time. However, there is only so much one can do to optimize and resolve bottlenecks in an algorithm that scales as $\mathcal{O}(N)$. By implementing these on the GPU the algorithms retained their $\mathcal{O}(N)$ and $\mathcal{O}(N \log N)$ scaling, but profit from the high computational speed and bandwidth of the GPU. To prevent any further CPU limitations we took it to the final step and implemented all parts of the tree-code algorithm on the GPU. This eliminates the need to transfer large amounts of data between the CPU and GPU during each time-step. All these optimizations come together in the new GPU-enabled tree-code `Bonsai`. The chapter shows the performance of the individual parts of the algorithm and describes why the code works efficiently when using either a shared time-step or a block time-step method.

1.8.4 Chapter 5 - Many Minor Mergers

Observations of the early universe ($z \gtrsim 2$) show that there are galaxies which are both massive and very compact. However, these galaxies seem to have disappeared in the local universe. In this chapter we use `Bonsai` to test if this disappearance can be explained by growth caused by infalling smaller galaxies. The theory is that these infalling galaxies cause the compact galaxy to grow more in size than in mass. The efficiency of the code allows us to test this hypothesis by simulating a wide variety of gasless galaxy merger configuratons. We simulate mergers that have mass ratios between q =1:1 (equal mass) to q =1:160. Next we analyze the merger remnants and test which remnants have properties that can best reproduce the current day observations. From the results we can conclude that the observation can be explained by mergers that involved mass ratios of q = 1:5 — 1:10. Furthermore, the simulations indicate that mass ratios less than 1:10 can not be the cause, because in that case the growth in size would be larger than what is determined from the observational results.

1.8.5 Chapter 6 - Parallel Bonsai

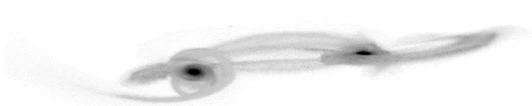
In chapter 4 we introduced the `Bonsai` code which gets its high efficiency and performance by limiting the amount of communication with the CPU and by keeping all data on the GPU. However, this means that the amount of memory on the GPU is a limitation when increasing the number of particles. This limitation can be overcome by using multiple GPUs, so that we not only gain extra memory but also extra performance. In this chapter we present the parallel version of `Bonsai` which uses the CPU to communicate with other GPUs. These GPUs can either be installed in the same system or in seperate systems that are connected with each other in a non-shared memory architecture. By overlapping the



(network) communication time with computations on the GPU we are able to hide most of the required communication overhead of the relatively slow PCI-bus and network cables. This allows us to scale the simulation code to thousands of GPUs, and execute simulations that contain not millions, but billions of particles. All of this while maintaining over 70% parallel efficiency. There are two main reasons to simulate these large models on parallel GPU systems. The first is of practical nature, with the advent of accelerator technology we see more and more supercomputers that get their performance from this new technology. Without a simulation code that can use accelerators (GPUs) you will not be able to get performance from, let alone access to, these systems. The second reason is that if we want to simulate models with billions of particles in reasonable time (a few weeks at most) we need high performance. With these models we can obtain more detailed results which are required if we want to keep up with observational results that pour in from new observatory platforms such as the Gaia satellite.

2 Sapporo2: A versatile direct N-body library

Astrophysical direct N-body methods have been one of the first production algorithms to be implemented using NVIDIAs `CUDA` architecture. Now, almost six years later, the GPU is the most used accelerator device in astronomy for simulating stellar systems. In this paper we present the implementation of the `Sapporo2` N-body library, which allows researchers to use the GPU for N-body simulations with little to no effort. The first version, released five years ago, is actively used, but lacks advanced features and versatility in used numerical precision and support for higher order integrators. In this updated version we have rebuilt the code from scratch and added support for `OpenCL`, multi-precision and higher order integrators. We show how to tune these codes for different GPU architectures and present how to continue utilizing the GPU as optimally as possible even when only a small number of particles is integrated. This careful tuning allows `Sapporo2` to be faster than `Sapporo1` even with the added options and double precision data loads. The code shows excellent scaling on a range of NVIDIA and AMD GPUs in single and double precision accuracy.

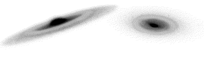


2.1 Background

The class of algorithms, commonly referred to as direct N-body algorithms is still one of the most commonly used methods for simulations in astrophysics. These algorithms are relative simple in concept, but can be applied to a wide range of problems. From the simulation of few body problems, such as planetary stability to star-clusters and even small scale galaxy simulations. However, these algorithms are also computationally expensive as they scale as O(N^2). This makes the method unsuitable for large N ($> 10^6$), for these large N simulations one usually resorts to a lower precision method like the Barnes-Hut tree-code method Barnes and Hut (1986) that scales as O($NlogN$) or the Particle Mesh method that scales as O(N) (e.g. Hohl and Hockney (1969); Hockney and Eastwood (1981)). These methods, although faster, are also notably less accurate and not suitable for simulations that rely on the high accuracy that direct integration offers. On the other end of the spectrum you can find even higher accuracy methods which use arbitrary precision (Boekholt et al. in prep). The results of Boekholt et al. indicate that the accuracy offered by the default (double precision) direct N-body methods is sufficient for most scientific problems.

The direct N-body algorithm is deceivingly simple, in the basic form it performs N^2 gravitational computations, which is an embarrassingly parallel problem that can be efficiently implemented on almost any computer architecture with a limited amount of code lines. A number of good examples can be found on the `Nbabel.org` website. This site contains examples of a simple N-body simulation code implemented in a wide range of programming languages. However, in practice there are many variations of the algorithms in use, with up to 8th order integrations Nitadori and Makino (2008), algorithmic extensions such as block-time stepping McMillan (1986), neighbour-schemes Ahmad and Cohen (1973), see Bédorf and Portegies Zwart (2012) and references therein for more examples. These variations transform the simple O(N^2) shared time-step implementation in one with many dependencies and where the amount of parallelism can differ per time-step. Especially the dynamic block time-stepping method adds complexity to the algorithm, since the number of particles that participate in the computations change with each integration step. This variable number of particles involved in the computations forces the use of different parallelisation strategies. In the worst case there is only one particle integrated which eliminates most of the standard parallelisation methods for N^2 algorithms. There is extensive literature on high performance direct N-body methods with the first being described in 1963 Aarseth (1963). The method has been efficiently implemented on parallel machines McMillan (1986), vector machines Hertz and McMillan (1988) and dedicated hardware such as the GRAPEs Makino and Taiji (1998). For an overview we refer the interested reader to the following reviews Bédorf and Portegies Zwart (2012); Heggie and Hut (2003); Dehnen and Read (2011)

In this paper we present our direct N-body library, `Sapporo2`. The library contains built-in support for second order leap-frog (GRAPE5), fourth order Hermite (GRAPE6) and the sixth order Hermite integrators. The numerical precision can be specified at run time and depends on requirements for performance and accuracy. Furthermore, the library can keep track of the nearest neighbours by returning a list containing all particles within a



certain radius. Depending on the available hardware the library operates with CUDA and OpenCL, and has the option to run on multiple-GPUs if installed in the same system. The library computes the gravitational force on particles that are integrated with block time-step algorithms. However, the library can trivially be applied to any other N^2 particle method by replacing the force equations.

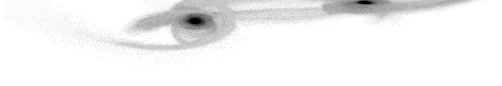
2.2 Methods

With Graphic Processing Units (GPUs) being readily available in the computational astrophysics community for over 5 years we will defer a full description of their specifics and peculiarities Bédorf and Portegies Zwart (2012); Belleman et al. (2008); Nyland et al. (2007); NVIDIA (2013a). Here we only give a short overview to stage the context for the following sections. In GPU enabled programs we distinguish two parts of code. The 'host' code, used to control the GPU, is executed on the CPU; whereas the 'device' code, performing the majority of the computations, is executed on the GPU. Each GPU consists of a set of multiprocessors and each of these multiprocessors contains a set of computational units. We send work to the GPU in blocks for further processing by the multiprocessors. In general a GPU requires a large amount of these blocks to saturate the device in order to hide most of the latencies that originate from communication with the off-chip memory. These blocks contain a number of threads that perform computations. These threads are grouped together in 'warps' for NVIDIA machines or 'wavefronts' on AMD machines. Threads that are grouped together share the same execution path and program counter. The smaller the number of threads that are grouped the smaller the impact of thread divergence. On current devices a warp consists of 32 threads and a wavefront contains 64 threads. This difference in size has an effect on the performance (see Section 2.3).

2.2.1 Parallelisation method

To solve the mutual forces for an N-body system the forces exerted by the j-particles (sources) onto the i-particles (sinks) have to be computed. Depending on the used algorithm the sources and sinks can either belong to the same or a completely different particle set. Neither is it required that these sets have the same dimensions. In worst case situations this algorithm scales as $O(N^2)$, but since each sink particle can be computed independently it is also embarrassingly parallel. The amount of parallelism however depends on the number of sink particles. For example, in high precision gravitational direct N-body algorithms that employ block-time stepping the number of sink particles ranges between 1 and N. In general the number of sinks is $\ll$ than the number of sources, because only the particles of which the position and velocity require an update are integrated McMillan (1986). As aconsequence the amount of available parallelism in this algorithm is very diverse, and depends directly on the number of active sink particles.

Currently there are two commonly used methods for solving N^2 like algorithms using GPUs. The first performs parallelisation over the sink particles Hamada and Iitaka (2007); Belleman et al. (2008); Nyland et al. (2007) which launches a separate compute thread for each sink particle. This is efficient when the number of sinks is large ($> 10^4$), because



then the number of compute threads is sufficiently high to saturate the GPU. However, when the number of sink particles is small ($\leqslant 10^4$) there are not enough active compute threads to hide the memory and instruction latencies. As a result the GPU will be under utilized and only reaches a fraction of the available peak performance. We expect that future devices require an even larger number of running threads to reach peak performance, in which case the number of sink particles has to be even larger to continuously saturate the device. However, adjusting the number of sink particles to keep parallel efficiency is not ideal, because then one artificially increases the amount of work (and runtime) in favor of efficiency. Therefore, a second method was introduced in `Sapporo1` Gaburov et al. (2009) which takes a slightly different approach. In `Sapporo1` we parallelize over the source particles and keep the number of sink particles that is concurrently integrated fixed to a certain number. The source particles are split into subsets, each of which forms the input against which a set of sink particles is integrated. The smaller the number of sink particles the more subsets of source particles we can make. It is possible to saturate the GPU with enough subsets, so if the combined number of sink and source particles is large enough[1] you can reach high performance even if the number of sinks or sources is small.

Of the two parallelisation methods the first one is most efficient when using a shared-time step algorithm, because fewer steps are involved in computing the gravity. However, `Sapporo1` is more suitable for block-time stepping methods commonly used in high precision gravitational N-body methods. Even though this method requires an extra step to combine the partial results from the different subsets. The `Sapporo1` method is also applied in this work. With `Sapporo1` being around for 5 years we completely rewrote it and renamed it to `Sapporo2` which is compatible with current hardware and is easy to tune and adapt to future generation accelerator devices and algorithms. The next set of paragraphs describe the implementation and the choices we made.

2.2.2 Implementation

CUDA and OpenCL

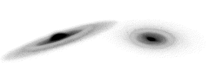
When NVIDIA introduced the CUDA framework in 2007 it came with compilers, runtime libraries and examples. CUDA is an extension to the 'C' programming language and as such came with language changes. These extensions are part of the device and, more importantly, part of the host code[2]. The use of these extensions requires that the host code is compiled using the compiler supplied by NVIDIA. With the introduction of the 'driver API' this was no longer required. The 'driver API' does not require modifications to the 'C' language for the host code. However, writing CUDA programs with the 'driver API' is more involved than with the 'runtime API', since actions that were previously done by the NVIDIA compiler now have to be performed by the programmer.

When the OpenCL programming language was introduced in 2009 it came with a set of extensions to the 'C' language to be used in the device code. There are no changes to the language used for writing the host code, instead OpenCL comes with a specification of

---

[1] The exact number required to reach peak performance depends on the used architecture, but if the total number of gravitational interactions is $\geqslant 10^6$ it is possible to saturate the GPU

[2] The most notable addition is the `<<<>>>' construction to start compute kernels.



functions to interact with the device. This specification is very similar to the specification used in the CUDA driver API and follows the same program flow.

In order to support both OpenCL and CUDA in `Sapporo2` we exploited the similarity between the CUDA driver API and the OpenCL API. We developed a set of `C++` classes on top of these APIs which offer an unified interface for the host code. The classes encapsulate a subset of the OpenCL and CUDA functions for creating device contexts, memory buffers (including functions to copy data) and kernel operations (loading, compiling, launching). Then depending on which class is included at compile time the code is executed using OpenCL or CUDA. The classes have no support for the more advanced CUDA features such as OpenGL and Direct3D interoperability.

Kernel-code With the wrapper classes the host-code is language independent. For the device code this is not the case, even though the languages are based on similar principles the support for advanced features like `C++` templates, printing and debugging functionality in CUDA makes it much more convenient to develop in pure CUDA. After that we port the working code to OpenCL. The use of templates in particular reduces the amount of code. In the CUDA version all possible kernel combinations are implemented using a single file with templates. For OpenCL a separate file has to be written for each combination of integrator and numerical precision.

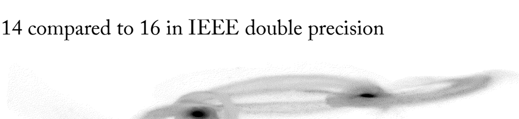
The method used to compute the gravitational forces is comparable to the method used in `Sapporo1` with only minor changes to allow for double precision data loads/stores and more efficient loop execution.

Numerical Accuracy

During the development of `Sapporo1` GPUs lacked support for IEEE-754 double precision computations and therefore all the compute work was done in either single or double-single precision[3]. The resulting force computation had similar precision as the, at that time, commonly used GRAPE hardware Makino and Taiji (1998); Gaburov et al. (2009). This level of accuracy is sufficient for the fourth order Hermite integration scheme Makino and Aarseth (1992); Portegies Zwart and Boekholt (2014). Currently, however there are integrators that accurately solve the equations of motions of stars around black-holes, planets around stars and similar systems that encounter high mass ratios. For these kind of simulations one often prefers IEEE-754 double precision to solve the equations of motion. The current generation of GPUs supports IEEE-754, which enables computations that require this high level of accuracy. Therefore the data in `Sapporo2` is always stored in double precision. The advantage of this is that we can easily add additional higher order integrators that require double precision accuracy computations without having to rewrite major parts of the host code. Examples of such integrators are the 6th and 8th order Hermite integrators Nitadori and Makino (2008). The performance impact of double precision storage on algorithms that do not require double precision computations is limited. Before the actual computations are executed the particle properties are converted to either float or double-single and the precision therefore does not influence the computational performance. The

---

[3] In this precision, the number of significant digits is 14 compared to 16 in IEEE double precision



penalty for loading and storing double the amount of data is relative small as can be seen in the result section where `Sapporo1` is compared to `Sapporo2`.

multiple GPUs

Our new N-body library can distribute the computational work over multiple GPUs, as long as they are installed in the same system. While in `Sapporo1` this was implemented using the `boost` threading library, this is now handled using `OpenMP`. The multi-GPU parallelisation is achieved by parallelisation over the source particles. In `Sapporo1` each GPU contained a copy of all source particles (as in Harfst et al. (2007)), but in `Sapporo2` the source particles are distributed over the used devices using the round-robin method. Each GPU now only holds a subset of the source particles which reduces memory requirements, transfer time and the time to execute the prediction step on the source particles. However, the order of the particle distribution and therefore the addition order is changed when comparing `Sapporo1` and `Sapporo2`. This in turn can lead to differences in the least significant digit when comparing the computed force of `Sapporo1` to `Sapporo2`.

Other differences

The final difference between `Sapporo1` and `Sapporo2` is the way the partial results of the parallelisation blocks are combined. `Sapporo1` contains two computational kernels to solve the gravitational forces. The first computes the partial forces for the individual blocks of source particles, and the second sums the partial results. With the use of atomic operators these two kernels can be combined, which reduces the complexity of maintaining two compute kernels when adding new functionality at a minimal performance impact. The expectation is that future devices require more active threads to saturate the GPU, but at the same time offer improved atomic performance. The single kernel method that we introduced here will automatically scale to future devices and offers less overhead than launching a separate reduction kernel.

2.3 Results

In astrophysics the current most commonly used integration method is the fourth order Hermite Makino and Aarseth (1992) which requires per particle the nearest neighbour and a list of neighbours within a certain radius. This is what `Sapporo1` computes and how the GRAPE hardware operates Makino and Taiji (1998). The used numerical precision in this method is the double-single variant. In order to compare the new implementation with the results of `Sapporo1`, all results in this section, unless indicated otherwise, refer to the double-single fourth order Hermite integrator.

For the performance tests we used different machines, depending on which GPU was used. All the machines with NVIDIA GPUs have `CUDA 5.5` toolkit and drivers installed. For the machine with the AMD card we used toolkit version 2.8.1.0 and driver version 13.4.

The full list of used GPUs can be found in Tab. 2.1, the table shows properties such as clock speed and number of cores. In order to compare the various GPUs we also show

the theoretical performance, relative with respect to the `GTX480`. Since, theoretical performance is not always reachable we also show the relative practical performance as computed with a simple N-body kernel that is designed for shared-time steps, similar to the N-body example in the CUDA SDK Nyland et al. (2007).

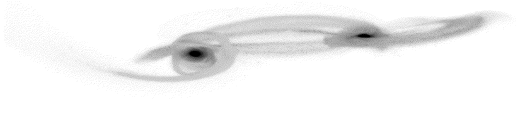
	Cores	Core MHZ	Mem MHZ	Mem bw	TPP	PPP
GTX480	480	1401	3696	384	1	1
GTX680	1536	1006	6008	256	2.3	1.7
K20m	2496	706	5200	320	2.6	1.8
GTX Titan	2688	837	6144	384	3.35	2.2
HD7970	2048	925	5500	384	2.8	2.3

Table 2.1: GPUs used in this work. The first column indicates the GPU, the second column the number of computational cores and the third their clock speed. The fourth and fifth column show the memory clock speed and memory bus width. The sixth and seventh column indicate the relative performance, when using single precision, where we set the performance of the GTX480 to 1. For the sixth column these numbers are determined using the theoretical peak performance (TPP) of the chips. The seventh column indicates the relative practical peak performance (PPP) which is determined using a simple embarrisingly parallel N-body code.

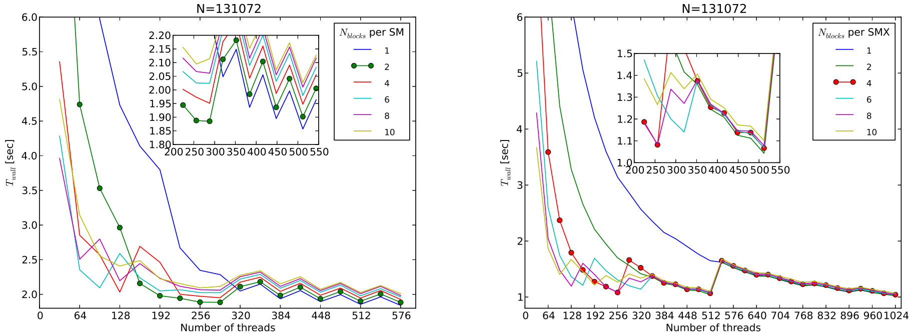
2.3.1 Thread-block configuration

Since `Sapporo2` is designed around the concept of a fixed number of blocks and threads (see Section 2.2) the first thing to determine is the optimal configuration of threads and blocks. We test a range of configurations where we vary the number of blocks per multi-processor and the number of threads per block. The results for four different GPU architectures are presented in Fig. 2.1. In this figure each line represents a certain number of blocks per multi-processor, N_{blocks}. The x-axis indicates the number of threads in a thread-block, $N_{threads}$. The range of this axis depends on the hardware. For the `HD7970` architecture we can not launch more than $N_{threads} = 256$, and for the `GTX480` the limit is $N_{threads} = 576$. For the two `Kepler` devices `680GTX` and `K20m` we can launch up to $N_{threads} = 1024$ giving these last two devices the largest set of configuration options. The y-axis shows the required wall-clock time to compute the forces using the indicated configuration, the bottom line indicates the most optimal configuration.

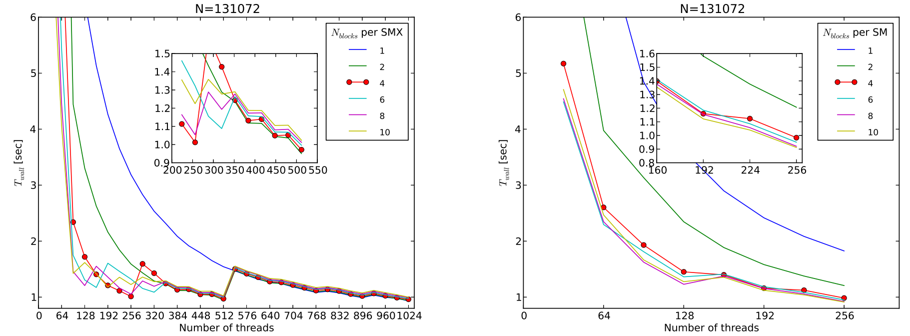
For the `680GTX` and the `K20m` the N_{blocks} configurations reach similar performance when $N_{threads} > 512$. This indicates that at that point there are so many active threads per multi-processor that there are not enough resources (registers and/or shared-memory) to accommodate multiple thread-blocks per multi-processor at the same time. To make the code suitable for block time-steps the configuration with the least number of threads that gives the highest performance would be the most ideal. For the `HD7970` this is $N_{threads} = 256$ while for the `Kepler` architectures $N_{threads} = 512$ gives a slightly lower execution time than $N_{threads} = 256$ and $N_{threads} = 1024$. However, we chose to use $N_{threads} = 256$ for all configurations and use 2D thread-blocks on the `Kepler` devices to launch 512 or 1024 threads. For each architecture the optimal configuration is indicated with the circles



in Fig. 2.1.



(a) `480GTX` (b) `680GTX`



(c) `K20m` (d) `HD7970`

Figure 2.1: The figure shows the required integration time (y-axis) for $N = 131072$ source particles using different number of sink particles (number of threads, x-axis). Each line indicates a different configuration. In each configuration we changed the number of blocks launched per GPU multi-processor for different GPU architectures. Shown in panel A NVIDIAs `Fermi` architecture, in panel B the NVIDIA `Kepler, GK104` architecture in panel C the NVIDIA `Kepler, GK110` and the AMD `Tahiti` architecture in panel D. The AMD architectures are limited to 256 threads. The configurations that we have chosen as our default settings for the number of blocks are the lines with the filled circle markers.

2.3.2 Block-size / active-particles

Now we inspect the performance of `Sapporo2` in combination with a block-time step algorithm. We measured the time to compute the gravitational forces using either the NVIDIA GPU Profiler or the built-in event timings of OpenCL. The number of active sink particles, N_{active}, is varied between 1 and the optimal $N_{threads}$ as specified in the

previous paragraph. The results are averaged over 100 runs and presented in Fig. 2.2. We used 131072 source particles which is enough to saturate the GPU and is currently the average number of particles used in direct N-body simulations.

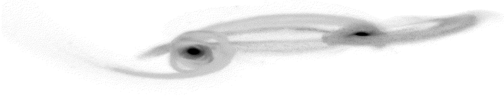
The straight striped lines in Fig. 2.2 indicate the theoretical linear scaling from $(0, 0)$ to $(256, X)$ where X is the execution time of the indicated GPU when $N_{active} = 256$. Visible in the figure are the jumps in the execution time that coincide with the warp (wavefront) size of 32 (64). For NVIDIA devices we can start 2D thread-blocks for all values of N_{active}, since the maximum number of threads that can be active on the device is $\geqslant 512$. The effect of this is visible in the more responsive execution times of the NVIDIA devices when decreasing N_{active} compared to the AMD device. Each time N_{active} drops below a multiple of the maximum number of active threads, the execution time will also decrease. Up to $N_{active} \lesssim 64$ after which the execution time goes down linearly, because of the multiple blocks that can be started for any value of N_{active}. The lines indicated with '1D' in the legend show the execution time if we would not subdivide the work further using 2D thread-blocks. This will under-utilize the GPU and results in increased execution times for $N_{active} < 128$.

The performance difference between `CUDA` and `OpenCL` is minimal which indicates that the compute part of both implementations inhabits similar behavior. For most values of N_{active} the timings of `Sapporo1` and `Sapporo2` are comparable. Only for $N_{active} < 64$ we see a slight advantage for `Sapporo1` where the larger data loads of `Sapporo2` result in a slightly longer execution time. However, the improvements made in `Sapporo2` result in higher performance and a more responsive execution time compared to `Sapporo1` when $N_{active} \geqslant 128$. For the `HD7970`, there is barely any improvement when N_{active} decreases from 256 to 128. There is a slight drop in the execution time at $N_{active} = 192$ which coincides with one less active wavefront compared to $N_{active} = 256$. When $N_{active} \leqslant 128$ we can launch 2D blocks and the performance improves again and approaches that of the NVIDIA hardware, but the larger wavefront size compared to the warp size causes the the execution times to be less responsive to changes of N_{active}.

2.3.3 Range of N

Now that we selected the thread-block configuration we continue with testing the performance when computing the gravitational forces using N_{sink} particles and N_{source} particles, resulting in $N_{sink} \times N_{source}$ force computations. The results are presented in the left panel of Fig. 2.3. This figure shows the results for the five GPUs using `CUDA`, `OpenCL`, `Sapporo1` and `Sapporo2`. The execution time includes the time required to send the input data and retrieve the results from the device.

The difference between `Sapporo1` and `Sapporo2` (both the `CUDA` and `OpenCL` versions) on the K20m GPU are neglibible. `Sapporo1` is slightly faster for $N < 10^4$, because of the increased data-transfer sizes in `Sapporo2`, which influence the performance more when the number of computations is relatively small. `Sapporo2` is slightly faster than `Sapporo1` when $N \geqslant 10^4$, because of the various optimizations added to the new version. The difference between the `GTX680`, `K20m` and the `HD7970` configurations is relatively small. While the `GTX Titan` is almost $1.5\times$ faster and the `GTX480` almost $2\times$ slower than these three cards. These numbers are not unexpected when inspecting their theoretical per-



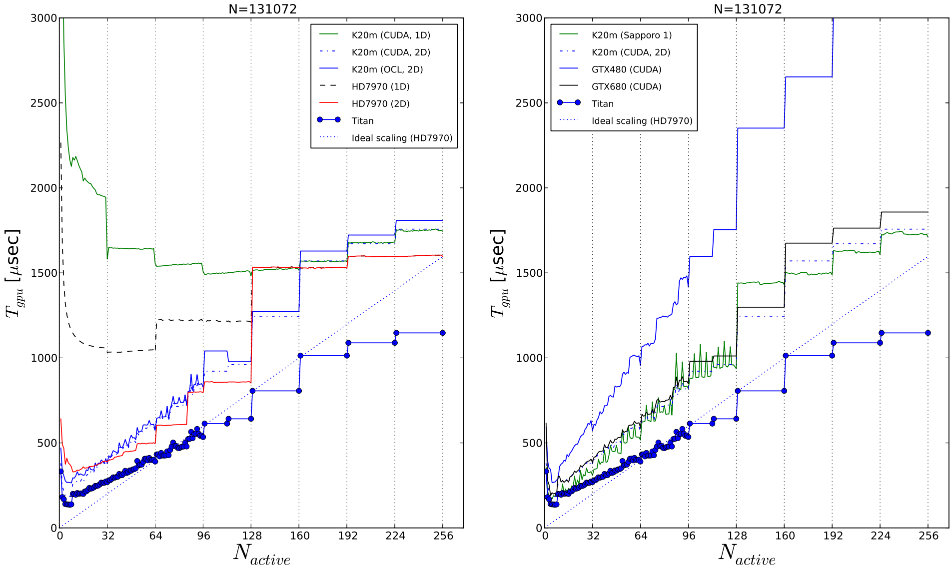


Figure 2.2: Performance for different numbers of active sink particles. The x-axis indicates the number of active particles and the y-axis the required time to compute the gravitational force using 131072 source particles ($N_{active} \times N$ gravity computations). The presented time only includes the time required to compute the gravity, the data transfer times are not included. In both panels the linear striped line shows the ideal scaling from the most optimal configuration with 256 active particles to the worst case situation with 1 active particle for one of the shown devices. The left panel shows the effect on the performance when using 1D thread-blocks instead of 2D on AMD and NVIDIA hardware. Furthermore we show the effect of using `OpenCL` instead of `CUDA` on NVIDIA hardware. When using 1D thread-blocks the GPU becomes underutilized when N_{active} becomes smaller than ~ 128. This is visible as the execution time increases while N_{active} becomes smaller. The right panel compares the performance of the five different GPUs as indicated. Furthermore it shows that the performance of `Sapporo2` is comparable to that of `Sapporo1`.

formance (see Tab. 2.1). For $N < 10^5$ we further see that the performance of the `HD7970` is lower than for the NVIDIA cards. This difference is caused by slower data transfer rates between the host and device for the `HD7970`. Something similar can be seen when we compare the `OpenCL` version of the `K20m` with the `CUDA` version. Close inspection of the timings indicate that this difference is caused by longer CPU-GPU transfer times in the `OpenCL` version when transfering small amounts of data (< 100 KB), which for small N forms a larger part of the total execution time.

2.3.4 Double precision vs Double-single precision

As mentioned in Section 2.2.2 the higher order integrators require the use of double precision computations. Therefore we test the performance impact when using full native double precision instead of double-single precision. For this test we use the `GTX680`, `K20m` and the `HD7970`. The theoretical peak performance when using double precision computations is lower than the peak performance when using single precision computations. The double

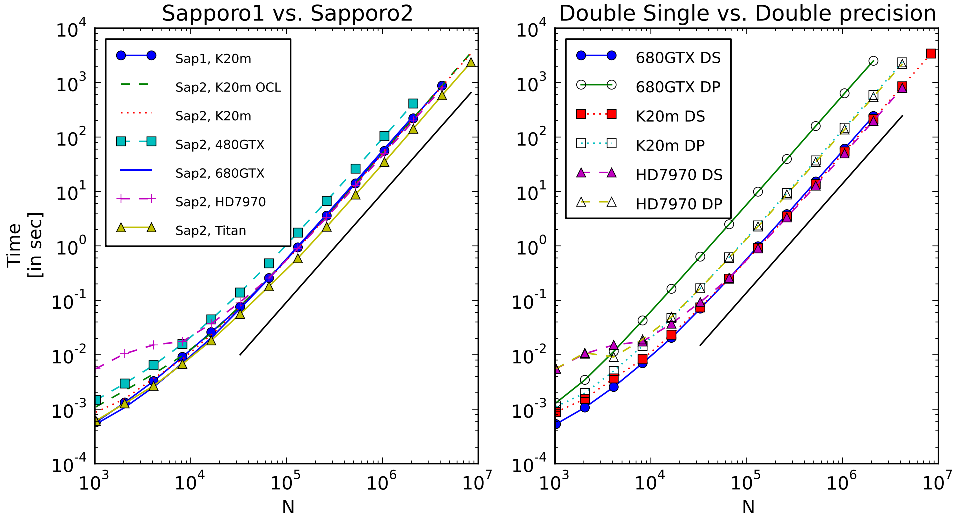
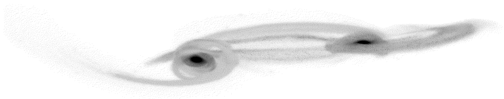


Figure 2.3: Time required to solve N^2 force computations using different configurations. In both panels the number of source particles is equal to the number of sink particles which is indicated on the x-axis. The y-axis indicates the required wall-clock time to execute the gravity computation and to perform the data transfers. Unless otherwise indicated we use CUDA for the NVIDIA devices. The left panel shows the performance of `Sapporo1` on a `K20m` GPU and `Sapporo2` on 5 different GPUs using a mixture of `CUDA` and `OpenCL`. The straight solid line indicates N^2 scaling. The right panel shows the difference in performance between double-single and double precision. We show the performance for three different devices. The double-single timings are indicated by the filled symbols. The double-precision performance numbers are indicated by the lines with the open symbols. The straight solid line indicates N^2 scaling.

precision performance of the `K20m` is one third that of the single precision performance. For the `GTX680` this is $\frac{1}{24}$th and for the `HD7970` this is one fourth. As in the previous section we use the wall-clock time required to perform N^2 force computations to compare the devices. The results are presented in the right panel of Fig. 2.3, here the double precision timings are indicated with the open symbols and the double-single timings with the closed symbols.

As in the previous paragraph, when using double-single precision the performance is comparable for all three devices. However, when using double-precision the differences become more clear. As expected, based on the theoretical numbers, the `GTX680` is slower than the other two devices. The performance of the `K20m` and the `HD7970` are comparable for $N > 10^4$. For smaller N the performance is more influenced by the transfer rates between the host and the device than by its actual compute speed.

Taking a closer look at the differences we see that the performance of the `GTX680` in full double precision is about $\sim 10\times$ lower than when using double-single precision. For the other two cards the double precision performance is roughly $\sim 1.5\times$ lower. For all the devices this is roughly a factor of 2 difference from what can be expected based on the specifications. This difference can be explained by the knowledge that the number of oper-



ations is not exactly the same for the two versions[4] and even in the double single method we use the special operation units to compute the `sqrt`. Another reason for the discrepancy between the practical and theoretical numbers is that we keep track of the nearest neighbours which requires the same operations for the double single and the double precision implementation. Combining this with the knowledge that we already execute a number of double precision operations to perform atomic additions and data reads, results in the observed difference between the theoretical and empirically found performance numbers.

2.3.5 Sixth order performance

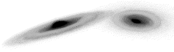
The reason to use sixth order integrators compared to lower order integrators is that, on average, they are able to take larger time-steps. They are also better in handling systems that contain large mass ratios (for example when the system contains a supermassive blackhole). The larger time-step results in more active particles per block-step which improves the GPU efficiency. However, to accurately compute the higher order derivatives double precision accuracy has to be used. This negatively impacts the performance and in Fig. 2.4 we show just how big this impact is. As in the previous figures we present the time to compute N^2 forces. Presented are the performance of the sixth order kernel using double precision, the fourth order kernel using double-single precision and the fourth order kernel using double precision. As expected, the sixth order requires the most time to complete as it executes the most operations. The difference between the fourth order in double-single and the sixth order in double precision is about a factor 4. However, if we compare the performance of the double precision fourth order kernel with the sixth order kernel the difference is only about a factor of 1.4. This small difference in performance shows that it is beneficial to consider using a sixth order integrator when using high mass ratios or if, for example, high accuracy is required to trace tight orbits.

2.3.6 Multi-GPU

As described in Section 2.2, `Sapporo2` supports multiple GPUs in parallel. The parallelised parts are the force computation, data transfer and prediction of the source particles. The transfer of particle properties to the device and the transfer of the force computation results from the device are serial operations. These operations have a small but constant overhead, independent of the number of GPUs. For the measurements in this section we use the total wallclock time required to compute the forces on N particles (as in Section 2.3.3). The speed-up compared to 1 GPU is presented in Fig. 2.5. The timings are from the `K20m` GPUs which have enough memory to store up to 8×10^6 particles. For $N > 10^4$ it is efficient to use all available GPUs in the system and for $N \leqslant 10^4$ all multi-GPU configurations show similar performance. The only exception here is when $N = 10^3$ at which point the overhead of using 4 GPUs is larger than the gain in compute power. For large enough N the scaling is near perfect ($T_{single-GPU}/T_{multi-GPU}$), since the execution time is dominated by the computation of the gravitational interactions.

---

[4] Double single requires more computations than single precision on which the theoretical numbers are based



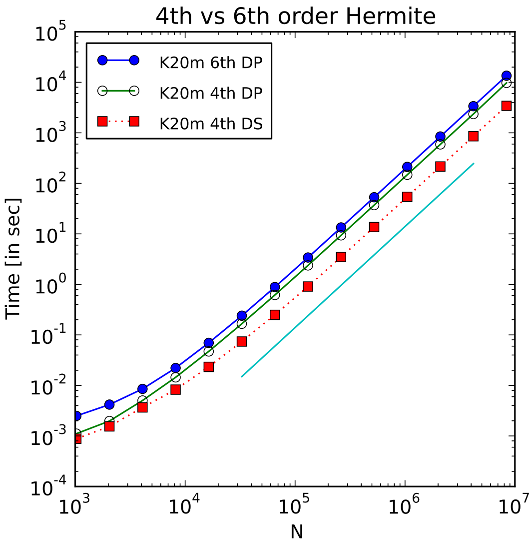


Figure 2.4: Performance difference between fourth and sixth order kernels. Shown is the time required to solve N^2 force computations using different configurations. The number of source particles is equal to the number of sink particles indicated on the x-axis. The y-axis indicates the required wall-clock time to execute the gravity computation and to perform the data transfers. The fourth-order configuration using double-single precision is indicated by the dotted line with square symbols. the fourth-order configuration using double precision is indicated by the solid line with open circles and the solid line with closed circles indicates a sixth-order configuration using double precision. The straight solid line without symbols indicates the N^2 scaling. Timings performed on a `K20m` GPU using `CUDA 5.5`.

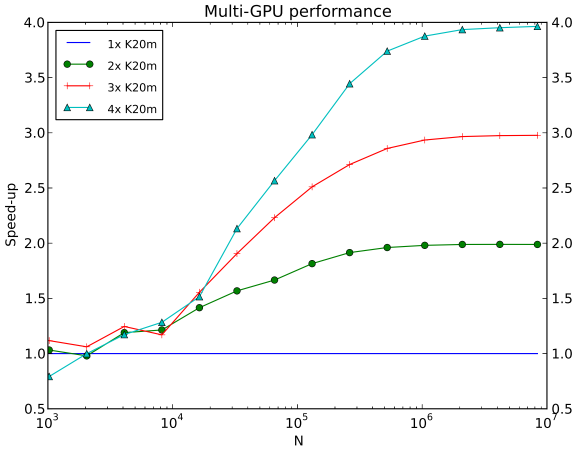
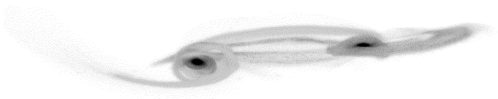


Figure 2.5: Multi-GPU speed-up over using one GPU. For each configuration the total wall-clock time is used to compute the speed-up (y-axis) for a given N (x-axis). The wall-clock time includes the time required for the reduction steps and data transfers. Timings performed on `K20m` GPUs using `Sapporo2` and `CUDA 5.5`.

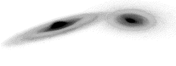


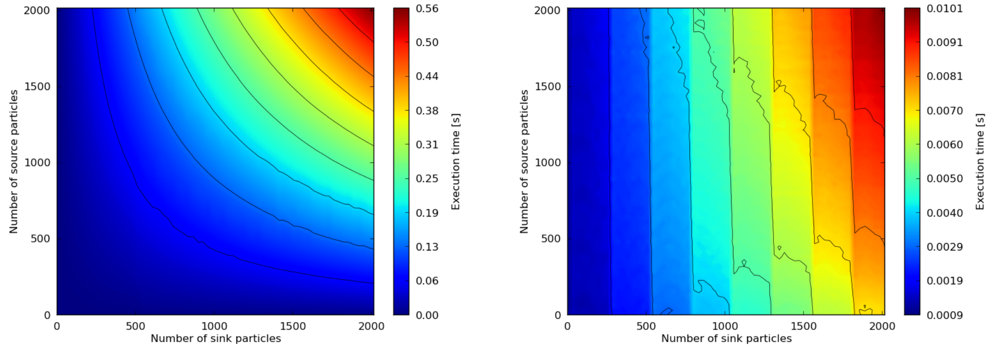
2.4 Discussion and CPU support

2.4.1 CPU

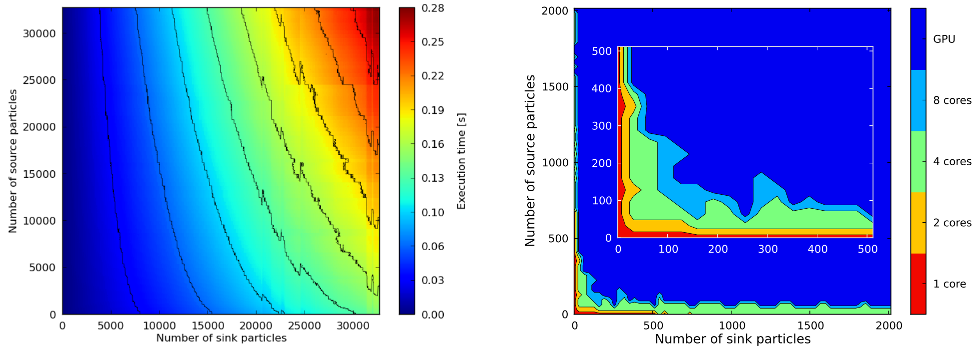
With the availability of CPUs with 8 and more cores that support advanced vector instructions there is the recurring question if it is not faster to compute the gravity on the CPU than on the GPU. Especially since there is no need to transfer data between the host and the device which can be relatively costly when the number of particles is $\leqslant 1024$. To test exactly for which number of particles the CPU is faster than the GPU we added a CPU implementation to `Sapporo2`. This CPU version has support for SSE2 vector instructions and OpenMP parallelisation. The only kernel implemented is the fourth order integrator, including support for neighbour lists and nearest neighbours (particle-ID and distance). Because the performance of the GPU depends on the combination of sink and source particles we test a grid of combinations for the number of sink and source particles when measuring the time to compute the gravitational forces. The results for the CPU (a Xeon E5620 @ 2.4Ghz), using a single core, are presented in Fig. 2.6a. In this figure (and all the following figures) the x-axis indicates the number of sinks and the y-axis the number of sources. The execution time is indicated by the colour from blue (fastest) to red (slowest). The smooth transition from blue to red from the bottom left corner to the top right indicates that the performance does not preferentially depend on either the source or sink particles but rather on the combined number of interactions. This matches our expectations, because the parallelisation granularity on the CPU is as small as the vector width, which is 4. On the GPU this granularity is much higher, as presented in Fig. 2.6b, here we see bands of different colour every 256 particles. Which corresponds to the number of threads used in a thread-block ($N_{threads}$). With 256 sink particles we have the most optimal performance of a block, if however we would have 257 sink particles we process the first 256 sinks using optimal settings while the 257th sink particle is processed relative inefficient. This granularity becomes less obvious when we increase the number of interactions as presented in Fig. 2.6c. Here we see the same effect appearing as with the CPU (Fig. 2.6a), where the granularity becomes less visible once we saturate the device and use completely filled thread-blocks for most of the particles. The final panel, Fig. 2.6d, indicates per combination of source and sink particles which CPU or GPU configuration is the fastest. For the CPU we measured the execution time when using 1,2,4 or 8 cores. In this panel the colours indicate the method which gives the shortest execution times.

When either the number of sinks or the number of sources is relative small ($\leqslant 100$) the CPU implementation performs best. However, when the number of sinks or sources is > 100 the GPU outperforms the CPU. When using a CPU implementation that uses the AVX or AVX2 instruction sets the borders of these regions would shift slightly upwards. The CPU would then be faster for a larger number of source/sink particles, but that would only be at most a factor of 2 to 4 more particles. The data of Fig. 2.6 confirms that our choice to implement the `Sapporo2` library for the GPU is an efficient method for realistic data-set sizes.





(a) `480GTX` (b) `680GTX`

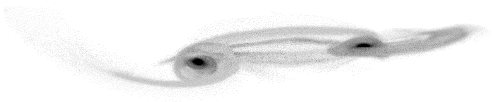


(c) `K20m` (d) `HD7970`

Figure 2.6: GPU and CPU execution times. In all the subplots the x-axis indicates the number of sink particles and the y-axis the number of source particles used. For subplots a,b and c the raw execution times are presented and indicated with the colours. Plot d does not present the execution time but rather which of the configuration gives the best performance. The inset of plot d is a zoom-in of the main plot. Note that the colours are scaled per plot and are not comparable between the different subplots. All the GPU times include the time required to copy data between the host and device.

2.4.2 XeonPhi

Because the `Sapporo2` library can be built with `OpenCL` it should, theoretically, be possible to run on any device that supports `OpenCL`. To put this to the test, we compiled the library with the Intel `OpenCL` implementation. However, although the code compiled without problems it did not produce correct answers. We tested the library both on an Intel CPU and the Intel `XeonPhi` accelerator. Neither the CPU, nor the `XeonPhi` produced correct results. Furthermore, the performance of the `XeonPhi` was about 100× lower than what



can be expected from its theoretical peak performance. We made some changes to the configuration parameters such as $N_{threads}$ and N_{blocks}, however this did not result in any presentable performance. We suspect that the Intel `OpenCL` implementation, especially for XeonPhi, contains a number of limitations that causes it to generate bad performing and/or incorrect code. Therefore the `Sapporo2` library is not portable to Intel architectures with their current `OpenCL` implementation. This does not imply that the `XeonPhi` has bad performance in general, since it is possible to achieve good performance on N-body codes that is comparable to GPUs. However, this requires code that is specifically tuned to the `XeonPhi` architecture (K. Nitadori, private communication [5]).

2.5 Conclusion

The here presented `Sapporo2` library makes it easy to enable GPU acceleration for direct N-body codes. We have seen that the difference between the `CUDA` and `OpenCL` implementations is minimal when there are enough particles to make the simulation compute limited. However, if many small data transfers are required, for example when the integrator takes very small time-steps with few active particles, the `CUDA` implementation will be faster. Apart from the here presented fourth and sixth order integrators the library also contains a second order implementation. And because of the storage of data in double precision it can be trivially expanded with an eight order integrator. The performance gain when using multiple GPUs implies that it is efficient to configure GPU machines that contain more than 1 GPU. This will improve the time to solution for simulations with more than 10^4 particles.

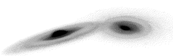
The `OpenCL` support and built-in tuning methods would allow easy extension to other `OpenCL` suported devices. However, this would require a mature `OpenCL` library that supports atomic operations and double precision data types. For the `CUDA` devices this is not that a problem since the current `CUDA` libraries already have mature support for the used operations and the library automatically scales to future architectures. The only property that has to be set is the number of thread-blocks per multiprocessor and this can be easily identified using the figures as presented in Section 2.3.1.

Acknowledgements

This work was supported by the Netherlands Research Council NWO (grants #643.200.503, #639.073.803, #614.061.608, # 612.071.503, #643.000.802).

---

[5] Also see `https://github.com/nitadori/Hermite` and `http://research.colfaxinternational.com/post/2013/01/07/Nbody-Xeon-Phi.aspx`.



3

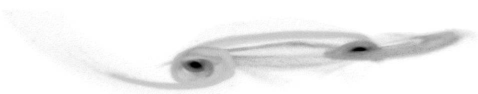
Gravitational tree-code on graphics processing units: implementation in CUDA

We present a new very fast tree-code which runs on massively parallel Graphical Processing Units (GPU) with NVIDIA CUDA architecture. The tree-construction and calculation of multipole moments is carried out on the host CPU, while the force calculation which consists of tree walks and evaluation of interaction list is carried out on the GPU. In this way we achieve a sustained performance of about 100GFLOP/s and data transfer rates of about 50GB/s. It takes about a second to compute forces on a million particles with an opening angle of $\theta \approx 0.5$. The code has a convenient user interface and is freely available for use[1].

Evghenii Gaburov, Jeroen Bédorf and Simon Portegies Zwart
Procedia Computer Science, Volume 1, Issue 1, p.1119-1127, May 2010.

---

[1] `http://castle.strw.leidenuniv.nl/software/octgrav.html`



3.1 Introduction

Direct force evaluation methods have always been popular because of their simplicity and unprecedented accuracy. Since the mid 1980's, however, approximation methods like the hierarchical tree-code (Barnes and Hut 1986) have gained enormous popularity among researchers, in particular for studying astronomical self-gravitating N-body systems (Aarseth 2003) and for studying softmatter molecular-dynamics problems (Frenkel and Smit 2001). For these applications, direct force evaluation algorithms strongly limit the applicability, mainly due to the $O(N^2)$ time complexity of the problem.

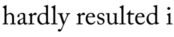
Tree-codes, however, have always had a dramatic set back compared to direct methods, in the sense that the latter benefits from the developments in special purpose hardware, like the GRAPE and MD-GRAPE family of computers Makino and Taiji (1998); Makino (2001), which increase workstation performance by two to three orders of magnitude. On the other hand, tree-codes show a better scaling of the compute time with the number of processors on large parallel supercomputers Warren and Salmon (1993); Warren et al (1997) compared to direct N-body methods Harfst et al. (2007); Gualandris et al. (2007). As a results, large scale tree-code simulations are generally performed on Beowulf-type clusters or supercomputers, whereas direct N-body simulations are performed on workstations with attached GRAPE hardware.

Tree-codes, due to their hierarchical and recursive nature are hard to run efficiently on dedicated Single Instruction Multiple Data (SIMD) hardware like GRAPE, though some benefit has been demonstrated by using pseudo-particle methods to solve for the higher-order moments in the calculation of multipole moments of the particle distributions in grid cells Kawai et al. (2004).

Recently, the popularity of computer games has led to the development of massively parallel vector processors for rendering three-dimensional graphical images. Graphical Processing Units (or GPUs) have evolved from fixed function hardware to general purpose parallel processors. The theoretical peak speed of these processors increases at a rate faster than Moores' law (Moore 1965), and at the moment top roughly 200 GFLOP for a single card. The cost of these cards is dramatically reduced by the enormous volumes in which they are produced, mainly for gamers, whereas GRAPE hardware remains relatively expensive.

The gravitational N-body problem proved to be rather ideal to port to modern GPUs, and the first success in porting the N-body problem to programmable GPUs was achieved by Nyland et al. (2004), but it was only after the introduction of the NVIDIA G80 architecture that accurate force evaluation algorithms could be implemented Portegies Zwart et al. (2007) and that the performance became comparable to special purpose computers Belleman et al. (2008); Gaburov et al. (2009).

Even in these implementations, the tree-code, though pioneered in Belleman et al. (2008), still hardly resulted in a speed-up compared to general purpose processors. In this paper we present a novel implementation of a tree-code on the NVIDIA GPU hardware using the CUDA programming environment.



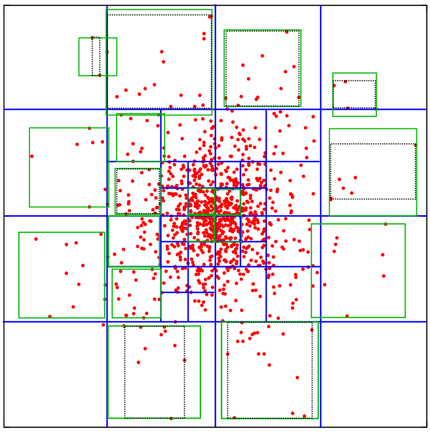


Figure 3.1: Illustration of our tree-structure, shown in 2D for clarity. Initially, the space is recursively subdivided into cubic cells until all cells contain less than N_{leaf} particles (blue squares). All cells (including parent cells) are stored in a tree-structure. Afterwards, we compute a tight bounding box for the particles in each cell (dotted rectangles) and cell's boundary. The latter is a cube with a side length equal to the largest side length of the bounding box and the same centre (green squares).

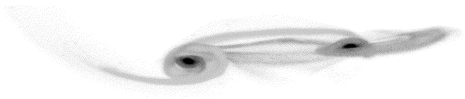
3.2 Implementation

In the classical implementation of the tree-code algorithm all the work is done on the CPU, since special purpose hardware was not available at that time Barnes and Hut (1986). With the introduction of GRAPE special purpose hardware Ito et al. (1990); Fukushige et al. (1991), it became computationally favourable to let the special purpose hardware, instead of the CPU, calculate accelerations. Construction of the interaction list in these implementations takes nearly as much time as calculating the accelerations. Since the latest generation of GPUs allows more complex operations, it becomes possible to build the interaction list directly on the GPU. In this case, it is only necessary to transport the tree-structure to the GPU. Since the bandwidth on the host computer is about an order of magnitude lower than on the GPU, it is also desirable to offload bandwidth intensive operations to the GPU. The construction of the interaction list is such an operation. The novel element in our tree-code is construction of the interaction list on the GPU. The remaining parts of the tree-code algorithm (tree-construction, calculation of node properties and time integration) are executed on the host. The host is also responsible for the allocation of memory and the data transfer to and from the GPU. In the next sections we will cover the details of the host and device steps.

3.2.1 Building the octree

We construct the octree in the same way as done in the original BH tree-code. We define the computational domain as a cube containing all particles in the system. This cube is recursively divided into eight equal-size cubes called cells. The length of the resulting cells is half the length of the parent cell. Each of these cells is further subdivided, until less than N_{leaf} particles are left. We call these cells leaves, whereas cells containing more than N_{leaf} particles are referred to as nodes. The cell containing all particles is the root node.

The resulting tree-structure is schematically shown in fig3.1. From this tree-structure we construct groups for the tree walk (c.f. section 3.2.2), which are the cells with the num-



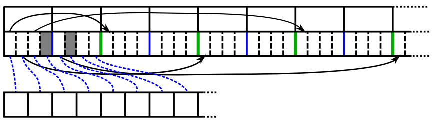


Figure 3.2: Illustration of the tree structure as stored in device memory.

ber of particles less than N_{groups}, and compute properties for each cell, such as boundary, mass, centre of mass, and quadrupole moments, which are required to calculate accelerations McMillan and Aarseth (1993).

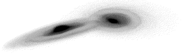
In order to efficiently walk the octree on the device, its structure requires some reorganisation. In particular, we would like to minimise the number of memory accesses since they are relative expensive (up to 600 clock cycles). In fig3.2, we show the tree-structure as stored on the GPU. The upper array in the figure is the link-list of the tree, which we call the main tree-array. Each element in this array (separated by blue vertical lines) stores four integers in a single 128-bit words (dashed vertical lines). This structure is particularly favourable because the device is able to read a 128-bit word into four 32-bit registers using one memory access instruction. Two array-elements represent one cell in the tree (green line) with indices to each of the eight children in the main tree-array (indicated by the arrows). A grey filled element in this list means that a child is a leaf (it has no children of its own), and hence it needs not to be referenced. We also use auxiliary tree-arrays in the device memory which store properties of each cell, such as its boundary, mass, centre of mass and multiple moments. The index of each cell in the main tree-array is directly related to its index in the auxiliary tree-arrays by bitshift and addition operations.

The device execution model is designed in such a way that threads which execute the same operation are grouped in warps, where each warp consists of 32 threads. Therefore, all threads in a warp follow the same code path. If this condition is not fulfilled, the divergent code path is serialised, therefore negatively impacting the performance (NVIDIA Corp. 2007). To minimise this, we rearrange groups in memory to make sure that neighbouring groups in space are also neighbouring in memory. Combined with similar tree paths that neighbouring groups have, this will minimise data and code path divergence for neighbouring threads, and therefore further improves the performance.

3.2.2 Construction of an interaction list

In the standard BH-tree algorithm, the interaction lists are constructed for each particle, but particles in the same groups have similar interaction lists. We make use of this fact by building the lists for groups instead of particles (Barnes 1990). The particles in each group, therefore, share the same interaction list, which is typically longer than it would have been by determining it on a particle basis. The advantage here is the reduction of the number of tree walks by N_{group}. The tree walk is performed on the GPU in such a way that a single GPU thread is used per group. To take advantage of the cached texture memory, we make sure that neighbouring threads correspond to neighbouring groups.

Owing to the massively parallel architecture of the GPU, two tree walks are required to construct interaction lists. In the first walk, each thread computes the size of the in-



teraction list for a group. This data is copied to the host, where we compute the total size of the interaction list, and memory addresses to which threads should write lists without intruding on other threads' data. In the second tree walk, the threads write the interaction lists to the device memory.

List 3.1: A pseudo code for our non-recursive stack-based tree walk.

```
1  while (stack.non_empty)
2    node = stack.pop                       ;; get next node from the stack
3    one  = fetch(children, node + 0)       ;; cached fetch 1st four children
4    two  = fetch(children, node + 1)       ;; cached fetch 2nd four children
5    test_cell<0...4>(node, one, stack)     ;; test sub-cell in octant one to four
6    test_cell<5...8>(node, two, stack)     ;; test sub-cell in octant four to eight
```

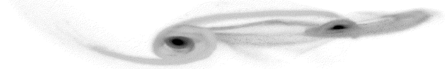
List 3.2: Pseudo code for `test_cell` subroutine.

```
1  template<oct>test_cell(node, child, stack)
2     child = fetch(cell_pos, 8*node +oct)    ;; fetch data of the child
3     if (open_node(leaf_data, child))        ;; if the child has to be opened,
4       if (child != leaf) stack.push(child)  ;; store it in stack if it is a node
5       else              leaf += 1           ;; otherwise increment the leaf counter
6     else                cell += 1           ;; else, increment the cell counter
```

We implemented the tree walk via a non-recursive stack-based algorithm (the pseudo code is shown in List 3.1), because the current GPU architecture does not support recursive function calls. In short, every node of the tree, starting with the root node, reads indices of its children by fetching two consecutive 128-bit words (eight 32 bit integers) from texture memory. Each of these eight children is tested against the node-opening criteria θ (the pseudo code for this step is shown in List 3.2), and in the case of a positive result a child is stored in the stack (line 4 in the listing), otherwise it is considered to be a part of the interaction list. In the latter case, we check whether the child is a leaf, and if so, we increment a counter for the leaf-interaction list (line 5), otherwise a counter for the node-interaction list (line 6). This division of the interaction lists is motivated by the different methods used to compute the accelerations from nodes and leaves (c.f. section 3.2.3). In the second tree walk, we store the index of the cell in the appropriate interaction list instead of counting the nodes and leafs.

3.2.3 Calculating accelerations from the interaction list

In the previous step, we have obtained two interaction lists: one for nodes and one for leaves. The former is used to compute accelerations due to nodes, and the latter due to leaves. The pseudo-code for a particle-node interaction is shown in List 3.3 and the memory access pattern is demonstrated in the left panel of fig3.3. This algorithm is similar to the one used in the `kirin` and `sapporo` libraries for direct N-body simulations Belleman et al. (2008); Gaburov et al. (2009). In short, we use a block of threads per group, such that a thread in a block is assigned to a particle in a group; these particles share the same interaction list. Each thread loads a fraction of the nodes from the node-interaction list into shared memory (blue threads in the figure, lines 2 and 3 in the listing). To ensure that all the data is loaded into shared memory, we put a barrier for all threads (line 4), and afterwards each thread computes gravitational acceleration from the nodes in shared memory (line 5). Prior loading a new set of nodes into the shared memory (green threads



List 3.4: Body-leaf interaction

```
1 for (i = 0; i < list_len; i += block_size) {
2   leaf = leaf_interaction_list[i + threads_id]
3   shared_leaves[threadIdx] = cells_list[leaf] ;; read leaves to the shared memory
4   __syncthreads()
5   for (j = 0; j < block_size; j++)            ;; process each leaf
6     shared_bodies[thread_id] =  bodies[shared_leaves[j].first + thread_id]
7     __syncthreads();
8     interact(body_in_a_thread, shared_bodies, shared_leaves[j].len);
9     __syncthreads();
```

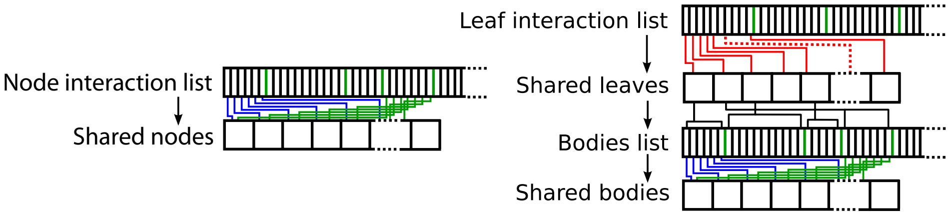


Figure 3.3: Memory access pattern in a body-node (left) and body-leaf (right) interaction.

in the figure), we ensure that all the threads have completed their calculations (line 6). We repeat this cycle until the interaction list is exhausted.

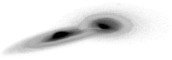
List 3.3: Body-node interaction

```
1 for (i = 0; i < list_len; i += block_size)
2   cellIdx = cell_interact_lst[i + thread_id]
3   shared_cells[threadIdx] = cells_lst[cellIdx] ;; read nodes to the shared memory
4   __syncthreads()                              ;; thread barrier
5   interact(body_in_a_thread, shared_cell)      ;; evaluate accelerations
6   __syncthreads()                              ;; thread barrier
```

Calculations of gravitational acceleration due to the leaves differs in several ways. The pseudo-code of this algorithm is presented in List 3.4, and the memory access pattern is displayed in the right panel of Fig. 3.3. First, each thread fetches leaf properties, such as index of the first body and the number of bodies in the leaf, from texture memory into shared memory (red lines in the figure, lines 2 and 3 in the listing). This data is used to identify bodies from which the accelerations have to be computed (black lines). Finally, threads read these bodies into shared memory (blue and green lines, line 6) in order to calculate accelerations (line 8). This process is repeated until the leaf-interaction list is exhausted.

3.3 Results

In this section we study the accuracy and performance of the tree code. First we quantify the errors in acceleration produced by the code and then we test its performance. For this purpose we use a model of the Milky Way galaxy (Widrow and Dubinski 2005). We model the galaxy with $N = 10^4, 3 \cdot 10^4, 10^5, 3 \cdot 10^5, 10^6, 3 \cdot 10^6$ and 10^7 particles, such that the mass ratio of bulge, disk and halo particles is 1:1:4. We then proceed with the



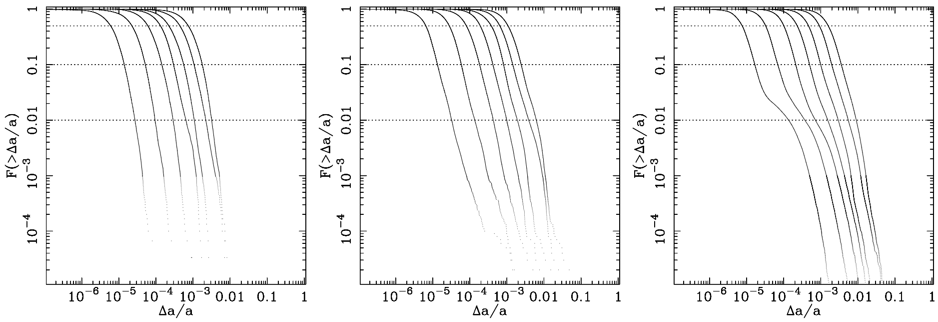


Figure 3.4: Each panel displays a fraction of particles having relative acceleration error (vertical axis) greater than a given value (horizontal axis). In each panel, we show errors for various opening angles from $\theta = 0.2$ (the leftmost curve in each panel), 0.3, 0.4, 0.5, 0.6 and 0.7 (the rightmost curve). The number of particles are $3 \cdot 10^4$, 10^5, 10^6 for panels from left to right respectively. The dotted horizontal lines show 50%, 10% and 1% of the error distribution.

measurements of the code performance. In all test we use $N_{\rm leaf} = 64$ and $N_{\rm group} = 64$ which we find produce the best performance on both G80/G92 and GT200 architecture. The GPU used in all the tests is a GeForce 8800Ultra.

3.3.1 Accuracy of approximation

We quantify the error in acceleration in the following way:
$\Delta a/a = |\mathbf{a}_{\rm tree} - \mathbf{a}_{\rm direct}|/|\mathbf{a}_{\rm direct}|$, where $\mathbf{a}_{\rm tree}$ and $\mathbf{a}_{\rm direct}$ are accelerations calculated by the tree and direct summation respectively. The latter was carried out on the same GPU as the tree-code. This allowed us to asses errors on systems as large as 10 million particles[2]. In fig3.4 we show error distributions for different numbers of particles and for different opening angles. In each panel, we show which fraction of particles (vertical-axis) has a relative error in acceleration larger than a given value (horizontal axis). The horizontal lines show 50th, 10th and 1st percentile of cumulative distribution. This data shows that acceleration errors in this tree-code are consistent with the errors produced by existing tree-codes with quadrupole corrections (Dehnen 2002; Springel et al. 2001; Stadel 2001).

We show dependence of errors on both opening angle and number of particles in fig3.5. In the leftmost panel of the figure, we plot the median and the first percentile of the relative force error distribution as a function of the opening angle θ for various number of particles $N = 3 \cdot 10^4$ (the lowest blue dotted and red dashed lines), $3 \cdot 10^5$ and $3 \cdot 10^6$ (the upper blue dotted and red dashed lines). As expected, the error increases as a function of θ with the following scaling from the least-squared fit, $\Delta a/a \propto \theta^4$. However, the errors increase with the number of particles: the error doubles when the number of particles is increased by two orders of magnitude. This increase of the error is caused by the large number of particles in a leaf, which in our case is 64, to obtain the best performance. We conducted a test with $N_{\rm leaf} = 8$, and indeed observed the expected decrease of the error when the

---

[2] We used the NVIDIA 8800Ultra GPU for this paper, and it takes ∼10 GPU hours to compute the exact force on a system with 10 million particles with double precision emulation (Gaburov et al. 2009)

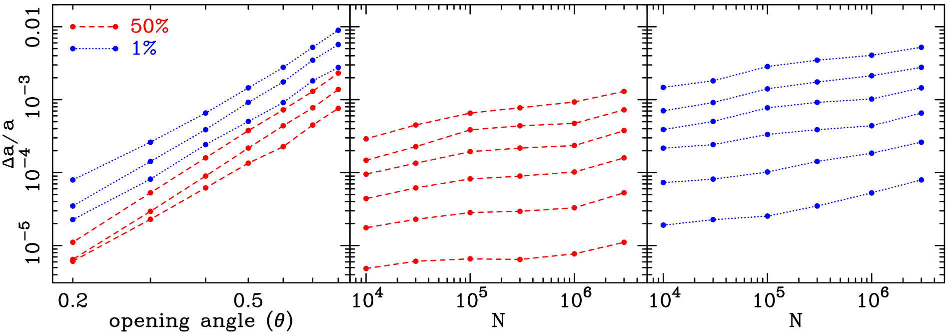


Figure 3.5: The median and the first percentile of the relative acceleration error distribution as a function of the opening angle and the number of particles. In the leftmost panel we show lines for $3 \cdot 10^4$ (the bottom dotted and dashed lines) and $3 \cdot 10^6$ (the top dotted and dashed lines) particles. The middle and the right panels display the error for $\theta = 0.2$ (the bottom lines), $0.3, 0.4, 0.5, 0.6$ and 0.7 (the upper lines).

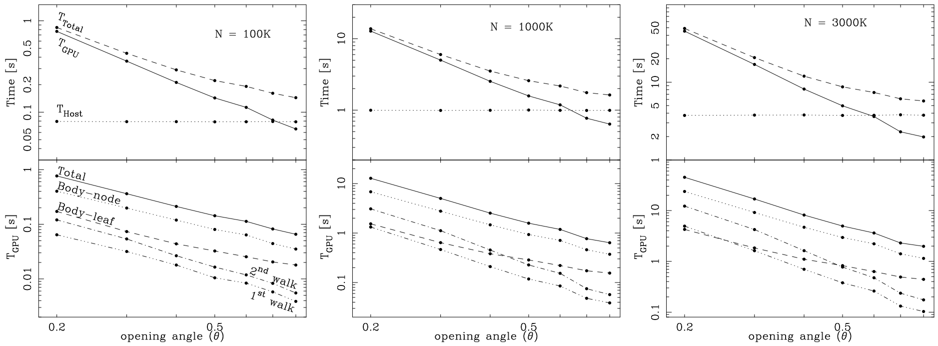
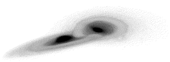


Figure 3.6: Wall-clock timing results as function of the opening angle and number of particles. In each panel, the solid line shows the time spent on the GPU. The dotted line on the top panel shows the time spent on the host, and the total wall-clock time is shown with the dashed line.

number of particles increases; this error, however, is twice as large compared to $N_{\rm leaf} = 64$ for $N \sim 10^6$.

3.3.2 Timing

In fig3.6 we present the timing data as a function of θ and for various N. The $T_{\rm Host}$ (dotted line in the figure) is independent of θ, which demonstrates that construction of the octree only depends on the number of particles in the system, with $T_{\rm Host} \propto N \log N$. This time becomes comparable to the time spend on the GPU calculating accelerations for $N \gtrsim 10^6$ and $\theta \gtrsim 0.5$. This is caused by the empirically measured near-linear scaling of time spend on GPU with N. As the number of particles increases, the GPU part of the code performs more efficiently, and therefore the scaling drops from $N \log N$ to near-linear (fig3.7). We therefore conclude, that the optimal opening angle for our code is $\theta \approx 0.5$.



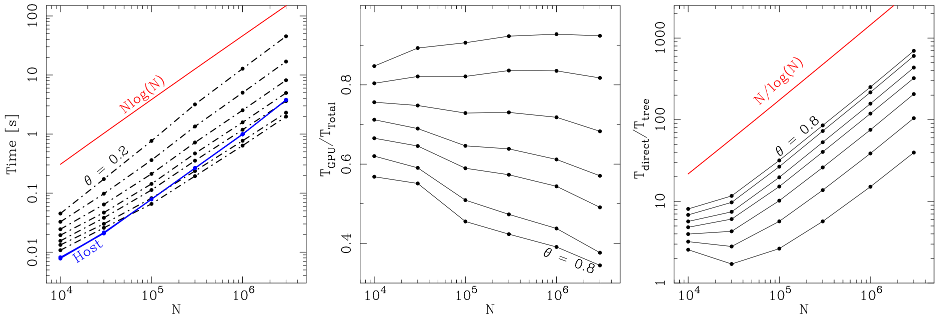


Figure 3.7: Timing results as a function of particle number. The leftmost panel displays time spent on the GPU (black dash-dotted lines) and host CPU (blue solid line) parts as a function of the number of particles. The expected scaling $N\log(N)$ is shown in the red solid line. The ratio of the time spent on GPU to the total wall-clock time is given in the middle panel. The speed-up compared to direct summation is shown in the rightmost panel. The expected scaling $N/\log(N)$ is shown with a solid red line.

In the leftmost panel of fig3.7 we show N dependence of the time spent on the host and the device for various opening angles. In particular, $T_{\rm GPU}$ scaling falls between $N\log(N)$ and N, which we explained by the increased efficiency of the GPU part of our code with larger number of particles. This plot also shows that the host calculation time is a small fraction of the GPU calculation time, except for $N \gtrsim 10^6$ and $\theta \gtrsim 0.5$. The middle panel of the figure shows the ratio of the time spent on the device to the total time. Finally, the rightmost panel shows the ratio between the time required to compute forces by direct summation and the time required by the tree-code. As we expected, the scaling is consistent with $N^2/(N\log(N)) = N/log(N)$.

3.3.3 Device utilisation

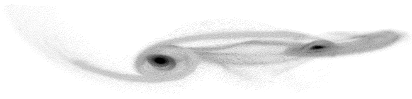
We quantify the efficiency of our code to utilise the GPU resources by measuring both instruction and data throughput, and then compare the results to the theoretical peak values provided by the device manufacturer. In fig3.8 we show both bandwidth and computational performance as function of θ for three different N. We see that the calculation of accelerations operates at about 100GFLOPs[3]. This is comparable to the peak performance of the GRAPE-6a special-purpose hardware, but this utilises only $\sim$ 30% of the computational power of the GPU[4]. This occurs because the average number of bodies in a group is a factor of 3 or 4 smaller than the $N_{\rm group}$, which we set to 64 in our tests. On average, therefore, only about 30% of the threads in a block are active.

The novelty of this code is the GPU-based tree walk. Since there is little arithmetic intensity in these operations, the code is, therefore, limited by the bandwidth of the device. We show in fig3.8 that our code achieves respectable bandwidth: in excess of 50GB/s

---

[3] We count 38 and 70 FLOPs for each body-leaf and body-node interaction respectively.

[4] Our tests were carried out on a NVIDIA 8800Ultra GPU, which has 128 streaming processors each operating at clock speed of 1.5Ghz. Given that the GPU is able to perform up to two floating point operation per clock cycle, the theoretical peak performance is $2 \times 128 = 384$GFLOP/s.



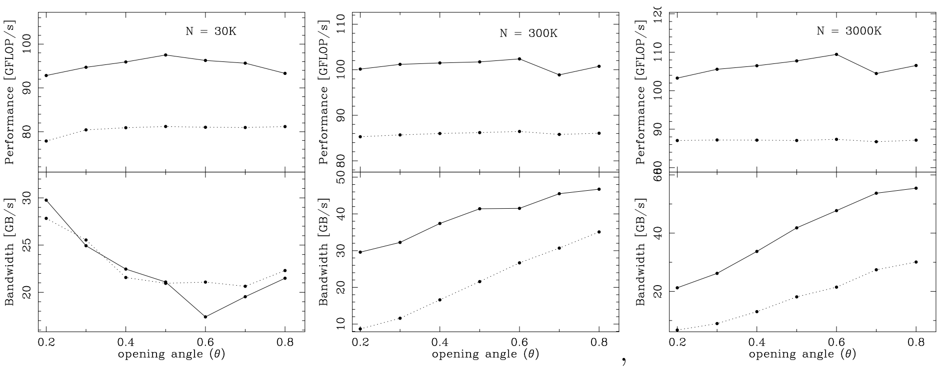


Figure 3.8: Device utilisation as a function of the opening angle and number of particles. Each bottom panel shows the bandwidth for the first tree walk (solid line) and the second tree walk (dotted line). The top halves show the performance of the calculation of the accelerations for the node interaction list (solid line) and the leaf interaction list (dotted line) in GFLOP/s.

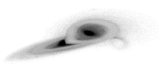
during the first tree walk, in which only (cached) scatter memory reads are executed. The second tree walk, which constructs the interaction list, is notably slower because there data is written to memory–an operation which is significantly slower compared to reads from texture memory. We observe that the bandwidth decreases with θ in both tree walks, which is due to increasingly divergent tree-paths between neighbouring groups, and an increase of the write to read ratio in memory operations.

3.4 Discussion and Conclusions

We present a fast gravitational tree-code which is executed on Graphics Processing Units (GPU) supporting the NVIDIA CUDA architecture. The novelty of this code is the GPU-based tree-walk which, combined with the GPU-based calculation of accelerations, shows good scaling for various particle numbers and different opening angles θ. The hereby produced energy error is comparable to existing CPU based tree-codes with quadrupole corrections. The code makes optimal use of the available device resources and shows excellent scaling to new architectures. Tests indicate that the NVIDIA GT200 architecture, which has roughly twice the resources as the used G80 architecture, performs the integration twice as fast. Specifically, the sustained science rate on a realistic galaxy merger simulation with $8 \cdot 10^5$ particles is $1.6 \cdot 10^6$ particles/second. Our tests revealed our GPU implementation to be two order of magnitudes faster than the widely-used CPU version of the Barnes-Hut tree-code from the NEMO stellar dynamics package (Teuben 1995). However, our code is only an order of magnitude faster compared to a SSE-vectorised tree-code specially tuned for x86-64 processors and the Phantom-GRAPE library [5] Hamada et al (Hamada et al. 2009a) presented a similarly tuned tree code, in which Phantom-grape is replaced with a GPU library (Hamada and Iitaka 2007). In this way, it was possible to

---

[5] Private communication with Keigo Nitadori, the author of the Phantom-GRAPE library.

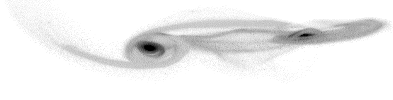


achieve the 200 GFLOP/s, and science rate of about 10^6 particles/s. However, this code does not include quadrupole corrections, and therefore GFLOP/s comparison between the two codes is meaningless. Nevertheless, the science rate of the two codes is comparable for similar opening angles, which implies that our tree-code provides more accurate results for the same performance.

As it generally occurs with other algorithms, the introduction of a massively parallel accelerator usually makes the host calculations and non-parallelisable parts of the code, as small as they may be, the bottleneck. In our case, we used optimised device code and for the host code we used general tree-construction and tree-walk recursive algorithms. It is possible to improve these algorithms to increase the performance of the host part, but it is likely to remain a bottleneck. Even with the use of modern quad-core processors this part is hard to optimize since its largely a sequential operation.

Acknowledgements

We thank Derek Groen, Stefan Harfst and Keigo Nitadori for valuable suggestions and reading of the manuscript. This work is supported by NWO (via grants #635.000.303, #643.200.503, VIDI grant #639.042.607, VICI grant #6939.073.803 and grant #643.000.802). We thank the University of Amsterdam, where part of this work was done, for their hospitality.



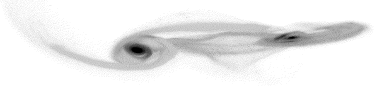
4 | A sparse octree gravitational N-body code that runs entirely on the GPU processor

We present the implementation and performance of a new gravitational N-body tree-code that is specifically designed for the graphics processing unit (GPU)[1] . All parts of the tree-code algorithm are executed on the GPU. We present algorithms for parallel construction and traversing of sparse octrees. These algorithms are implemented in CUDA and tested on NVIDIA GPUs, but they are portable to OpenCL and can easily be used on many-core devices from other manufacturers. This portability is achieved by using general parallel-scan and sort methods. The gravitational tree-code outperforms tuned CPU code during the tree-construction and shows a performance improvement of more than a factor 20 overall, resulting in a processing rate of more than 2.8 million particles per second.

Jeroen Bédorf, Evghenii Gaburov and Simon Portegies Zwart
Journal of Computational Physics, Volume 231, Issue 7, p. 2825-2839, March 2012.

---

[1] The code is publicly available at:
`http://castle.strw.leidenuniv.nl/software.html`



4.1 Introduction

A common way to partition a three-dimensional domain is the use of octrees, which recursively subdivide space into eight octants. This structure is the three-dimensional extension of a binary tree, which recursively divides the one dimensional domain in halves. One can distinguish two types of octrees, namely dense and sparse. In the former, all branches contain an equal number of children and the structure contains no empty branches. A sparse octree is an octree of which most of the nodes are empty (like a sparse matrix), and the structure is based on the underlying particle distribution. In this paper we will only focus on sparse octrees which are quite typical for non-homogenous particle distributions.

Octrees are commonly used in applications that require distance or intersection based criteria. For example, the octree data-structure can be used for the range search method de Berg et al. (2000). On a set of N particles a range search using an octree reduces the complexity of the search from $\mathcal{O}(N)$ to $\mathcal{O}(\log N)$ per particle. The former, though computationally expensive, can easily be implemented in parallel for many particles. The later requires more communication and book keeping when developing a parallel method. Still for large number of particles ($\sim N \geqslant 10^5$) hierarchical[2] methods are more efficient than brute force methods. Currently parallel octree implementations are found in a wide range of problems, among which self gravity simulations, smoothed particle hydrodynamics, molecular dynamics, clump finding, ray tracing and voxel rendering; in addition to the octree data-structure these problems often require long computation times. For high resolution simulations ($\sim N \geqslant 10^5$) 1 (Central Processing Unit) CPU is not sufficient. Therefore one has to use computer clusters or even supercomputers, both of which are expensive and scarce. An attractive alternative is a Graphics Processing Unit (GPU).

Over the years GPUs have grown from game devices into more general purpose compute hardware. With the GPUs becoming less specialised, new programming languages like Brook Buck et al. (2004b), CUDA NVIDIA (2010) and OpenCL Khronos Group Std. (2010) were introduced and allow the GPU to be used efficiently for non-graphics related problems. One has to use these special programming languages in order to be able to get the most performance out of a GPU. This can be realized by considering the enormous difference between today's CPU and GPU. The former has up to 8 cores which can execute two threads each, whereas a modern GPU exhibits hundreds of cores and can execute thousands of threads in parallel. The GPU can contain a large number of cores, because it has fewer resources allocated to control logic compared to a general purpose CPU. This limited control logic renders the GPU unsuitable for non-parallel problems, but makes it more than an order of magnitude faster than the CPU on massively parallel problems NVIDIA (2010). With the recent introduction of fast double precision floating point operations, L1 and L2 caches and ECC memory the GPU has become a major component in the High Performance Computing market. The number of dedicated GPU clusters is steadily increasing and the latest generation of supercomputers have nodes equipped with GPUs, and have established themselves in the upper regions of the top500[3].

This wide spread of GPUs can also be seen in the acceptance of GPUs in computa-

---

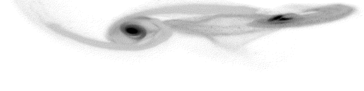
[2] Tree data-structures are commonly referred to as hierarchical data-structures

[3] Top500 Supercomputing November 2010 list, http://www.top500.org

tional astrophysics research. For algorithms with few data dependencies, such as direct N-body simulations, programming the GPU is relatively straightforward. Here various implementations are able to reach almost peak-performance Portegies Zwart et al. (2007); Hamada and Iitaka (2007); Belleman et al. (2008) and with the introduction of N-body libraries the GPU has taken over the GRAPE (GRAvity PipE Makino and Taiji (1998)) [4] as preferred computation device for stellar dynamics Gaburov et al. (2009). Although it is not a trivial task to efficiently utilise the computational power of GPUs, the success with direct N-body methods shows the potential of the GPU in practice. For algorithms with many data dependencies or limited parallelism it is much harder to make efficient use of parallel architectures. A good example of this are the gravitational tree-code algorithms which were introduced in 1986 Barnes and Hut (1986) as a sequential algorithm and later extended to make efficient use of vector machines Barnes (1990). Around this time the GRAPE hardware was introduced which made is possible to execute direct N-body simulations at the same speed as a simulation with a tree-code implementation, while the former scales as $\mathcal{O}(N^2)$ and the latter as $\mathcal{O}(N \log N)$. The hierarchical nature of the tree-code method makes it difficult to parallelise the algorithms, but it is possible to speed-up the computational most intensive part, namely the computation of gravitational interactions. The GRAPE hardware, although unsuitable for constructing and traversing the tree-structure, is able to efficiently compute the gravitational interactions. Therefore a method was developed to create lists of interacting particles on the host and then let the GRAPE solve the gravitational interactions Fukushige et al. (1991); Makino (2004). Recently this method has successfully been applied to GPUs Hamada et al. (2009c,a); Hamada and Nitadori (2010). With the GPU being able to efficiently calculate the force interactions, other parts like the tree-construction and tree-traverse become the bottleneck of the application. Moving the data intensive tree-traverse to the GPU partially lifts this bottleneck Gaburov et al. (2010); Yokota and Barba (2011); (Chapter 3). This method turns out to be effective for shared time-step integration algorithms, but is less effective for block time-step implementations. In a block time-step algorithm not all particles are updated at the same simulation time-step, but only when required. This results in a more accurate (less round-off errors, because the reduced number of interactions) and more efficient (less unnecessary time-steps) simulation. The number of particles being integrated per step can be a fraction of the total number of particles which significantly reduces the amount of parallelism. Also the percentage of time spent on solving gravitational interactions goes down and other parts of the algorithm (e.g. construction, traversal and time integration) become more important. This makes the hierarchical tree N-body codes less attractive, since CPU-GPU communication and tree-construction will become the major bottlenecks Belleman et al. (2008); Gaburov et al. (2010). One solution is to implement the tree-construction on the GPU as has been done for surface reconstruction Zhou et al. (2008) and the creation of bounding volume hierarchies Lauterbach et al. (2009); Pantaleoni and Luebke (2010). An other possibility is is to implement all parts of the algorithm on the GPU using atomic operations and particle insertions Burtscher and Pingali (2011) here the authors, like us, execute all parts of the algorithm on the GPU. When we were in the final stages of finish-

---

[4] The GRAPE is a plug-in board equipped with a processor that has the gravitational equations programmed in hardware.



ing the paper we were able to test the implementation by Burtscher et al. Burtscher and Pingali (2011). It is difficult to compare the codes since they have different monopole expansions and multipole acceptance criteria (see Sections 4.3.2 and 4.3.3). However, even though our implementation has higher multipole moments (quadrupole versus monopole) and a more strict multipole acceptance criteria it is at least 4 times faster.

In this work we devised algorithms to execute the tree-construction on the GPU instead of on the CPU as is customarily done. In addition we redesign the tree-traverse algorithms for effective execution on the GPU. The time integration of the particles is also executed on the GPU in order to remove the necessity of transferring data between the host and the GPU completely. This combination of algorithms is excellently suitable for shared and block time-step simulations. Although here implemented as part of a gravitational N-body code (called `Bonsai`, Section 4.3), the algorithms are applicable and extendable to related methods that use hierarchical data structures.

4.2 Sparse octrees on GPUs

The tree construction and the tree-traverse rely on scan algorithms, which can be efficiently implemented on GPUs. (4.A). Here we discuss the main algorithms that can be found in all hierarchical methods. Starting with the construction of the tree-structure in Section 4.2.1, followed by the method to traverse the previously built tree-structure in Section 4.2.2. The methods that are more specific for a gravitational N-body tree-algorithm are presented in Section 4.3.

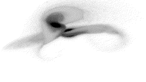
4.2.1 Tree construction

The common algorithm to construct an octree is based on sequential particle insertion Barnes and Hut (1986) and is in this form not suitable for massively parallel processors. However, a substantial degree of parallelism can be obtained if the tree is constructed layer-by-layer[5] from top to bottom. The construction of a single tree-level can be efficiently vectorised which is required if one uses massively parallel architectures.

To vectorise the tree construction particles have to be mapped from a spatial representation to a linear array while preserving locality. This implies that particles that are nearby in 3D space also have to be nearby in the 1D representation. Space filling curves, which trace through the three dimensional space of the data enable such reordering of particles. The first use of space filling curves in a tree-code was presented by Warren and Salmon (1993) Warren and Salmon (1993) to sort particles in a parallel tree-code for the efficient distribution of particles over multiple systems. This sorting also improves the cache-efficiency of the tree-traverse since most particles that are required during the interactions are stored locally, which improves caching and reduces communication. We adopt the Morton space filling curve (also known as Z-order) Morton (1966), because of the existence of a one-to-one map between N-dimensional coordinates and the corresponding 1D Morton-key. The Morton-keys give a 1D representation of the original ND

---

[5] A tree-structure is built-up from several layers (also called levels), with the top most level called the root, the bottom levels leaves and in between the nodes.



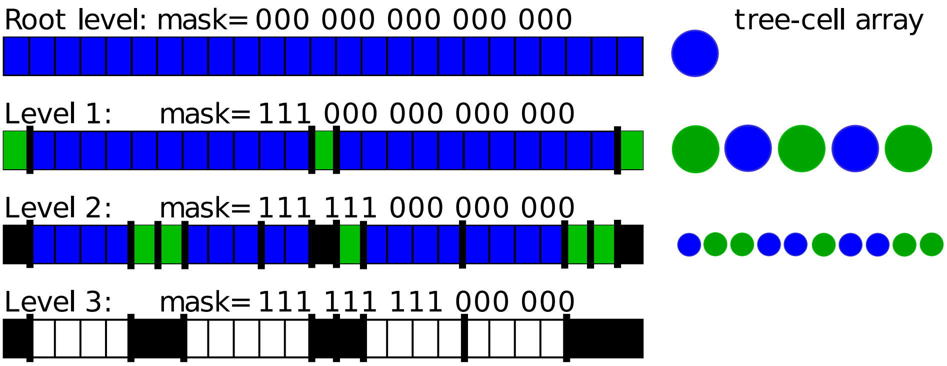


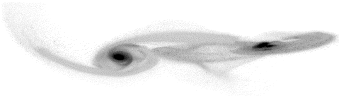
Figure 4.1: Schematic representation of particle grouping into a tree cell. (Left panel) The particles Morton keys are masked[6] with the level mask. Next the particles with the same masked keys are grouped together into cells (indicated by a thick separator, such as in level 1). Cells with less than $N_{\rm leaf}$ particles are marked as leaves (green), and the corresponding particles are flagged (black boxes as in level 2 and 3) and not further used for the tree construction. The other cells are called nodes (blue) and their particles are used constructing the next levels. The tree construction is complete as soon as all particles are assigned to leafs, or when the maximal depth of the tree is reached. (Right panel) The resulting array containing the created tree cells.

coordinate space and are computed using bit-based operations (4.B). After the keys are calculated the particles are sorted in increasing key order to achieve a Z-ordered particle distribution in memory. The sorting is performed using the radix-sort algorithm (see for our implementation details 4.A), which we selected because of its better performance compared to alternative sorting algorithms on the GPU Satish et al. (2009); Merill and Grimshaw (2010). After sorting the particles have to be grouped into tree cells. In Fig. 4.1 (left panel), we schematically demonstrate the procedure. For a given level, we mask the keys[6] of non-flagged particles (non-black elements of the array in the figure), by assigning one particle per GPU thread. The thread fetches the precomputed key, applies a mask (based on the current tree level) and the result is the octree cell to which the particle should be assigned. Particles with identical masked keys are grouped together since they belong to the same cell. The grouping is implemented via the parallel compact algorithm (4.A). We allow multiple particles to be assigned to the same cell in order to reduce the size of the tree-structure. The maximum number of particles that is assigned to a cell is $N_{\rm leaf}$, which we set to $N_{\rm leaf} = 16$. Cells containing up-to $N_{\rm leaf}$ particles are marked as leaves, otherwise they are marked as nodes. If a particle is assigned to a leaf the particle is flagged as complete (black elements of the array in the figure). The masking and grouping procedure is repeated for every single level in serial until all particles are assigned to leaves or that the maximal depth of the tree is reached, whichever occurs first. When all particles are assigned to leaves all required tree cells have been created and are stored in a continuous array (right panel of Fig. 4.1).

However, to complete the tree-construction, the parent cells need to be linked to their

---

[6] The masking is a bitwise operation that preserves the bits which are specified by the mask, for example masking “1011b” with “1100b” results in “1000b”.



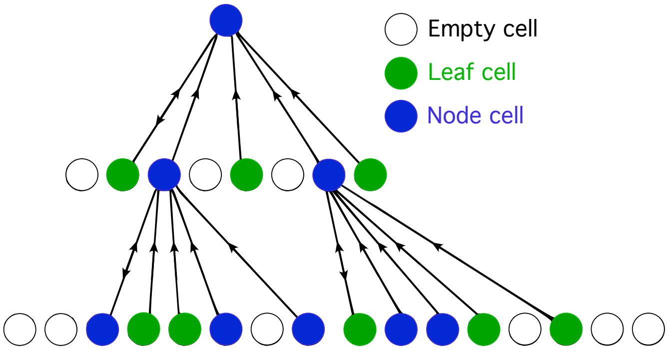


Figure 4.2: Schematic illustration of the tree link process. Each cell of the tree, except the empty cells which are not stored, is assigned to a GPU thread. The thread locates the first child, if it exists, and the parent of the cell. The threads increment the child counter of the parent (indicated by the up arrows) and store the first child in the memory of the cell. This operation requires atomic read-modify-write operations because threads concurrently modify data at the same memory location.

children. We use a separate function to connect the parent and child cells with each other (linking the tree). This function assigns a cell to a single thread which locates its parent and the first child if the cell is not a leaf (Fig. 4.2). This is achieved by a binary search of the appropriate Morton key in the array of tree-cells, which is already sorted in increasing key order during the construction phase. The thread increases the child counter of its parent cell and stores the index of the first child. To reduce memory, we use a single integer to store both the index to the first child and the number of children (most significant 3 bits). If the cell is a leaf, we store the index of the first particle instead of the index of the first child cell, together with the number of particles in the leaf. During these operations many threads concurrently write data to the same memory location. To prevent race conditions, we apply atomic read-modify-write instructions for modifying the data. At the end of this step, the octree is complete and can be used to compute gravitational attractions.

4.2.2 Tree traverse

To take advantage of the massively parallel nature of GPUs we use breadth first, instead of the more common depth first, traversal. Both breadth first and depth first can be parallelised, but only the former can be efficiently vectorised. To further vectorise the calculation we do not traverse a single particle, but rather a group of particles. This approach is known as Barnes' vectorisation of the tree-traverse Barnes (1990). The groups are based on the tree-cells to take advantage of the particle locality as created during the tree-construction. The number of particles in a group is $\leqslant N_{\mathrm{crit}}$ which is typically set to 64 and the total number of groups is N_{groups}. The groups are associated with a GPU thread-block where the number of threads in a block is N_{block} with $N_{\mathrm{block}} >= N_{\mathrm{crit}}$. Hereby we assume that thousands of such blocks are able to run in parallel. This assumption is valid for CUDA-enabled GPUs, as well as on AMD Radeon GPUs via OpenCL. If the code is executed with shared time-steps, all particles are updated at the same time and subsequently all groups are marked as active, for block time-steps this number varies between 1 and N_{groups}.

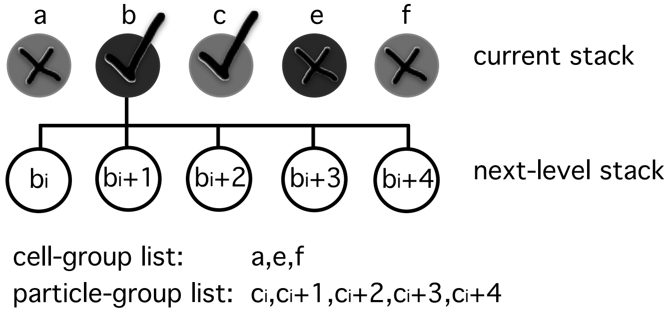


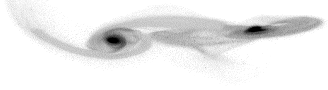
Figure 4.3: Illustration of a single level tree-traverse. There are five cells in the current stack. Those cells which are marked with crosses terminate traverse, and therefore are added to the cell-group list for subsequent evaluation. Otherwise, if the cell is a node, its children are added to the next-level stack. Because children are contiguous in the tree-cell array, they are named as b_i, $b_i + 1$, …, $b_i + 4$, where b_i is the index of the first child of the node b. If the cell is a leaf, its particles are added to the particle-cell interaction list. Because particles in a leaf are also contiguous in memory, we only need to know the index of the first particle, c_i, and the number of particles in a leaf, which is 5 here.

Each thread block executes the same algorithm but on a different set of particles.

Each thread in a block reads particle data that belongs to the corresponding group as well as group information which is required for the tree traverse. If the number of particles assigned to a group is smaller than N_{block} by a factor of two or more, we use multiple threads (up to 4) per particle to further parallelise the calculations. As soon as the particle and group data is read by the threads we proceed with the tree-traverse.

On the CPU the tree-traverse algorithm is generally implemented using recursion, but on the GPU this is not commonly supported and hard to parallelise. Therefore we use a stack based breadth first tree-traverse which allows parallelisation. Initially, cells from one of the topmost levels of the tree are stored in the current-level stack and the next-level stack is empty; in principle, this can be the root level, but since it consists of one cell, the root node, only one thread from N_{block} will be active. Taking a deeper level prevents this and results in more parallelism. We loop over the cells in the current-level stack with increments of N_{block}. Within the loop, the cells are distributed among the threads with no more than one cell per thread. A thread reads the cell's properties and tests whether or not to traverse the tree further down from this cell; if so, and if the cell is a node the indexes of its children are added to the next-level stack. If however, the cell is a leaf, the indexes of constituent particles are stored in the particle-group interaction list. Should the traverse be terminated then the cell itself is added to cell-group interaction list (Fig.4.3).

The cell-group interaction list is evaluated when the size of the list exceeds N_{block}. To achieve data parallelism each thread reads properties of an interacting cell into fast low-latency on-chip memory that can be shared between the threads, namely shared memory (CUDA) or local memory (OpenCL). Each thread then progresses over the data in the on-chip memory and accumulates partial interactions on its particle. At the end of this pass, the size of the interaction list is decreased by N_{block} and a new pass is started until the size of the list falls below N_{block}. The particle-group interaction list is evaluated in exactly the same way, except that particle data is read into shared memory instead of cell data.



This is a standard data-sharing approach that has been used in a variety of N-body codes, e.g. Nyland et al. Nyland et al. (2007).

The tree-traverse loop is repeated until all the cells from the current-level stack are processed. If the next-level stack is empty, the tree-traverse is complete, however if the next-level stack is non-empty its data is copied to the current-level stack. The next-level stack is cleaned and the current-level stack is processed. When the tree-traverse is complete either the cell-group, particle-group or both interaction lists may be non-empty. In such case the elements in these lists are distributed among the threads and evaluated in parallel. Finally if multiple threads per particle are used an extra reduction step combines the results of these threads to get the final interaction result.

4.3 Gravitational Tree-code

To demonstrate the feasibility and performance, we implement a version of the gravitational Barnes-Hut tree-code Barnes and Hut (1986) which is solely based on the sparse octree methods presented in the previous sections. In contrast to other existing GPU codes Hamada et al. (2009a); Gaburov et al. (2010) and Chapter 3, our implementation runs entirely on GPUs. Apart from the previous described methods to construct and traverse the tree-structure we implement time integration (Section 4.3.1), time-step calculation and tree-cell properties computation on the GPU (Section 4.3.2). The cell opening method, which sets the accuracy and performance of the tree-traverse, is described in Section 4.3.3.

4.3.1 Time Integration

To move particles forward in time we apply the leapfrog predictor-corrector algorithm described by Hut et al. (1995) Hut et al. (1995). Here the position and velocity are predicted to the next simulation time using previously calculated accelerations. Then the new accelerations are computed (tree-traverse) and the velocities are corrected. This is done for all particles in parallel or on a subset of particles in case of the block-time step regime. For a cluster of $\gtrsim 10^5$ particles, the time required for one prediction-correction step is less than 1% of the total execution time and therefore negligible.

4.3.2 Tree-cell properties

Tree-cell properties are a summarized representation of the underlying particle distribution. The multipole moments are used to compute the forces between tree-cells and the particles that traverse the tree. In this implementation of the Barnes-Hut tree-code we use only monopole and quadrupole moments McMillan and Aarseth (1993). Multipole moments are computed from particle positions and need to be recomputed at each time-step; any slowdown in their computation may substantially influence the execution time. To parallelise this process we initially compute the multipole moments of each leaf in parallel. We subsequently traverse the tree from bottom to top. At each level the multipole moments of the nodes are computed in parallel using the moments of the cells one level below (Fig. 4.4). The number of GPU threads used per level is equal to the number of

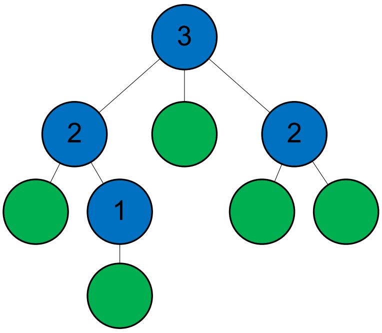


Figure 4.4: Illustration of the computation of the multipole moments. First the properties of the leaves are calculated (green circles). Then the properties of the nodes are calculated level-by-level from bottom to top. This is indicated by the numbers in the nodes, first we compute the properties of the node with number 1, followed by the nodes with number 2 and finally the root node.

nodes at that level. These computations are performed in double precision since our tests indicated that the NVIDIA compiler aggressively optimises single precision arithmetic operations, which results in an error of at most 1% in the multipole moments. Double precision arithmetic solved this problem and since the functions are memory-bound[7] the overhead is less than a factor 2. As final step the double precision values are converted back to single precision to be used during the tree-traverse.

4.3.3 Cell opening criterion

In a gravitational tree-code the multipole acceptance criterion (MAC) decides whether to use the multipole moments of a cell or to further traverse the tree. This influences the precision and execution time of the tree-code. The further the tree is traversed the more accurate the computed acceleration will be. However, traversing the tree further results in a higher execution time since the number of gravitational interactions increases. Therefore the choice of MAC is important, since it tries to determine, giving a set of parameters, when the distance between a particle and a tree cell is large enough that the resulting force approximation error is small enough to be negligible. The MAC used in this work is a combination of the method introduced by Barnes (1994) Barnes (1994) and the method used for tree-traversal on vector machines Barnes (1990). This gives the following acceptance criterion,

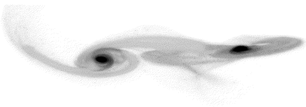
$$d > \frac{l}{\theta} + \delta \tag{4.1}$$

where d is the smallest distance between a group and the center of mass of the cell, l is the size of the cell, θ is a dimensionless parameter that controls the accuracy and δ is the distance between the cell's geometrical center and the center of mass. If d is larger than the right side of the equation the distance is large enough to use the multipole moment instead of traversing to the child cells.

Fig. 4.5 gives an overview of this method. We also implemented the minimal distance MAC Salmon and Warren (1994), which results in an acceleration error that is between

---

[7] On GPUs we distinguish two kind of performance limitations, memory-bound and compute-bound. In the former the performance is limited by the memory speed and memory bandwidth, in the later the performance is limited by the computation speed.



Hardware model	Xeon E5620	8800 GTS512	C1060	GTX285	C2050	GTX480
Architecture	Gulftown	G92	GT200	GT200	GF100	GF100
# Cores	4	128	240	240	448	480
Core (Mhz)	2400	1625	1296	1476	1150	1401
Memory (Mhz)	1066	1000	800	1243	1550	1848
Interface (bit)	192	256	512	512	384	384
Bandwidth (GBs)	25.6	64	102	159	148	177.4
Peak (GFLOPs)[2]	76.8	624	933	1063	1030	1345
Memory size (GB)	16	0.5	4	1	2.5	1.5

[1] All calculations in this work are done in single precision arithmetic.

[2] The peak performance is calculated as follows:

-Gulftown: #Cores × Core speed × 8 (SSE flops/cycle)
-G92 & GT200: #Cores × Core speed × 3 (flops/cycle)
-GF100: #Cores × Core speed × 2 (flops/cycle)

Table 4.1: Used hardware. The Xeon is the CPU in the host system, the other five devices are GPUs.

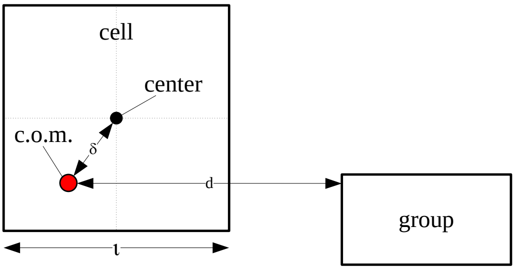


Figure 4.5: Illustration of the computation of the multipole moments. First the properties of the leaves are calculated (green circles). Then the properties of the nodes are calculated level-by-level from bottom to top. This is indicated by the numbers in the nodes, first we compute the properties of the node with number 1, followed by the nodes with number 2 and finally the root node.

10% and 50% smaller for the same θ than the MAC used here. The computation time, however, is almost a factor 3 higher since more cells are accepted (opened).

The accuracy of the tree-traverse is controlled by the parameter θ. Larger values of θ causes fewer cells to be opened and consequently results in a shallower tree-traverse and a faster evaluation of the underlying simulation. Smaller values of θ have the exact opposite effect, but result in a more accurate integration. In the hypothetical case that all the tree cells are opened ($\theta \to 0$) the tree-code turns in an inefficient direct N-body code. In Section 4.4.1 we adopt $\theta = 0.5$ and $\theta = 0.75$ to show the dependence of the execution time on the opening angle. In Section 4.4.2 we vary θ between 0.2 and 0.8 to show the dependence of the acceleration error on θ.

4.4 Performance and Accuracy

In this section we compare the performance of our implementation of the gravitational N-body code (`Bonsai`) with CPU implementations of comparable algorithms. Furthermore, we use a statistical test to compare the accuracy of `Bonsai` with a direct summation code. As final test `Bonsai` is compared with a direct N-body code and a set of N-body tree-codes using a production type galaxy merger simulation.

Even though there are quite a number of tree-code implementations each has its own specific details and it is therefore difficult to give a one-to-one comparison with other

tree-codes. The implementations closest to this work are the parallel CPU tree-code of John Dubinski (1996) Dubinski (1996) (`Partree`) and the GPU accelerated tree-code `Octgrav` Gaburov et al. (2010), also see Chapter 3. Other often used tree-codes either have a different MAC or lack quadrupole corrections. The default version of `Octgrav` has a different MAC than `Bonsai`, but for the galaxy merger simulation a version of `Octgrav` is used that employs the same method as `Bonsai` (Section 4.3.3). We use `phiGRAPE` Harfst et al. (2007) in combination with the `Sapporo` Gaburov et al. (2009) direct N-body GPU library for the comparison with direct N-body simulations. Although here used as standalone codes, most of them are part of the AMUSE frameworkPortegies Zwart et al. (2009), as will be a future version of `Bonsai` which would make the comparison trivial to execute.

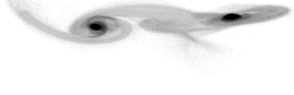
The hardware used to run the tests is presented in Table 4.1. For the CPU calculations we used an Intel Xeon E5620 CPU which has 4 physical cores. For GPUs, we used 1 GPU with the G92 architecture (GeForce 8800GTS512), 2 GPUs with the GT200 architecture (GeForce 285GTX and Tesla C1060), and 2 GPUs with the GF100 architecture (GTX480 and Tesla C2050). All these GPUs are produced by NVIDIA. The Tesla C2050 GPU is marketed as a professional High Performance Computing card and has the option to enable error-correcting code memory (ECC). With ECC enabled extra checks on the data are conducted to prevent the use of corrupted data in computations, but this has a measurable impact on the performance. Therefore, the tests on the C2050 are executed twice, once with and once without ECC enabled to measure the impact of ECC.

All calculations are conducted in single precision arithmetic except for the computation of the monopole and quadrupole moments in `Bonsai` and the force calculation during the acceleration test in `phiGRAPE` for which we use double precision arithmetic.

4.4.1 Performance

To measure the performance of the implemented algorithms we execute simulations using Plummer Plummer (1915) spheres with $N = 2^{15}$ (32k) up to $N = 2^{24}$ (16M) particles (up to $N = 2^{22}$ (4M) for the GTX480, because of memory limitations). For the most time critical parts of the algorithm we measure the wall-clock time. For the tree-construction we distinguish three parts, namely sorting of the keys (sorting), reordering of particle properties based on the sorted keys (moving) and construction and linking of tree-cells (tree-construction). Furthermore, are timings presented for the multipole computation and tree-traverse. The results are presented in Fig. 4.6. The wall-clock time spend in the sorting, moving, tree-construction and multipole computation algorithms scales linearly with N for $N \gtrsim 10^6$. For smaller N, however, the scaling is sub-linear, because the parallel scan algorithms require more than 10^5 particles to saturate the GPU. The inset of Fig. 4.6 shows that the average number of particle-cell interactions doubles between $N \gtrsim$ 32k and $N \lesssim$ 1M and keeps gradually increasing for $N \gtrsim$ 1M. Finally, more than 90% with $\theta = 0.75$ and 95% with $\theta = 0.5$ of the wall-clock time is spent on tree-traversal. This allows for block time-step execution where the tree-traverse time is reduced by a factor $N_{\text{groups}}/N_{\text{active}}$, where N_{active} is the number of groups with particles that have to be updated.

In Fig. 4.7 we compare the performance of the tree-algorithms between the three gen-



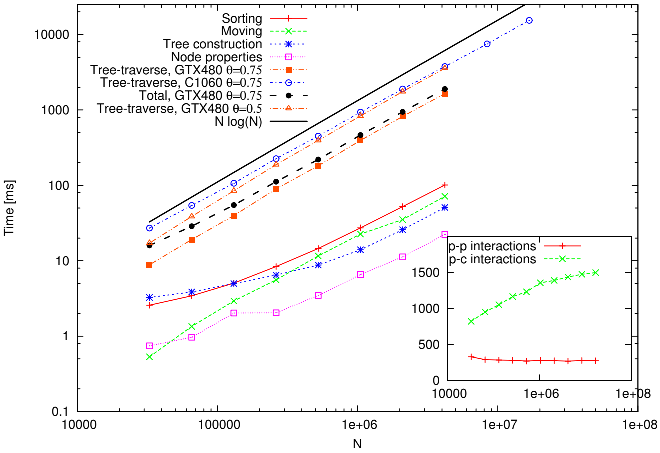
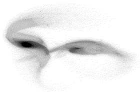


Figure 4.6: The wall-clock time spent by various parts of the program versus the number of particles N. We used Plummer models as initial conditions Plummer (1915) and varied the number of particles over two orders of magnitude. The solid black line, which is offset arbitrarily, shows the theoretical $\mathcal{O}(N \log N)$ scaling Barnes and Hut (1986). The asymptotic complexity of the tree-construction approaches $\mathcal{O}(N)$, because all the constituent primitives share the same complexity. The tree-construction timing comes from the GTX480. To show that the linear scaling continues we added timing data for the C1060, which allows usage of larger data sets. For the GTX480 we included the results of the tree-traverse with $\theta = 0.5$ and the results of the tree-traverse with $\theta = 0.75$. The inset shows the average number of particle-particle and particle-cell interactions for each simulation where $\theta = 0.75$.

erations of GPUs as well as against tuned CPU implementations[8]. For all algorithms the CPU is between a factor of 2 (data reordering) to almost a factor 30 (tree-traverse) slower than the fastest GPU (GTX480). Comparing the results of the different GPUs we see that the GTS512 is slowest in all algorithms except for the data moving phase, in which the C1060 is the slowest. This is surprising since the C1060 has more on-device bandwidth, but the lower memory clock-speed appears to have more influence than the total bandwidth. Overall the GF100 generation of GPUs have the best performance. In particular, during the tree-traverse part, they are almost a factor 2 faster than the GT200 series. This is more than their theoretical peak performance ratios, which are 1.1 and 1.25 for C1060 vs. C2050 and GTX285 vs. GTX480 respectively. In contrast, the GTX285 executes the tree-traverse faster than the C1060 by a factor of 1.1 which is exactly the peak performance ratio between these GPUs. We believe that the difference between the GT200 and GF100 GPUs is mainly caused by the lack of L1 and L2 caches on GT200 GPUs that are present on GF100 GPUs. In the latter, non-coalesced memory accesses are cached, which occur frequently during the tree-traverse, this reduces the need to request data from the

---

[8] The tree-construction method is similar to Warren and Salmon (1993), and was implemented by Keigo Nitadori with OpenMP and SSE support. The tree-traverse is, however, from the CPU version of the MPI-parallel tree-code by John Dubinski Dubinski (1996). It has monopole and quadrupole moments and uses the same multipole acceptance criterion as our code. We ran this code on the Xeon E5620 CPU using 4 parallel processes where each process uses one of the 4 available physical cores.



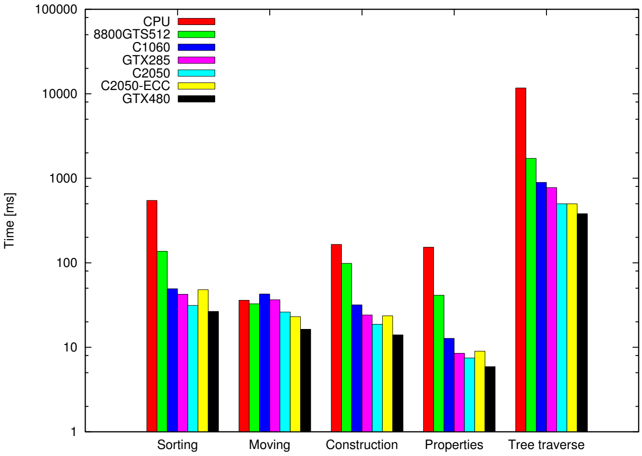


Figure 4.7: Wall-clock time spent by the CPU and different generations of GPUs on various primitive algorithms. The bars show the time spent on the five selected sections of code on the CPU and 5 different GPUs (spread over 3 generations). The results indicate that the code outperforms the CPU on all fronts, and scales in a predictable manner between different GPUs. The `C2050-ECC` bars indicate the runs on the C2050 with ECC enabled, the `C2050` bars indicate the runs with ECC disabled. Note that the y-axis is in log scale. (Timings using a 2^{20} million body Plummer sphere with $\theta = 0.75$)

relatively slow global memory. This is supported by auxiliary tests where the texture cache on the GT200 GPUs is used to cache non-coalesced memory reads, which resulted in a reduction of the tree-traverse execution time between 20 and 30%. Comparing the C2050 results with ECC-memory to those without ECC-memory we notice a performance impact on memory-bound functions that can be as high as 50% (sorting), while the impact on the compute-bound tree-traverse is negligible, because the time to perform the ECC is hidden behind computations. Overall we find that the implementation scales very well over the different GPU generations and makes optimal use of the newly introduced features of the GF100 architecture. The performance of the tree-traverse with $\theta = 0.75$ is 2.1M particles/s and 2.8M particles/s on the C2050 and GTX480 respectively for N = 1M.

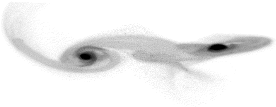
4.4.2 Accuracy

To measure the accuracy of the tree-code we use two tests. In the first, the accelerations due to the tree-code are compared with accelerations computed by direct summation. In the second test, we compared the performance and accuracy of three tree-codes and a direct summation code using a galaxy merger simulation.

Acceleration

To quantify the error in the accelerations between `phiGRAPE` and `Bonsai` we calculate

$$\Delta a/a = |\mathbf{a}_{\text{tree}} - \mathbf{a}_{\text{direct}}|/|\mathbf{a}_{\text{direct}}|, \tag{4.2}$$



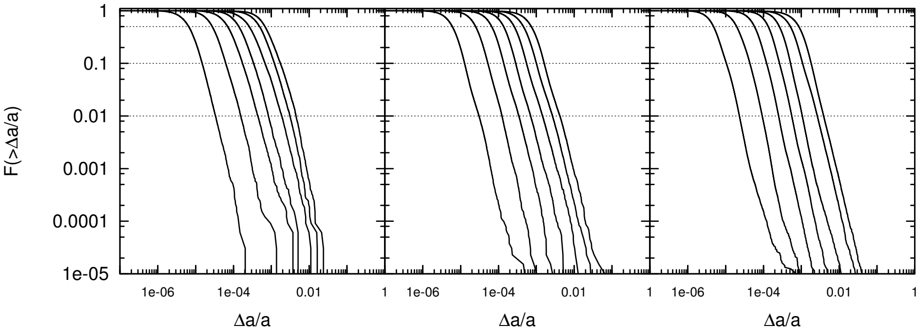


Figure 4.8: Each panel displays a fraction of particles, $F(>\Delta a/a)$, having a relative acceleration error, $\Delta a/a$, (vertical axis) greater than a specified value (horizontal axis). In each panel the solid lines show the errors for various opening angles, from left to right $\theta = 0.2$, 0.3, 0.4, 0.5, 0.6, 0.7 and 0.8. The panels show, from left to right, simulations with $N = 32768$, $N = 131072$ and $N = 1048576$ particles. The dotted horizontal lines indicate 50%, 10% and 1% of the error distribution.

where $\mathbf{a}_{\text{tree}}$ and $\mathbf{a}_{\text{direct}}$ are accelerations obtained by tree and direct summation respectively. The direct summation results are computed with a double precision version of `Sapporo`, while for tree summation single precision is used. For both methods the softening is set to zero and a GTX480 GPU is used as computation device.

In Fig. 4.8 the error distribution for different particle numbers and opening angles is shown. Each panel shows the fraction of particles (vertical-axis) having a relative acceleration error larger than a given value (horizontal-axis). The three horizontal dotted lines show the 50th, 10th and 1st percentile of the cumulative distribution (top to bottom). The results indicate that the acceleration error is slightly smaller (less than an order of magnitude) than `Octgrav` and comparable to CPU tree-codes with quadrupole corrections Dehnen (2002); Springel et al. (2001); Stadel (2001). In `Octgrav` a different MAC is used than in `Bonsai` which explains the better accuracy results of `Bonsai`.

The dependence of the acceleration error on θ and the number of particles is shown in Fig. 4.9. Here the median and first percentile of the relative acceleration error distributions of Fig. 4.8 are plotted as a function of θ. The figure shows that the relative acceleration error is nearly independent of N, which is a major improvement compared to `Octgrav` where the relative acceleration error clearly depends on N (Figure 5 in Gaburov et al. (2010)). The results are consistent with those of `Partree` Dubinski (1996) which uses the same MAC.

Galaxy merger

A realistic comparison between the different N-body codes, instead of statistical tests only, is performed by executing a galaxy merger simulation. The merger consists of two galaxies, each with 10^5 dark matter particles, 2×10^4 star particles and one super massive black hole (for a total of 240.002 bodies). The galaxies have a 1:3 mass ratio and the pericenter is 10kpc. The merger is simulated with `Bonsai`, `Octgrav`, `Partree` and `phiGRAPE`. The used hardware for `Bonsai` and `Octgrav` was 1 GTX480, `Partree` used 4 cores of

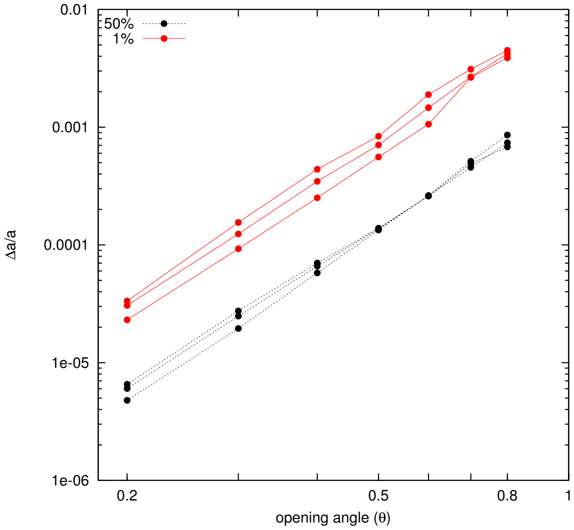


Figure 4.9: The median and the first percentile of the relative acceleration error distribution as a function of the opening angle and the number of particles. We show the lines for $N = 32768$ (bottom striped and solid line) $N = 131072$ (middle striped and dotted line) and $N = 1048576$ (top striped and dotted line).

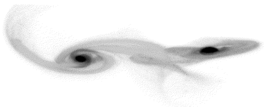
the Intel Xeon E5620 and `phiGRAPE` used 4 GTX480 GPUs. With each tree-code we ran two simulations, one with $\theta = 0.5$ and one with $\theta = 0.75$. The end-time of the simulations is $T = 1000$ with a shared time-step of $\frac{1}{64}$ (resulting in 64000 steps) and a gravitational softening of $\epsilon = 0.1$. For `phiGRAPE` the default time-step settings were used, with the maximum time-step set to $\frac{1}{16}$ and softening set to $\epsilon = 0.1$. The settings are summarised in the first four columns of Table 4.2.

We compared the density, cumulative mass and circular velocity profiles of the merger product as produced by the different simulation codes, but apart from slight differences caused by small number statistics the profiles are identical. As final comparison we recorded the distance between the two black holes over the course of the simulation. The results of which are shown in the bottom panel of Fig. 4.10, the results are indistinguishable up to the third pericenter passage at $t = 300$ after which the results, because of numerical differences, become incomparable. Apart from the simulation results we also compare the energy conservation. This is done by computing the relative energy error (dE, Eq. 4.3) and the maximal relative energy error (dE_{max}).

$$dE = \frac{E_0 - E_t}{E_0} \tag{4.3}$$

Here E_0 is the total energy (potential energy + kinetic energy) at the start of the simulation and E_t is the total energy at time t. The time is in N-body units.

The maximal relative energy error is presented in the top panel of Fig. 4.10, the tree-code simulations with $\theta = 0.75$ give the highest dE_{max} which occurs for both $\theta = 0.75$ and $\theta = 0.5$ during the second pericenter passage at $t \approx 280$. For $\theta = 0.5$, the dE_{max} is roughly a factor 2 smaller than for $\theta = 0.75$. For `phiGRAPE` dE_{max} shows a drift and does not stay constant after the second pericenter passage. The drift in the energy error is caused by the formation of binaries, for which `phiGRAPE` has no special treatment, resulting in the observed drift instead of a random walk.



Simulation	Hardware	dt	θ	dE_{end} $[\times 10^{-4}]$	dE_{max} $[\times 10^{-4}]$	time [s]
phiGRAPE	4x GTX480	block	-	-1.9	1.8	62068
Bonsai run1	1x GTX480	$\frac{1}{64}$	0.75	0.21	2.8	7102
Bonsai run2	1x GTX480	$\frac{1}{64}$	0.50	0.44	1.3	12687
Octgrav run1	1x GTX480	$\frac{1}{64}$	0.75	-1.1	3.5	11651
Octgrav run2	1x GTX480	$\frac{1}{64}$	0.50	-1.1	2.2	15958
Partree run1	Xeon E5620	$\frac{1}{64}$	0.75	-3.5	3.8	118424
Partree run2	Xeon E5620	$\frac{1}{64}$	0.50	0.87	0.96	303504

Table 4.2: Settings and results of the galaxy merger. The first two columns indicate the software and hardware used, the third the time-step (dt) and the fourth the opening angle (θ). The last three columns present the results, energy error at the $time = 1000$ (fifth column), maximum energy error during the simulation (sixth column) and the total execution time (seventh column).

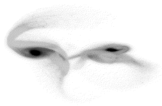
A detailed overview of the energy error is presented in Fig. 4.11 which shows the relative energy error (dE) over the course of the simulation. Comparing the dE of the tree-codes shows that `Bonsai` has a more stable evolution than `Octgrav` and `Partree`. Furthermore if we compare the results of $\theta = 0.75$ and $\theta = 0.5$ there hardly is any improvement visible for `Octgrav` while `Bonsai` and `Partree` show an energy error with smaller per time-step variance of the energy error.

The last thing to look at is the execution time of the various codes, which can be found in the last column of Tab. 4.2. Comparing `Bonsai` with `Octgrav` shows that the former is faster by a factor of 1.6 and 1.26 for $\theta = 0.75$ and $\theta = 0.5$ respectively. The smaller speed-up for $\theta = 0.5$ results from the fact that the tree-traverse, which takes up most of the execution time, is faster in `Octgrav` than in `Bonsai`. Comparing the execution time of `Bonsai` with that of `Partree` shows that the former is faster by a factor of 17 (24) with $\theta = 0.75$ ($\theta = 0.5$). Note that this speed-up is smaller than reported in Section 4.4.1 due to different initial conditions. Finally, when comparing `phiGRAPE` with `Bonsai`, we find that `Bonsai` completes the simulation on 1 GTX480 faster than `phiGRAPE`, which uses 4 GTX480 GPUs, by a factor of 8.7 ($\theta = 0.75$) and 4.9 ($\theta = 0.5$).

4.5 Discussion and Conclusions

We have presented algorithms to construct and traverse hierarchical data-structures efficiently. These algorithms are implemented as part of a gravitational N-body tree-code. In contrast to other existing GPU tree-codes, this implementation is executed on the GPU. While the code is written in CUDA, the methods themselves are portable to other massively parallel architectures, provided that parallel scan-algorithms exist for such architectures. For this implementation a custom CUDA API wrapper is used that can be replaced with an OpenCL version. As such the code can be ported to OpenCL, by only rewriting the GPU functions, which is currently work in progress.

The number of particles processed per unit time, is 2.8 million particles per second with $\theta = 0.75$ on a GTX480. Combined with the stable energy evolution and efficient



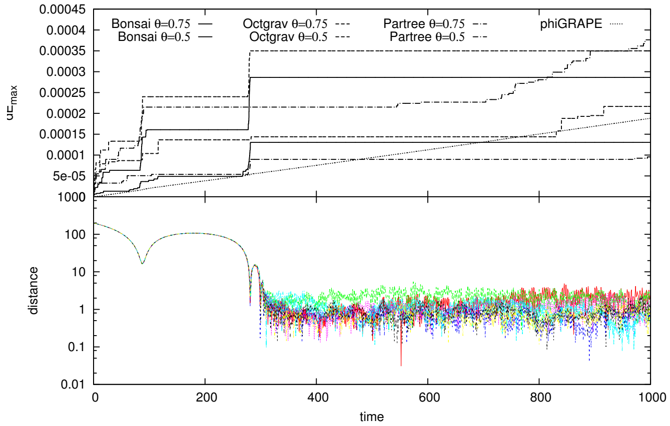
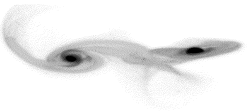


Figure 4.10: The top panel shows the maximal relative energy error over the course of the simulation. The solid lines show the results of `Bonsai`, the striped lines the results of `Octgrav` and the striped-dotted lines show the results of `Partree`. In all cases the top lines show the result of $\theta = 0.75$ and the bottom lines the result of $\theta = 0.5$. Finally the dotted line shows the result of `phiGRAPE`. The bottom panel shows the distance between the two supermassive black-holes. The lines themselves have no labels since up to $t = 300$ the lines follow the same path and after $t = 300$ the motion becomes chaotic and incomparable. The distance and time are in N-body units.

scaling permits us to routinely carry out simulations on the GPU. Since the current version can only use 1 GPU, the limitation is the amount of memory. For 5 million particles ±1 gigabyte of GPU memory is required.

Although the the tree-traverse in `Octgrav` is ±10% faster than in `Bonsai`, the latter is much more appropriate for large ($N > 10^6$) simulations and simulations which employ block-time steps. In `Octgrav` the complete tree-structure, particle array and multipole moments are send to the GPU during each time-step. When using shared-time steps this is a non-critical amount of overhead since the overall performance is dominated by the tree-traverse which takes up more than 90% of the total compute time. However, this balance changes if one uses block time-steps. The tree-traverse time is reduced by a factor $N_{\text{groups}}/N_{\text{active}}$, where N_{active} is the number of groups with particles that have to be updated. This number can be as small as a few percent of N_{groups}, and therefore tree-construction, particle prediction and communication becomes the bottleneck. By shifting these computations to the GPU, this ceases to be a problem, and the required host communication is removed entirely.

Even though the sorting, moving and tree-construction parts of the code take up roughly 10% of the execution time, these methods do not have to be executed during each time-step when using the block time-step method. It is sufficient to only recompute the multipole moments of tree-cells that have updated child particles. Only when the tree-traverse shows a considerable decline in performance the complete tree-structure has to be rebuild. This decline is the result of inefficient memory reads and an increase of the average number of particle-cell and particle-particle interactions. This quantity in-



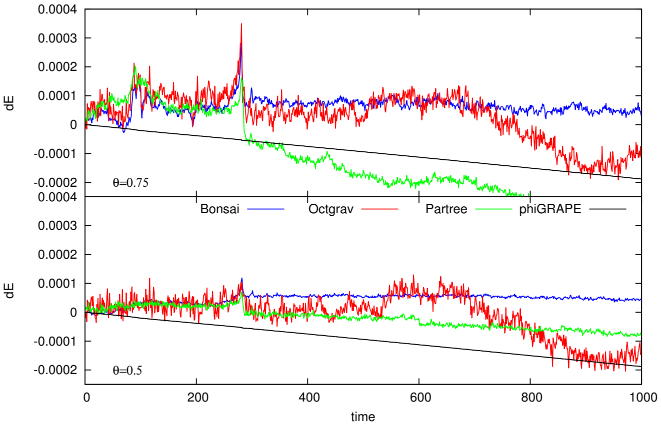


Figure 4.11: Relative energy error over the course of the simulation. The top panel shows the results of $\theta = 0.75$ and the bottom panel the results of $\theta = 0.5$, all other settings as defined in Tab. 4.2. The `Bonsai` results are shown by the blue lines, the `Octgrav` results are shown by the red lines, the `Partree` results are shown by the green lines and the black lines show the results of `phiGRAPE`. Note that the result of `phiGRAPE` is the same in the top and bottom panel and that the range of the y-axis is the same in both panels.

creases because the tree-cell size (l) increases, which causes more cells to be opened by the multipole acceptance criterion (Eq. 4.1).

Although the algorithms described herein are designed for a shared-memory architecture, they can be used to construct and traverse tree-structures on parallel GPU clusters using the methods described in Warren and Salmon (1993); Dubinski (1996). Furthermore, in case of a parallel GPU tree-code, the CPU can exchange particles with the other nodes, while the GPU is traversing the tree-structure of the local data. In this way, it is possible to hide most of the communication time.

The presented tree-construction and tree-traverse algorithms are not limited to the evaluation of gravitational forces, but can be applied to a variety of problems, such as neighbour search, clump finding algorithms, fast multipole method and ray tracing. In particular, it is straightforward to implement Smoothed Particle Hydrodynamics in such a code, therefore having a self-gravitating particle based hydrodynamics code implemented on the GPU.

Acknowledgements

This work is supported by NOVA and NWO grants (#639.073.803, #643.000.802, #614.061.608, #643.200.503, VIDI #639.042.607). The authors would like to thank Massimiliano Fatica and Mark Harris of NVIDIA for the help with getting the code to run on the Fermi architecture, Bernadetta Devecchi for her help with the galaxy merger simulation and Dan Caputo for his comments which improved the manuscript.

4.A Scan algorithms

Both tree-construction and tree-traverse make extensive use of parallel-scan algorithms, also known as parallel prefix-sum algorithms. These algorithms are examples of computations that seem inherently sequential, but for which an efficient parallel algorithm can be defined. Blelloch Guy E. Blelloch (1990) defines the scan operations as follows:

Definition: *The* all-prefix-sums *operation takes a binary associative operator* $\oplus$*, and an array of n elements*

$$[a_0, a_1, ..., a_{n-1}],$$

and returns the ordered set

$$[a_0, (a_0 \oplus a_1), ..., (a_0 \oplus a_1 \oplus ... \oplus a_{n-1})].$$

Example: If $\oplus$ is the addition operator, then the all-prefix-sums operation on the array

[3 1 7 0 4 1 6 3],

would return

[3 4 11 11 15 16 22 25].

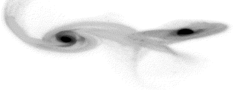
The prefix-sum algorithms form the building blocks for a variety of methods, including stream compaction, stream splitting, sorting, regular expressions, tree-depth determination and histogram construction. In the following paragraphs, we give a concise account on the algorithms we used in this work, however we refer the interested readers to the survey by Blelloch Guy E. Blelloch (1990) for further examples and detailed descriptions.

4.A.1 Stream Compaction

Stream compaction removes "invalid" items from a stream of elements; this algorithm is also known as stream reduction. In the left panel of the Fig.4.12 an example of a compaction is shown where invalid elements are removed and valid elements are placed at the start of the output stream.

4.A.2 Split and Sort

Stream split is related to stream compaction, but differs in that the invalid elements are placed behind the valid ones in the output stream instead of being discarded (right panel of Fig.4.12). This algorithm is used as building block for the radix sort primitive. Namely, for each bit of an integer we call the split algorithm, starting from the least significant bit, and terminating with the most significant bit Knuth (1997).



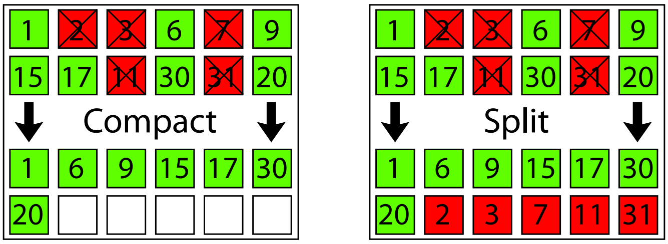


Figure 4.12: Example of compact and split algorithms. The "compact" discards invalid items whereas the "split" places these behind the valid items. We use this "split" primitive for our radix sort implementation because it preserves the item ordering–a property which is fundamental for the radix sort algorithm.

4.A.3 Implementation

All parts of our parallel octree algorithms use scan algorithms in one way or another. Therefore, it is important that these scan algorithms are implemented in the most efficient way and do not become the bottleneck of the application. There are many different implementations of scan algorithms on many-core architectures Mark Harris (NVIDIA) (2009); Sengupta et al. (2008); Billeter et al. (2009). We use the method of Billeter et al. Billeter et al. (2009) for stream compaction, split and radix-sort because it appears to be, at the moment of writing, the fastest and is easily adaptable for our purposes.

Briefly, the method consists of three steps:

1. Count the number of valid elements in the array.
2. Compute output offsets using parallel prefix-sum.
3. Place valid elements at the output offsets calculated in the previous step.

We used the prefix-sum method described by Sengupta et al. Sengupta et al. (2008), for all such operations in both the tree-construction and tree-traverse parts of the implementation.

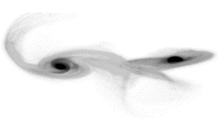
4.B Morton Key generation

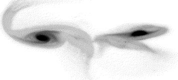
One of the properties of the Morton key is its direct mapping between coordinates and keys. To generate the keys given a set of coordinates one can make use of look-up tables or generate the keys directly. Since the use of look-up tables is relative inefficient on GPUs (because of the many parallel threads that want to access the same memory) we decided to compute the keys directly. First we convert the floating point positions into integer positions. This is done by shifting the reference frame to the lower left corner of the domain and then multiply the new positions by the size of the domain. Then we can apply bit-based dilate primitives to compute the Morton key (List.4.1, Raman and Wise (2008)). This dilate primitive converts the first 10 bits of an integer to a 30 bit representation, i.e. `0100111011` → `000 001 000 000 001 001 001 000 001 001`:

List 4.1: The GPU code which we use to dilate the first 10-bits of an integer.

```
int dilate(const int value) {
  unsigned int x;
  x = value & 0x03FF;
  x = ((x << 16) + x) & 0xFF0000FF;
  x = ((x <<  8) + x) & 0x0F00F00F;
  x = ((x <<  4) + x) & 0xC30C30C3;
  x = ((x <<  2) + x) & 0x49249249;
  return x;
}
```

This dilate primitive is combined with bit-shift and `OR` operators to get the particles' key. In our implementation, we used 60-bit keys, which is sufficient for an octree with the maximal depth of 20 levels. We store a 60-bit key in two 32-bit integers, each containing 30-bits of the key. The maximal depth imposes a limit on the method, but so far we have never run into problems with our simulations. This limitation can easily be lifted by either going to 90-bit keys for 30 levels or by modifying the tree-construction algorithm when we reach deepest levels. This is something we are currently investigating.

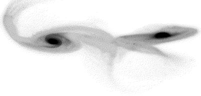




5 The Effect of Many Minor Mergers on the Size Growth of Compact Quiescent Galaxies

Massive galaxies with a half-mass radius $\lesssim 1$ kpc are observed in the early universe ($z \gtrsim 2$), but not in the local universe. In the local universe similar-mass (within a factor of two) galaxies tend to be a factor of 4 to 5 larger. Dry minor mergers are known to drive the evolution of the size of a galaxy without much increasing the mass, but it is unclear if the growth in size is sufficient to explain the observations. We test the hypothesis that galaxies grow through dry minor mergers by simulating merging galaxies with mass ratios of $q =$1:1 (equal mass) to $q =$1:160. In our N-body simulations the total mass of the parent galaxy doubles. We confirm that major mergers do not cause a sufficient growth in size. The observation can be explained with mergers with a mass ratio of $q =$1:5–1:10. Smaller mass ratios cause a more dramatic growth in size, up to a factor of $\sim$ 17 for mergers with a mass ratio of 1:80. For relatively massive minor mergers $q \gtrsim$ 1:20 the mass of the incoming child galaxies tend to settle in the halo of the parent galaxy. This is caused by the tidal stripping of the child galaxies by the time they enter the central portion of the parent. When the accretion of minor galaxies becomes more continuous, when $q \lesssim$ 1:40, the foreign mass tends to concentrate more in the central region of the parent galaxy. We speculate that this is caused by dynamic interactions between the child galaxies inside the merger remnant and the longer merging times when the difference in mass is larger. These interactions cause dynamical heating which results in accretion of mass inside the galaxy core and a reduction of the parent's circular velocity and density.

Jeroen Bédorf and Simon Portegies Zwart
Monthly Notices of the Royal Astronomical Society, Volume 431, Issue 1, p.767-780, May 2013.



5.1 Introduction

Hierarchical structure formation drives the growth of mass and size of galaxies with redshift (Peebles 1980; Lacey and Cole 1993). In this picture the mass and size of galaxies grow at a similar rate. Recently a population of small but relatively massive elliptical galaxies has been observed at $z \gtrsim 2$ (Daddi et al. 2005; Trujillo et al. 2006; Toft et al. 2007; van Dokkum et al. 2008; Franx et al. 2008; van de Sande et al. 2011; Szomoru et al. 2012). However at low redshift these galaxies are much more rare (Trujillo et al. 2009; van Dokkum et al. 2009; van der Wel et al. 2008; Taylor et al. 2010; Valentinuzzi et al. 2010; Trujillo et al. 2012), whereas there is a rich population of elliptical galaxies with similar mass but which are considerably larger in size (Martinez-Manso et al. 2011). This suggests that small and massive galaxies grow in size without acquiring much mass, which cannot be explained from major mergers as observed in hierarchical structure formation simulations (e.g. Naab et al. (2006); Oser et al. (2012) and references therein).

It has been suggested that dry minor mergers can cause a considerable growth in size without much increasing the mass of the galaxy, whereas major mergers tend to increase the mass without much increasing the size (Miller and Smith 1980; Naab et al. 2006; Oser et al. 2010, 2012; Trujillo et al. 2011).

Several studies on the topic seem to contradict each other, some argue that minor mergers can drive the observed growth e.g. (Bezanson et al. 2009; Naab et al. 2009; Hopkins et al. 2009, 2010). Others claim that the observed size growth in simulations is not sufficient if one takes into account cosmological scaling relations (Nipoti et al. 2009a, 2012; Cimatti et al. 2012). Encouraged by this discrepancy in the literature we decided to perform a series of simulations in which we model the encounters between compact massive galaxies and smaller, lower-mass counterparts. In our parameter search we included mass ratios from 1:1 all the way to 1:160. Even though such low-mass encounters are unlikely to be common (Oser et al. 2012), they approach a continuous in-fall of material on the parent galaxy.

With the merger simulations we study the growth in mass, size and the effect on the shape of the merger product. Our simulations are performed using the `Bonsai` tree code (Bédorf et al. 2012), (and Chapter 4), with up to 17.2 million equal-mass particles. This includes the dark-matter as well as the baryonic matter. In our simulations we recognize two regimes of galaxy growth. For $q \gtrsim$ 1:20 we confirm the inside-out growth, as discussed by (Hilz et al. 2013), whereas for $q \gtrsim$ 1:40 galaxies tend to form outside-in, meaning that the mass is accreted in the core of the primary galaxy rather than accreted on the outside. This change in behavior is caused by the self-interactions of minor galaxies for mass ratios $\lesssim 1 : 40$.

Recently (Hilz et al. 2012) studied the size increase from major mergers down to 1:10 minor mergers. Their results are consistent with our findings for comparable simulation parameters. The major difference between (Hilz et al. 2012) and the results presented here are the details regarding the orbital parameters of the merging galaxies; they drop their minor galaxies in one-by-one, whereas we initialize all galaxies in a spherical distribution at the start of the simulation. As a consequence in some of our simulations the merging process with many minor mergers is not completed by $T = 10\,\mathrm{Gyr}$. While the results

of our 1:1 to 1:10 mergers are consistent with the growth observed in the simulations of Hilz et al. (2012, 2013) and Oogi and Habe (2013), they are inconsistent with Nipoti et al. (2009b). The reason for this discrepancy cannot be the steeper density slope of the stellar bulge, as was suggested by Hilz et al. (2012), because we tried even higher density slopes for the minor child galaxies and found that this does not affect the growth of the merger product (see Appendix 5.B). The discrepancy between our results and those of Nipoti et al. (2009b,a) could be explained by the averaging of the simulation results, as was also suggested by Hilz et al. (2012).

5.2 Constraining the model parameters

To perform the simulations we use an updated version of the gravitational N-body tree-code `Bonsai` (Bédorf et al. 2012), Chapter 4. All simulations are performed on workstations with NVIDIA GTX480 graphical processing units. Simulations are run with 1 to 18 GPUs, depending on N.

The code has three important parameters; the choice of the time step dt, the softening length ϵ and the tree-code opening angle θ. The opening angle is set to $\theta = 0.5$ whereby we use the minimum distance opening criterion (Salmon and Warren 1994), which is computationally more expensive than the improved Barnes-Hut method (Dubinski 1996) by about a factor of two, but offers better accuracy (Bédorf et al. 2012).

After having fixed θ, the choice of dt and ϵ are quite critical for the quality of our results. We measure the quality of the simulation based on the error in the energy and the change in size of a galaxy model in isolation. We determine the optimal values for dt and ϵ by performing an analysis in which we run a series of models using a range of values for dt and ϵ. Although the time step and softening are discussed extensively in previous literature (e.g. (Merritt 1996; Athanassoula et al. 2000; Dehnen 2001)) we chose to do extra tests our-self. This for a couple of reasons, first most of the previous discussed methods propose a softening that scales with the number of particles. However, we use different number of particles for the primary galaxy and child galaxies so we had to find a configuration that works for both. Second, our tree-code uses an accurate multipole expansion, in combination with a opening angle criteria that is different from used in the previously mentioned papers. And finally we use two component models with power law density distributions while most examples in the previous mentioned papers are based on one component spherical Plummer models. Therefore we decided to rather empirically test how the softening and time-step affected the properties of our initial conditions instead of taking over a previously result obtained in a different setting. The initial conditions for these isolated galaxies are generated using GalactICS (Kuijken and Dubinski 1995; Widrow and Dubinski 2005; Widrow et al. 2008) and consist of a dark matter halo and a baryonic bulge. All models represent elliptical E0 galaxies. The bulges are modeled with a Sersic index of 3 and the dark-matter halo is represented by an NFW profile (Navarro et al. 1996).

In our simulations we collide the primary galaxy with a number of child galaxies, which are smaller and less massive. The mass ratio and the number of child galaxies which interact with the primary is one of the free parameters in our simulations. We generate the child galaxies using GalactICS by adopting the appropriate mass and size of the dark-matter

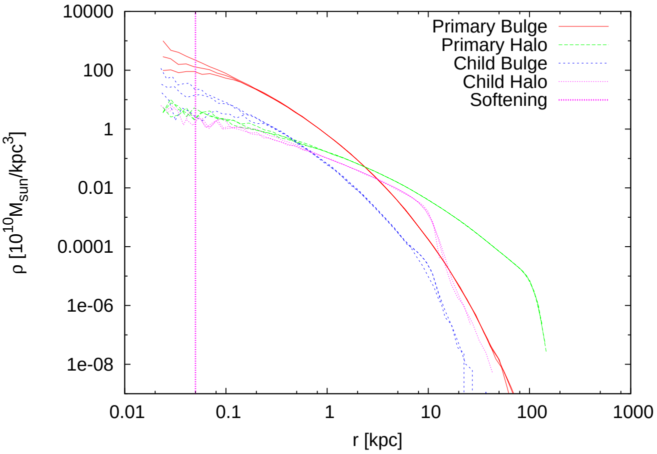
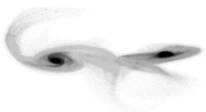


Figure 5.1: Density profiles of the HR baryonic bulge and dark-matter halos for the primary and child galaxies, which have evolved in isolation. For each component we present the density profile at zero age, at an age of 1 Gyr and at 10 Gyr. However, because the lines are almost indistinguishable and to prevent clutter we show each epoch with the same line style. The expansion in time of the child halo is caused by the escape of a few dark-matter particles. This is also the main reason for the expansion of the selected model, noticeable in the right hand-panel of Fig. 5.3. The vertical line near $R = 0.05$ kpc indicates the softening length in the HR simulation.

halo both of which we assume to be a constant factor smaller than in the primary galaxy. This factor is equal to the mass ratio of the child, e.g. for a 1:10 child the mass and cut-off radius is 10 times lower than that of the primary galaxy. As a consequence the density of the child galaxy is smaller by a factor $\sim$10 than that of the primary. In Appendix 5.B we demonstrate that the effect of our selected size and mass (and consequently density) of the child galaxy is not critical for our conclusions. In Fig. 5.1 we present the density profiles of the baryonic bulge and dark matter halo of our selected initial conditions at zero-age, at $T = 1$ Gyr which is the moment we intend to use them for the merger simulations in § 5.3, and at $T = 10$ Gyr. It takes a few crossing times for the galaxies to relax after generation, therefore we first let them evolve for $T = 1$ Gyr before we use them in a merger configuration. The differences in the density profile for the same component are hardly noticeable as a function of time. To prevent spurious mass segregation all particles in one simulation have the same mass. Mass differences between galaxies and baryonic and dark matter particle distributions are created by using different amounts of particles for each galaxy and galaxy components (see Tab. 5.1 for details).

The isolated galaxy simulations were performed for a galaxy with a total mass $M_{\rm parent} = 2.2 \times 10^{12}\ {\rm M}_{\odot}$, and a half-mass radius of 1.36 kpc. which we call the parent or primary galaxy and for each of the child galaxies. The child galaxies have a mass that is q times lower than that of the primary. For example the q =1:10 child has a mass of $M_{\rm child} = 2.2 \times 10^{11}\ {\rm M}_{\odot}$. To make sure that the child galaxies consist of enough particles to be accurately resolved and stay stable in isolation we use different resolution models for the initial conditions. Depending on the number of child galaxies (mass ratio) used. We



found that using a minimum number of 10^3 (10^4) particles for the bulge (dark matter) component proofed to be enough to keep the galaxies stable (Tab. 5.1).

We refer to low-resolution simulations (LR) for those in which the primary galaxy was modeled with $N = 2.2 \times 10^5$ and the 1:10 child with $N = 2.2 \times 10^4$. For the high-resolution (HR) simulations we adopted $N = 2.2 \times 10^6$ and $N = 2.2 \times 10^5$ for the primary and 1:10 child galaxies, respectively. The child galaxies with a different mass ratio are scaled in a similar way as the 1:10 child, namely the number of particles of the primary divided by the mass ratio. Each model was run for 10 Gyr. In Figs. 5.2 and 5.3 we present the results of the HR simulations; the error in the energy $\Delta E = (E_{10Gyr} - E_0)/E_0$ for the left-hand panels and the expansion factor of the bulge, γ(t), in isolation in the right-hand panels.

$$\gamma(t) \equiv R_{h-t}/R_{h-0Gyr} \tag{5.1}$$

Here R_{h-t} is the half-mass radius of the baryonic component of the galaxy at time t. In Fig. 5.2 we present the result for the primaries, and Fig. 5.3 for the 1:10 children.

As expected, ΔE decreases for decreasing dt and increasing ϵ, for the primary as well as for the child galaxy. The growth in size of the galaxies becomes smaller with decreasing dt. However, for ϵ this is not a trivial matter. For too large a softening the galaxy tends to expand substantially, but the effect is not as dramatic as for a too small value of ϵ. The growth for large ϵ is caused by spurious suppression of relaxation in the large ϵ runs. The growth for small ϵ is numerical, caused by strong encounters which are not resolved accurately in our simulation. This limitation in our numerical solver is also noticeable in the energy error for small ϵ (see Figs. 5.2 and Fig. 5.3).

Because we intend to study the expansion of the primary galaxies caused by mergers they should not expand in isolation. For our production simulations (see § 5.3) we select the ϵ which show the least expansion when the galaxy is run in isolation. The time step is chosen as large as possible (for performance) but giving the smallest possible energy error. We make the choice of dt and ϵ based on the simulations for the primary galaxy, which for the child also turns out to be the bests choice. For the HR simulations $dt = 1/192$ and $\epsilon = 0.05$, and for the LR simulations $dt = 1/128$ and $\epsilon = 0.1$. Using these settings the energy error of all the merger simulations stays below $\ll 10^{-4}$. These parameters are presented in dimension-less N-body units (Heggie and Mathieu 1986), but in physical units they translate to a time step of $dt \simeq 0.078$ Myr for the LR and $dt \simeq 0.052$ Myr for the HR simulations, and a softening length of $\epsilon \simeq 100$ pc, and $\epsilon \simeq 50$ pc for the LR and HR runs, respectively.

Comparing the most optimal settings with the results of previous work on softening values (Merritt 1996; Athanassoula et al. 2000; Dehnen 2001). The chosen softening values are comparable to the values advised in (Athanassoula et al. 2000) and (Dehnen 2001) with our chosen values being slightly larger to keep the energy error less than 10^{-3} while the size growth stays smaller than 10%. The difference between our results and Merritt (1996) is somewhat larger, but still within a factor of three. The difference can be attributed to the fact that we use more particles and use a tree-code instead of direct summation (see also Table 1. of Athanassoula et al. (1998)).

Most of the expansion of our galaxies occurs in the first few 100 Myr of the simulation, and therefore all galaxies are evolved to an age of 1 Gyr in isolation before they are used as

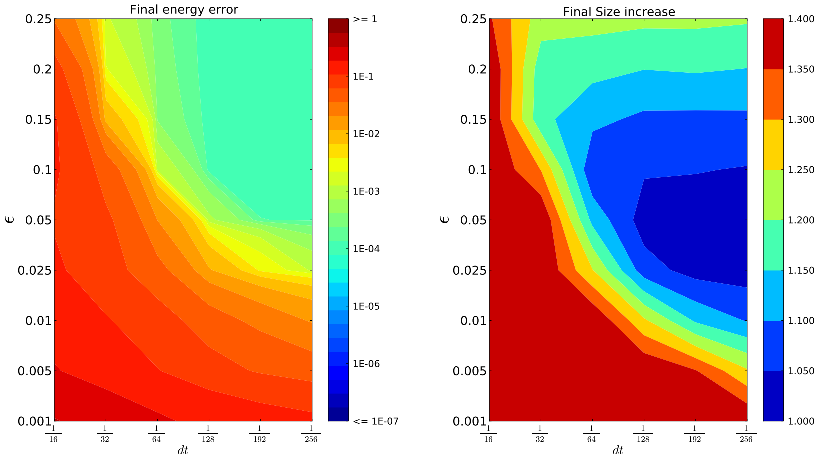


Figure 5.2: Results of the test simulations for the HR primary galaxies. The left panel gives the energy error ΔE, the right panel gives the size increase γ after 10 Gyr (see Eq. 5.2). Both panels present the information as a function of the time step (dt) and the softening length (ϵ). The simulations are performed at the grid-points in dt and ϵ, as indicated along the axes. The color scheme shows a linear interpolation between the grid points. The scaling to the colors are provided along the right-hand side of each panel, and exhibits a log-scale for the left hand panel and a linear scaling for the right-hand panel.

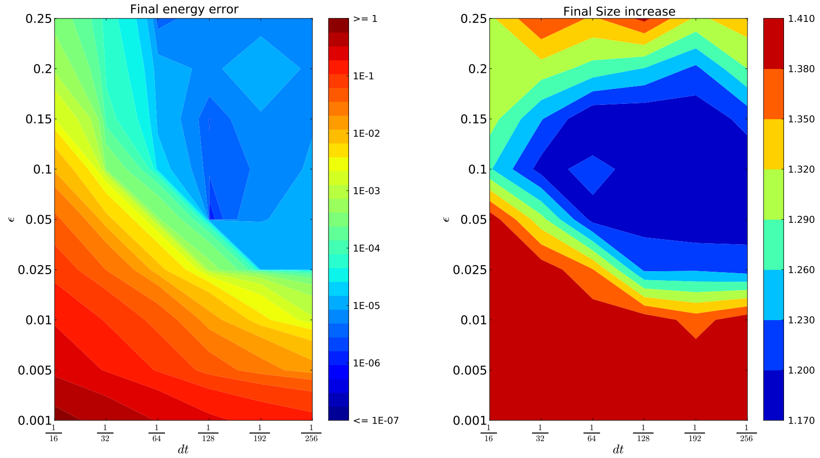
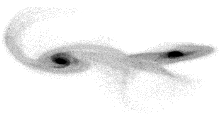


Figure 5.3: Results of the test simulations for the HR child galaxies, which are ten times less massive than the primary. See Fig. 5.2 for a further description of the figure.



			LR			HR		
Model	Type	Total mass [$M_\odot$]	N -	P mass [$M_\odot$]	ϵ [pc]	N -	P mass [$M_\odot$]	ϵ [pc]
Primary	Halo	2×10^{12}	1×10^5	1×10^7	100	2×10^6	1×10^6	50
	Bulge	2×10^{11}	1×10^4	1×10^7	100	2×10^5	1×10^6	50
Child $\frac{1}{5}$	Halo	4×10^{11}	4×10^4	1×10^7	100	4×10^5	1×10^6	50
	Bulge	4×10^{10}	4×10^3	1×10^7	100	4×10^4	1×10^6	50
Child $\frac{1}{10}$	Halo	2×10^{11}	2×10^4	1×10^7	100	2×10^5	1×10^6	50
	Bulge	2×10^{10}	2×10^3	1×10^7	100	2×10^4	1×10^6	50
Child $\frac{1}{20}$	Halo	1×10^{11}	-	-	-	1×10^5	1×10^6	50
	Bulge	1×10^{10}	-	-	-	1×10^4	1×10^6	50
Child $\frac{1}{40}$	Halo	5×10^{10}	-	-	-	5×10^4	1×10^6	50
	Bulge	5×10^9	-	-	-	5×10^3	1×10^6	50
Child $\frac{1}{80}$	Halo	2.5×10^{10}	-	-	-	2.5×10^4	1×10^6	50
	Bulge	2.5×10^9	-	-	-	2.5×10^3	1×10^6	50

Table 5.1: Galaxy properties. Characteristics of the base models used in the simulations. The first column indicates if the galaxy is the primary galaxy or one of the minor merger child galaxies. The second column indicates if the properties are for the dark matter halo or for the baryonic bulge and directly after in the third column the mass of the galaxy component. The next three columns show the properties when using low resolution models, the number of particles, mass per particle and the softening used. The final three columns show the number of particles, mass per particle and softening used when using high resolution models.

initial galaxies in the merger simulations.

As as extra test we ran several of the isolated galaxies using the direct N-body code `phiGRAPE` (Harfst et al. 2007) to confirm that the observed ΔE and expansion are not the result of using an approximation based simulation code. The results of these simulations are consistent with the results of `Bonsai`, but show a ΔE of $\sim 10^{-6}$, because of the higher accuracy of `phiGRAPE` .

With this choice of initial conditions, the child galaxies in the low resolution simulations still expand by at most a factor of 2 before the merger starts. To check how this affects our results we redo the simulation but at the start the child galaxies are represented as point masses. The point mass is replaced by a child galaxy as soon as it enters the dark matter halo of the primary galaxy. This simulation gives a nearly identical expansion of the primary galaxy to that in which the child was resolved from the start of the simulation.

5.3 Initializing the galaxy mergers

We use the isolated galaxies discussed in the previous § to study the effect of mergers. For this purpose we adopt the primary galaxy and have it interact with an identical copy or with a number of child galaxies. We refer to minor mergers when the primary:child mass $\leqslant$ 1:5. Major mergers in our study have equal mass.

5.3.1 Configuring the major mergers

In the major merger simulations two identical copies of the primary galaxy are placed on an elliptical orbit. The initial distance between the galaxies is 400 kpc, which exceeds the size of the dark-matter halos, which is about 200 kpc. The relative velocity is chosen such

that the minimal distance of approach during the first perigalactic passage P varies. We vary P from head-on (0 kpc) to 10, 50, 75 and 100 kpc for the widest encounters. All our merger simulations are run until 10 Gyr, after which both galaxies have merged to a single galaxy, except for the $P = 100$ kpc configurations (see § 5.4).

Because we use two identical copies of the primary galaxy the galaxies will be co-rotating during the collision. To study the influence of the rotation angle on the size of the merger product we run the simulations for $P = 0$ and $P = 50$ kpc with co-rotating and counter-rotating galaxies. This is done by changing the inclination and Ω of one of the two galaxies between $0\,^\circ$ and $270\,^\circ$ in steps of $90\,^\circ$ using a rotation matrix. After the rotation the galaxies are put on their elliptical orbit as described in the previous §. The stability of the isolated galaxies is not affected, because the positions and velocities are rotated in a consistent way. We did not perform this study on the other major mergers, because the effect of the inclination and Ω on the expansion factor turns out to be negligible (variation of the final size between the 16 configurations is ~ 0.02 kpc).

5.3.2 Configuring the minor mergers

The easiest way to generate an initial configuration with 1 primary galaxy and N child galaxies is by randomly positioning them in a spherical distribution. For each simulation we generate a Plummer distribution (Plummer 1915) of $N + 1$ particles. We then select a random particle and reposition it in the center of mass. We assign the mass of the parent galaxy to this central particle. The other particles are assigned the mass of the child galaxies. The total mass of all child galaxies in each simulation is the same as that of the primary galaxy. The model is then rescaled to a virial radius, R, and virial ratio (temperature), Q. We run 10 sets of initial conditions for each selected mass ratio. In the next section we discuss the results of a number of simulations with varying mass-ratio, R and Q. Using this procedure we created a range of minor merger models with varying R, $Q = 0$ and N, see Tab. 5.2. In Section 5.3.2 we relax this assumption and allow $Q > 0$ to study the effect of the temperature on the size of the merger product.

For the 1:10 minor mergers we adopted $R = 200$ kpc as well as $R = 500$ kpc. In the $R = 200$ configurations the child galaxies have a $\sim 50\%$ probability of being placed initially within the dark matter halo of the primary. For the $R = 500$ configurations, which are based on the same random Plummer realization but scaled to a larger virial radius, this is only $\sim 15\%$.

Plummer models - Varying virial temperatures

In order to study the effect of Q on our results we selected one of the 1:10 models. The merger remnant of the selected model has roughly the same mass in both the $R = 200$ and $R = 500$ configurations indicating that they were involved in an equal number of mergers by the end of the simulation. We used this initial configuration to vary Q between 0 (cold) and 1 (warm), see Tab. 5.2. Each run was repeated with $R = 200$ kpc and for $R = 500$ kpc. This results in a total of 12 different configurations, all of which are based on the same spatial distribution but with different virial radii and virial temperatures.

---

[1] based on same Plummer distribution as the $R = 200$ models.

mass ratio	R	Q	resolution	name
1:5	500	0	LR	1:5
1:10	200	0	LR	1:10, R200
1:10[1]	500	0	LR	1:10, R500
1:20	500	0	HR	1:20
1:40	500	0	HR	1:40
1:80	500	0	HR	1:80
1:10	200	0.0	LR	
1:10	200	0.05	LR	
1:10	200	0.1	LR	
1:10	200	0.3	LR	
1:10	200	0.5	LR	
1:10	200	1.0	LR	
1:10	500	0.0	LR	
1:10	500	0.05	LR	
1:10	500	0.1	LR	
1:10	500	0.3	LR	
1:10	500	0.5	LR	
1:10	500	1.0	LR	

Table 5.2: Initial conditions for the minor merger simulations. The first column indicates the used mass ratio. The second and third column indicate the virial radius, R, and virial temperature, Q, to which the configuration is scaled. The fourth column indicates the used resolution, either low-resolution ($N = 4.4 \times 10^5$) or high-resolution ($N = 4.4 \times 10^6$). The final column indicates the name used in the text when we refer to the configuration.

5.4 Results

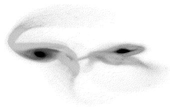
During the simulation snapshots are taken every 100M year and post-processed using `tipsy` [2]. We compute the density and cumulative mass profiles which are subsequently used to determine the half-mass radius (R_h), measured using the baryonic particles only. To compute the profiles `tipsy` computes the densest point in the simulation and then uses spherical shells centered on the densest point to bin the particles. To compute R_h we can not simply take the radius that contains half of the total mass in the merger system, because depending on the time of the snapshot, not all child galaxies have merged with the primary galaxy. Therefore we analyze the cumulative mass profile (based on all baryonic particles in the system) of the galaxy and take as total mass the point where the profile flattens. This flattening indicates that all child galaxies within that radius have merged[3] with the primary galaxy and that the unmerged child galaxies are too far away to be counted as part of the merger remnant. This total mass is then used to determine R_h. We illustrate this procedure in Fig. 5.4.

In Fig. 5.4 we present the cumulative mass-profiles of one of the 1:10 mergers with $R = 500$ and $Q = 0$, for the major merger with $P = 50\,\mathrm{kpc}$ and the profile of the isolated primary galaxy. During the first four Gyr of evolution the cumulative mass of the minor merger simulation gradually increases each time one of the child galaxies is accreted. We determine the total mass of the merger remnant by determining the first moment that the cumulative mass profile becomes flat. This happens, for example, for the 1 Gyr situation at a mass of $M \sim 2.2 \times 10^{11}$ M$_\odot$, and for 2 Gyr at $M \sim 3.2 \times 10^{11}$ M$_\odot$.

---

[2] http://www-hpcc.astro.washington.edu/tools/tipsy/tipsy.html

[3] We chose this method over a method where we check if particles are bound, since in this method we include child galaxies that would add to the luminosity of the remnant galaxy if they would be observed.



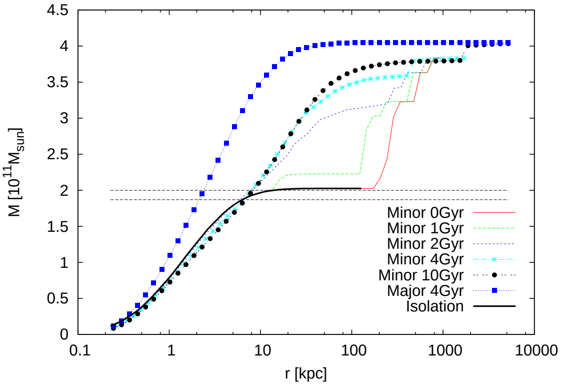
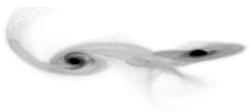


Figure 5.4: Cumulative mass profiles of the bulge. Shown are the profiles of one of the 1:10 merger simulations with $R = 500$ and $Q = 0$ and the major merger simulation with $P = 50$ kpc. The thick solid line shows the cumulative mass profile of the main galaxy at the start of the simulation. The horizontal lines show the half-mass of the major merger (top line at $2.0\ \mathrm{M}_{\odot}{}^{11}$) and of the minor merger (bottom line at $1.86\ \mathrm{M}_{\odot}{}^{11}$). The remaining lines show cumulative mass profiles of the minor merger as indicated.

We further notice that the minor merger remnant has grown in size with respect to the major merger remnant, even though it has accreted less mass. The major merger remnant ends up with twice the mass of the primary galaxy, e.g. a complete merger of all mass in the system. Whereas in the minor merger only 8 child galaxies have been accreted in the 4 Gyr after the start of the simulation. At the end of the simulation (T=10 Gyr) 9 child galaxies have merged with the primary galaxy. The last unmerged child appears in the cumulative mass distribution (striped line with solid circles), represented by the bump in the line at $r \sim 1800$ kpc (Fig. 5.4). As a consequence the total mass of the merger product at the end of the simulation is larger for the major merger than for the minor merger.

5.4.1 The growth of the primary due to subsequent mergers

We run each major merger model 16 times (with different inclination and Ω) and average the R_h. For each set of minor merger initial conditions (R and N, see Tab. 5.2.) we perform 10 simulations and as in the major mergers we average R_h. These results are presented in Fig. 5.5 for the growth in size, and Fig. 5.6 for the growth in mass. The shaded areas are the 1σ deviation from the mean, but for the major mergers the areas are barely visible in comparison with the width of the line. This indicates that the inclination and the angle Ω hardly affects the growth of the merger product. Only during the merger event itself some deviation is noticeable (at about 3 Gyr for the simulation with $P = 50$ kpc). The head-on configuration ($P = 0$) grows from an initial size of $R_h = 1.3$ kpc to 3.4 kpc, whereas the $P = 10$ to 75 kpc simulations grow to only about 2.4 kpc (only the $P = 50$ kpc is presented in Fig. 5.5). The $P = 100$ kpc did not experience a merger in our simulation within the time frame of 10 Gyr. For the minor merger configurations the result is quite different as can be seen in Fig. 5.5. The shaded areas indicate that there is variation in size between the different configurations. The size growth however is larger than that of the major mergers. The results indicate that the growth in size is directly related to the mass ratio of the child galaxies, the larger the difference in mass the larger the growth in size. For the 1:10 simulations the R_h is at least 7 kpc and for the 1:80 simulations the average R_h is at least 20 kpc while at the start of the simulation the primary galaxy has a R_h of 1.3 kpc.



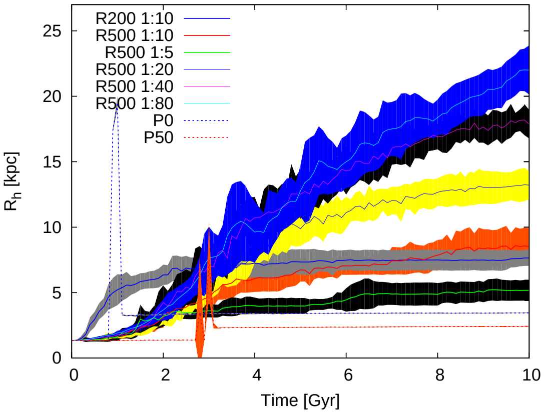


Figure 5.5: Evolution of R_h. The plot shows the mean (solid and dotted lines) and the standard deviation (shaded areas) of the R_h based on different merger configurations. The solid lines show the results of the minor merger configurations. The dotted lines show the major merger configurations with $P = 0$ and $P = 50$. The minor mergers are each averaged over 10 different Plummer realizations, the major mergers are averaged over 16 different combinations of Euler angles. For 1:80 we filtered out size jumps caused by invalid size detection.

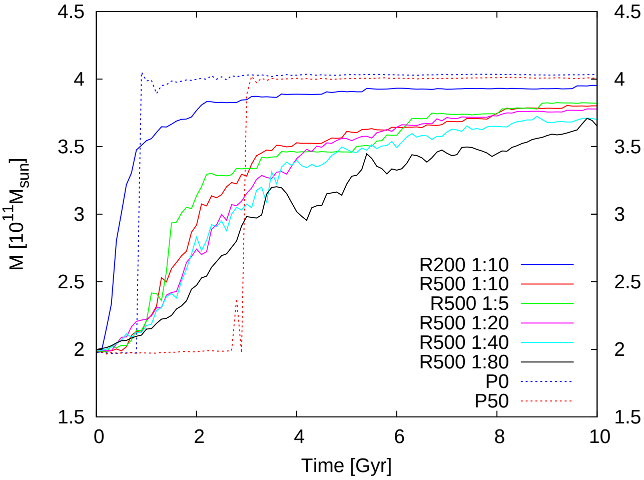


Figure 5.6: Evolution of the merger products mass. The same simulations are shown as in Fig. 5.5, but instead of the radius the average mass of each set of simulations is shown.

To study the effect of increasing the number of minor mergers (and decreasing the mass per child galaxy) in more detail we start by comparing the major merger with the 1:5 minor merger. This case results in a R_h of 5.1 kpc, averaged over 10 simulations. The 1σ deviation indicates that the result is rather consistent when rerunning with a different initial Plummer realization. The extreme values of R_h after 10 Gyr range between 4 and 6 kpc. Even the increase due to the 1:5 minor merger exceeds the $P = 0$ major merger case by a factor of ~ 1.2 to 1.8. By the end of the simulation at 10 Gyr on average 4 out of the 5 minor galaxies have merged with the primary galaxy, and we expect that if the last child galaxy would be accreted the increase in size would be even larger. The 1:10, R200 kpc minor mergers show an average size of 7.6 kpc, and 8.5 kpc for the 1:10, R500 kpc models; an increase of 5.8 to 6.5. The $R = 200$ kpc simulations merge on a shorter time scale compared to the $R = 500$ kpc, which results in a smaller expansion, but higher mass of the simulations performed in a smaller volume. This is also visible in Fig. 5.6 where the $R = 200$ kpc increases faster in mass and ends up being more massive than the $R =$

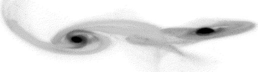
500 kpc case. Even though $R = 500\,\text{kpc}$ results in a less massive merger remnant than the $R = 200\,\text{kpc}$ simulations, it grows to a larger size. This is caused by the disturbing effect that the dark-matter halo of the primary galaxy has on the child galaxies. With more galaxies placed outside the halo for the $R = 500\,\text{kpc}$ case this effect will be more pronounced. While we continue to increase the number of child galaxies the size of the merger remnant continues to grow. We stopped with $N = 80$ children, at which moment the increase in size compared to the major merger exceeded a factor 17. The main reason for the growth of the merger remnant when increasing the number of child galaxies is by the heating of the merger remnant. For each doubling of the number of child galaxies from 1:5 to 1:10, etc., the growth in size of the merger product is about constant by $\sim$ 5 kpc. The growth in mass for each of these initial configurations is comparable, except for the 1:80, which grows less quickly. The smaller mass growth in the 1:80 is attributed to increased dynamical interactions among the child galaxies. These interactions cause the merging times per galaxy to take longer than when they would merge sequentially.

The combined growth in size and mass indicates an other effect, namely the larger the mass ratio the higher the growth in size. This is indicated in Fig. 5.7 where we present R_h as a function of the merger remnant mass (which is a function of time). This figure quantifies the growth in size of the merger remnant while increasing the number of mergers. Note that the mass ratio between the merger remnant and the child galaxies changes over time. For example the 1:10 minor mergers start with a mass difference of a factor 10 between one child galaxy and the primary galaxy, but after the primary galaxy has merged with 5 child galaxies the mass ratio between the merger remnant and one child galaxy is 1:15. Interestingly it is not necessarily the mass that drives the merger remnant size, but it is the number of children that merges with the primary galaxy. The solid curves are fits through the data, and represent exponential curves. The dashed line is the observed scaling relation of $R_h \propto M^{2.04}$ (van Dokkum et al. 2010). As we can see our minor mergers are above this relation while the major mergers are clearly smaller than the observational results. Indicating that the minor mergers can cause a size increase comparable or even larger than that observed.

5.4.2 The effect on the shape of the galaxies due to subsequent mergers

Now that we have established that the minor mergers can have a pronounced effect on the size of the galaxy remnant it will be interesting to see if we can recognize this galaxy growth by the shape of the merger remnant. To test the effect that the mergers have on the shape we measure the semi-principal axes a, b and c of the ellipsoidal with $a >= b >= c$. The axes lengths are obtained by computing the eigenvalues of the inertial tensor of the galaxy. Using this we plot the three axis ratios b/a, c/a and b/c, if these three ratios are 1 then the elliptical is perfectly spherical. If one of the axis is much smaller than the other two the galaxy is a flattened spherical.

In Figs. 5.8 and 5.9 we present how the shape of the merger remnant (measured at R_h) evolves over time (baryonic component only). For the major mergers we used the default configurations with $P = 0$ and 50 kpc and for the minor merger configurations we randomly took one of the 10 realisations for each of the used mass ratios. The x-axis shows



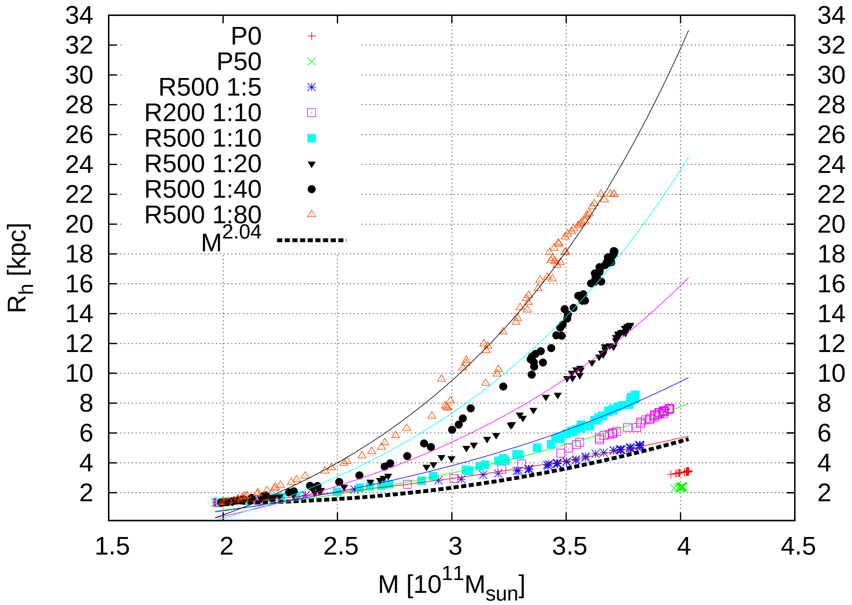


Figure 5.7: The mass of the merger remnant, M, versus R_h for the different configurations. The solid lines are exponential fits through the results of the minor merger configurations. The major mergers are not fitted. The thick dashed line (bottom line) is the observed scaling relation of $R_h \propto M^{2.04}$ (van Dokkum et al. 2010). Configurations are plotted after each merger event (but filtered to not show fly-by events) and therefore occur multiple times. The exponents of the fits are (from top to bottom), 0.9, 0.84, 0.74, 0.62, 0.57 and 0.48. If we make an exponential fit to the observed scaling relation than this would have an exponent of 0.5.

the simulation time while each of the three panels presents one of the aforementioned axis ratios. Visible is that all galaxies start of as a perfect sphere, which is the initial condition, but over time the merger remnants deform. The major mergers show the highest degree of deformation with large differences between the different axis. The effect is most severe for the heads-on collision which is expected as it is the most violent collision of all our configurations. Although the major merger remnants become slightly more spherical after the mergers are complete the galaxies will not become perfectly spherical within a Hubble time.

For 1:5 we notice, as with the the major mergers, a clear deformation from spherical, but the effect is less strong than for the major mergers. The individual merger events of 1:5 are visible by the spikes in the axis ratios. The impact that a merger event has on the shape of the merger remnant becomes smaller when the mass difference between merger remnant and the infalling galaxy increases. For the 1:10 configurations, we do see slight deformations during the merger events (the shark tooth's in the lines), but when the merger is complete the remnant settles back to spherical. The same effect is visible for 1:20, 1:40 and 1:80 (Fig. 5.9). The final merger remnant of the minor merger configurations is no longer perfectly spherical, but it clearly is not as flat as the major merger configurations.

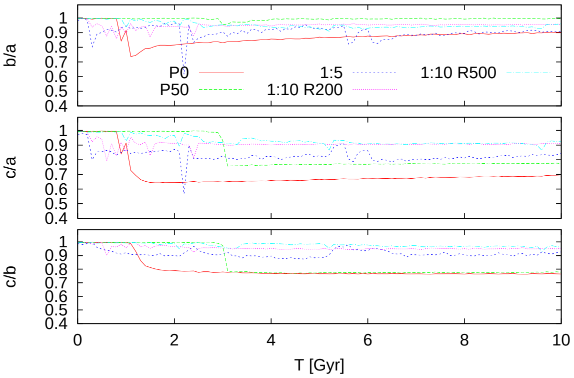


Figure 5.8: Axis ratios of the merger remnant. The three panels present the axis ratios b/a, c/a and b/c of the merger remnant (top to bottom, with a $\geqslant$ b $\geqslant$ c). As location to measure the shape is taken the mean R_h of the configuration. If all axis ratios are equal to 1 the shape is perfectly spherical. Shown are the results for the head-on major merger ($P = 0$ kpc), for the major merger with $P = 50$ kpc, for a random 1:5 merger and for a random 1:10 merger once with $R = 200$ kpc and once with $R = 500$ kpc.

Indicating that the accreted mass is distributed differently through out the merger remnant in the minor merger simulations than in the major merger simulations. We cover the location of the accreted mass in more detail in Section 5.5.

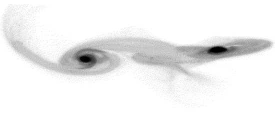
In Fig. 5.10 we present the galaxy shape at $T = 10$ Gyr from the core of the merger remnant up to $2 \times R_h$, while in the previous paragraph we looked at the shape of the merger over time and only at the R_h location. Because R_h is different for each configuration the lines representing the configurations in Fig. 5.10 have different lengths. The minor merger results are averaged over the different realisations of the model. We notice that the larger the differences in mass between the the primary galaxy and the child galaxies the more spherical the merger remnant stays (axis ratios near 1). The differences between the major and minor mergers are especially visible on the outsides of the merger remnant ($r > R_h$). The major mergers are much more deformed from perfectly spherical into flattened ellipticals while the minor mergers stay near spherical all the way to the outskirts of the galaxy. The difference in the core of 1:5 and 1:10 with respect to the other configurations is the result of running these simulations with only $N = 4.4 \times 10^5$ particles and accompanying larger softening. We checked this by comparing the individual results of one of the major mergers and a 1:10 high resolution simulation with the comparable configurations in low resolution. In the high resolution simulations we do not observe the deformation in the core, instead the axis-ratios have a similar profile as the other high resolution configurations.

5.4.3 The effect of the virial temperature

In the previous section the virial temperature (Q) of the minor merger configurations was always set to zero. This resulted in cold collapse and consequently the fastest possible merger. In this section we investigate the configurations described in Section 5.3.2 where Q is varied. The results are presented in Fig. 5.11 where we present R_h as a function of the merger remnant mass[4] (which is a function of time). In the top panel the results for the $R = 200$ kpc are presented and the bottom panel presents the results for the $R = 500$ kpc

---

[4] We filtered the results to not include fly-by events which temporarily increases R_h.



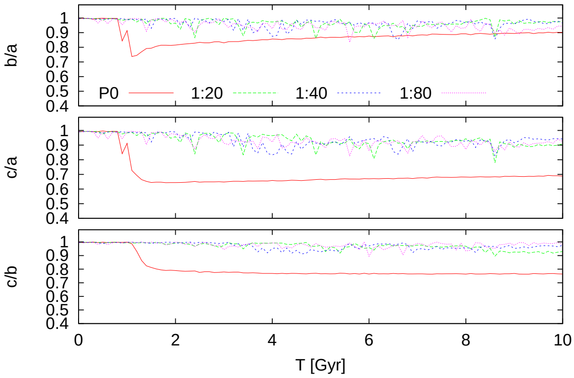


Figure 5.9: As Fig. 5.8, but now for one of the 1:20, 1:40 and 1:80 configurations. The head-on major merger from Fig. 5.8 is also presented (solid line).

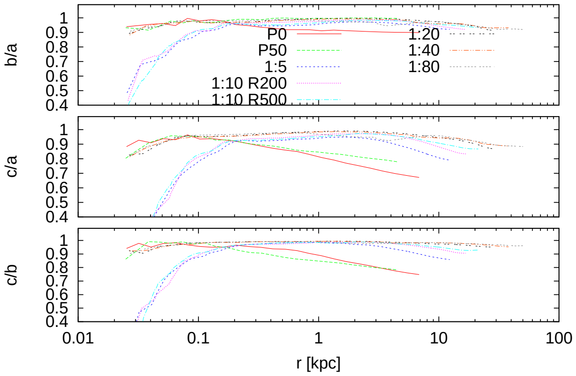


Figure 5.10: Final shape of merger remnant. On the x-axis we show the distance from the core of the merger remnant. The three panels present the axis ratios b/a, c/a and b/c (top to bottom, with a $\geqslant$ b $\geqslant$ c). As final measurement location we took $2 \times R_h$. The results for the minor mergers are averaged over all 10 different Plummer realisations. The differences in the cores of the 1:5 and 1:10 mergers is caused by the lower resolution with which these models have been run (see text for details).

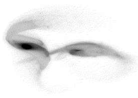
configurations. The results for the major mergers with $P = 0$, 10 and 50 kpc are presented in both panels.

The major mergers are the same as the ones presented in the previous section and end up with a final mass that is double that of the isolated galaxy ($M = \pm 4 \times 10^{11}\ \mathrm{M}_\odot$). Indicating that in all major merger configurations the two galaxies fully merge. The horizontal distribution in mass is caused by accretion of stars that were flung out and later fell back onto the merger product, thereby increasing the mass of the merger remnant.

The merger events that occur in the minor merger simulations can be seen in the horizontal distribution of the points. The horizontal distance between the points compares to a separation equal to the mass of a 1:10 child galaxy. Even though the minor merger configurations are based on the same spatial distribution, they are not per se involved in the same amount of mergers, this depends on R and Q. Comparing the cold ($Q \leqslant 0.1$) and warm ($Q > 0.1$) configurations we notice that at the end of the simulation the cold configurations have a higher mass than the warm case. In the warm configurations the child galaxies have a higher velocity which increases their merging times. However, both warm and cold show a size increase larger than that of the major merger simulations. This is confirmed by the lines that show exponential fits[5] through the configurations with $Q = 0$,

---

[5] Note that the fits through the $Q = 1.0$ results are based on only 4 (1) merger events in the $R = 200$ kpc



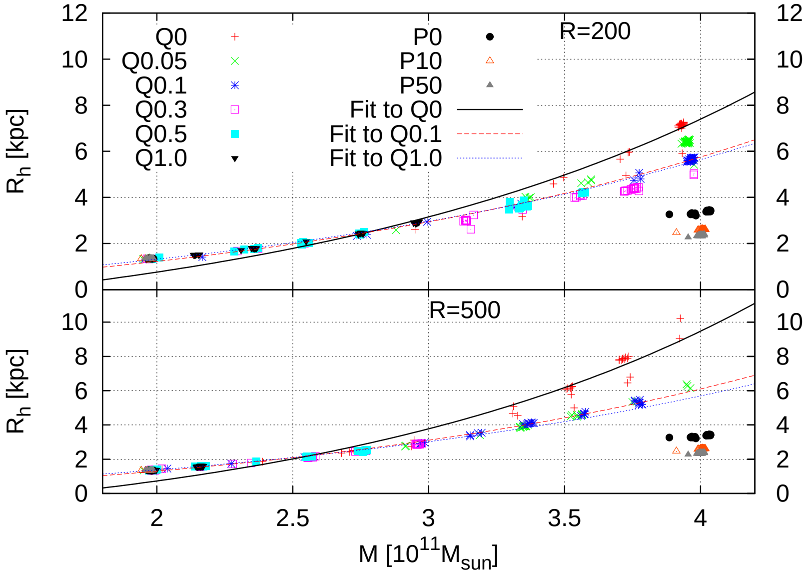
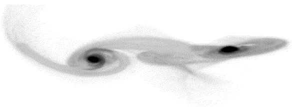


Figure 5.11: The mass, M, of the merger remnant versus the half-mass radius, R_h, for major merger configurations and a 1:10 merger configuration with different virial temperatures (Q). The minor mergers are all based on the same Plummer distribution, but rescaled to a virial radius of $R = 200$ ($R = 500$) in the top (bottom) panel and a virial temperature which is varied between 0 (cold) and 1 (warm). Both panels show the major mergers with $P = 0$, 10 and 50 kpc. The lines are exponential fits through the configurations with $Q = 0$ (solid), $Q = 0.1$ (striped) and $Q = 1.0$ (dotted). With exponents 0.57, 0.49 and 0.485 for $R = 200$ and 0.62, 0.49 and 0.484 for $R = 500$. Configurations are plotted after each merger event and therefore occur multiple times.

$Q = 0.1$ and $Q = 1.0$.

The fits indicate that if all child galaxies merge with the primary, the merger remnant would have a size that is at least a factor of two larger than the remnants formed by major merging independent of Q. For the $Q = 0$ mergers this is even three times as large as the major mergers. Also visible in Fig. 5.11 is the difference between the $R = 200$ kpc and $R = 500$ kpc configurations. Here we notice, as in Fig. 5.5, that the $R = 500$ kpc configurations show a larger size increase than the $R = 200$ kpc configurations, even though the amount of completed mergers in the $R = 500$ kpc is less than in the matching $R = 200$ kpc configuration. This is attributed by the higher energy input that the child galaxies give in the $R = 500$ kpc configuration compared to the $R = 200$ kpc situation where half the child galaxies are already within the dark matter halo of the primary galaxy at $T = 0$.

($R = 500$ kpc) configurations. The observed trend however is the same as for the $Q = 0.1$ configuration which is based on 10 (9) mergers.



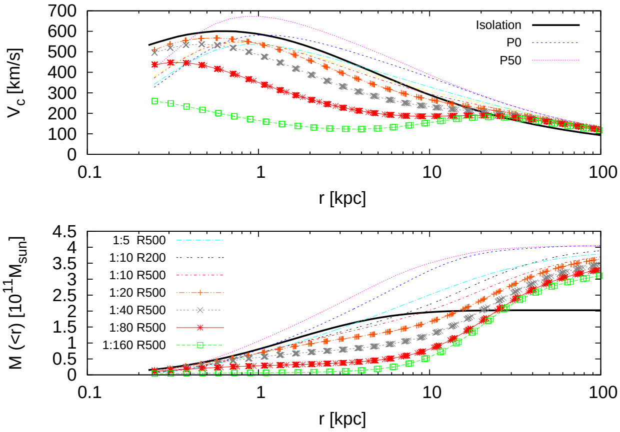


Figure 5.12: Circular velocity (top) and cumulative mass (bottom) profiles after 10 Gyr for different merger configurations. Each line shows the averaged result of all random realisations for each of the configurations. The velocity and mass profile of the isolated model are indicated with the thick solid line.

5.5 Discussion

The simulations indicate that the effects of the minor mergers becomes more pronounced when more mergers are involved in the mass growth. To test if this trend continues we performed three extra simulations with 160 child galaxies (mass ratio 1:160). With this many children the simulation approaches the regime of a continuous stream of infalling material. The three simulations gave very similar results with little variation between the runs. With this many child galaxies their distribution around the primary galaxy becomes almost uniform. The effect of the random placement of the child galaxies is therefore much smaller than in the simulations for 5 to 20 child galaxies. Furthermore some of the children will merge with other children before they reach the primary galaxy. For the 1:160 simulations we use $N = 8.8 \times 10^6$ particles evenly divided over the primary ($N = 4.4 \times 10^6$) and the child galaxies ($N = 27500$ per galaxy).

5.5.1 Properties of the merger remnant

The circular velocity ($V_c = \sqrt{M(< r)/r}$) and the cumulative mass profile at the end of the simulations (at $T = 10$ Gyr), both are presented in Fig. 5.12. The top panel shows V_c and the bottom panel presents the cumulative mass profile over the same distance from the core. For the mergers we averaged the results of the random realizations.

The major mergers cause an increase in the V_c compared to the isolated model. Whereby the remnants formed by major mergers with an elliptic orbit ($P = 50$ kpc) have a V_c which is ∼100 km/s higher than that of the isolated galaxy. For the remnants that formed by minor mergers we see a decrease in V_c compared to the isolated galaxy. Especially for the 1:40 and 1:80 the effect is quite pronounced, with the large size increase comes a decrease in velocity. This indicates that the lighter the child galaxies are the lower the V_c becomes, this trend continuous for the 1:160 models. These models have the lowest circular velocity of all the configurations while at the same time showing the largest size increase.

The size increase can be deduced from the cumulative mass profiles presented in the

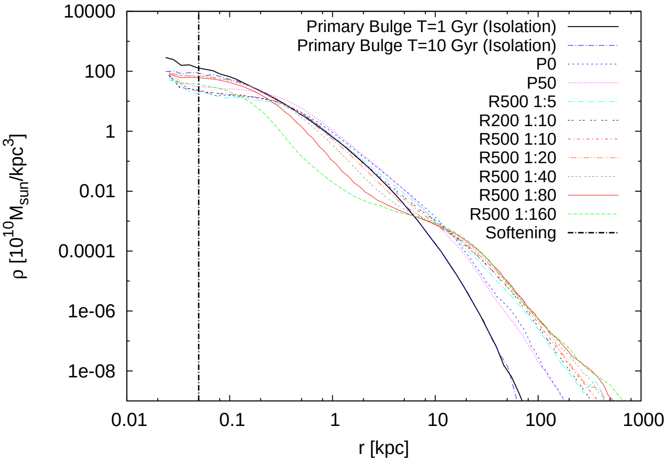
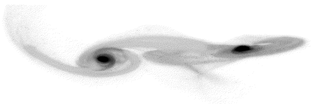


Figure 5.13: Density profiles of the different major and minor configurations at the end of the simulation ($T = 10\,\mathrm{Gyr}$). Also displayed is the initial density profile of the isolated galaxy. The vertical line is the softening used in the simulations.

bottom panel of Fig. 5.12. We notice that the mass in the core of the merger decreases when more child galaxies are involved in the merging process. This effect is especially pronounced for the smaller mass-ratios (<1:20). The smaller child galaxies have longer merging times, this combined with their interactions inside the core of the merger remnant does not allow the merger remnant to stabilize. Therefore the mass does not sink back to the core within the simulation period. This is also apparent in the density profiles presented in Fig. 5.13. Here the major mergers have a density comparable to that of the isolated galaxy which in turn is higher than the density of the minor mergers. For the 1:5 and 1:10 mergers the density profiles have a similar shape to that of the isolated galaxy and the major merger remnants, but the configurations with mass ratios of $\leqslant 1:20$ have a different shape. The relatively sudden depression in the density profiles of 1:80 and 1:160 near $r \sim 1\,\mathrm{kpc}$ is a trend that is visible in the 1:40 configuration and in slightly less pronounced form for the 1:20 configurations. The longer merging times and dynamic interactions that takes place when the number of child galaxies is $\geqslant 20$ causes a slowdown in circular velocity, a reduction in mass and therefore a drop in density in the inner parts of the merger remnant.

The effects described above influence where the mass of the child galaxies is accreted in the merger remnant. In Fig. 5.14 we present the fraction of baryonic mass that originates from the child galaxies as a function of the distance to the center of the merger remnant. The curves are constructed by binning the particles in 1 kpc bins and compute the fraction of mass that was brought in by the child galaxies. The curves representing the major mergers (solid-red for model P0 and dashed-green for P50) are horizontal and at 0.5 in Fig. 5.14, which indicates that they mix homogeneously. Minor mergers experience a different mixing; the precise effect depends sensitively on the number of infalling galaxies.

We take a closer look at the 1:5 to 1:20 mass ratios mergers. These configurations behave as expected Hilz et al. (2013); the fraction of child material is small in the core of the merger remnants and high in the outskirts ($r \gtrsim 10\,\mathrm{kpc}$). While the mass ratio continues to decrease the fraction of foreign material in the outskirts also decreases; until the majority of outskirt material is original for the most extreme mass ratio (1:160). We have to note here that for the most extreme mass ratio's the merger process has not completed at $T =$



10 Gyr, which is noticeable in Fig. 5.14 by the mean line appearing smaller than 0.5. In the central portion ($r \lesssim 7$ kpc) of the most extreme mass ratio mergers the opposite happens, in the sense that the contribution of foreign material is actually increasing (in particular noticeable for the 1:160 mass ratios).

In Fig. 5.15 we present the number density of the foreign and native particles as a function of the radius. We do this for a mass ratio of 1:1, 1:10 and 1:160. As we already discussed in relation to Fig. 5.14 the mass in the major merger remnant is evenly distributed between the primary and the infalling galaxy. In the 1:10 merger (top panel) the two curves for the foreign material and for the native material are clearly separated, in contrast to the lines for the major merger. This indicates that the native material is distributed differently than the foreign material. The same effect is noticeable in the 1:160 merger (lower panel in Fig. 5.15), but less pronounced. This indicates that in the 1:160 merger the material of the primary and child galaxies is mixed more evenly and approaches a mass distribution similar to that of the major mergers, whereas in the 1:10 merger the native and foreign material are distributed quite differently.

The results presented in the previous § indicate that for mass ratios 1:1 to 1:20 the accretion behaves according as if they grew from inside-out; the material of the child galaxies is stripped and deposited in the outside of the primary galaxy (Hilz et al. 2013). When the number of child galaxies increases beyond 20 this behavior changes in that the mixing becomes more homogeneous. This is caused by the self-interactions of the child galaxies with the nucleus of the primary. These self-interactions prevent the merger galaxy from settling to equilibrium and cause the core to heat up.

We theorize that the continuous bombardment of minor galaxies causes the merger remnant to remain quite dynamic which reduces the dynamical friction. This in itself allows new incomers to penetrate deeper into the parent galaxy. This effect mimics the process of violent relaxation (Lynden-Bell 1967). This effect becomes more efficient when the number of child galaxies increases. During the in-fall they interact with other infalling children before they break-up, after which their mass is distributed evenly in the merger remnant. The interactions are not strong enough to permanently eject a child. When the child galaxies are more massive 1:5 to 1:20 mergers self interactions between the children is negligible and they tend to be disrupted upon the first pericenter passage of the merger product. As a result their material tends to be deposited in the outskirts.

The net growth in size observed in our simulations exceeds those reported in (Nipoti et al. 2009a, 2012). The initial conditions of (Nipoti et al. 2009a, 2012), however, deviates from ours in the sense that they adopt a more realistic mass and shape distribution than in our more theoretical approach. As a consequence the results cannot be compared trivially. The more efficient growth in our simulations is a result in the distribution of the minor galaxies, and not by the choice of their densities; the effect of the child densities is negligible (see Appendix 5.B). Our results are consistent with those of Hilz et al. (2012); Oogi and Habe (2013) for mass ratios $< 1:20$, in the sense that the growth in size is sufficiently efficient to explain the observational results when the number of infalling galaxies > 5 (and consequently a mass ratio $< 1:5$). For more extreme mass ratios $\geqslant 1:20$ we find a gradual change in the behavior. This regime was not simulated by Hilz et al. (2012) and Oogi and Habe (2013).

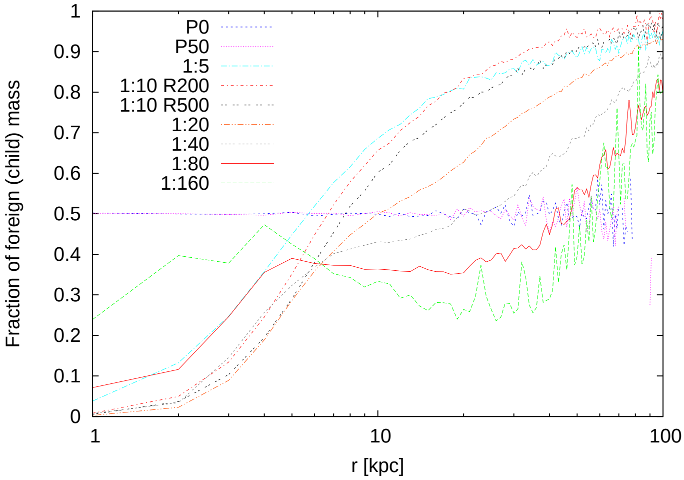


Figure 5.14: Origin of the mass in the merger remnant at $T = 10\,\mathrm{Gyr}$. Along the y-axis we present the fraction of mass that originates from the child galaxies at distance r from the center of the merger remnant. We binned the mass in 1 kpc bins. When the fraction drops to 0 then there is no contribution in mass from the child galaxies.

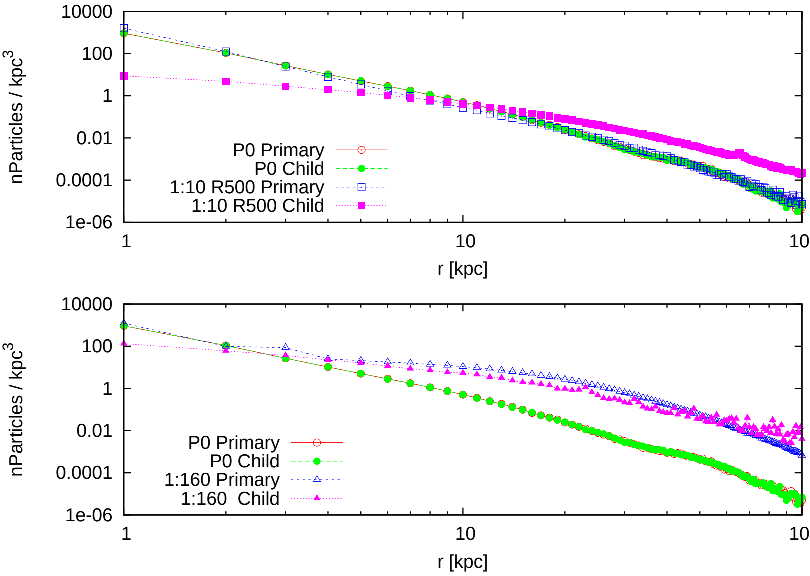
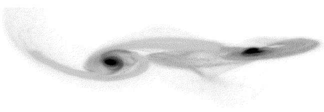


Figure 5.15: Particle distribution as function of distance from center, normalized to the volume size. The x-axis shows the distance to the remnants core, the y-axis the normalized particle count. This plot is similar as a density distribution, but now split into two lines indicating the origin of the particles (mass). The top and bottom panels both show the same averaged major merger result (P0). In addition the top panel shows the average result for the 1:10 $R = 500$ merger and the bottom panel the averaged result for the 1:160 mergers. Visible is that the 1:10 and major merger show different behaviour, where the major merger shows perfect mixing, the 1:10 shows a clear distinction between the primary and child galaxy mass distribution. The 1:160 however, shows a distribution that is closer to the major merger distribution than to the 1:10 distribution, indicating that 1:160 shows more signs of particle mixing.



5.6 Conclusion

We have studied dry minor mergers of galaxies in order to understand the observed growth in size without much increasing the mass of compact massive galaxies. The simulations ware performed the Bonsai GPU enabled tree code for N op to 17.6 million particles. Our simulations start with a major galaxy and a number of minor galaxies in a kinematically cold (and warm) environment. The total mass of the child galaxies equals that of the primary. The simulation were performed for 10 Gyr, and we study the size growth of the merger remnant. We demonstrate that mergers with a mass ratio 1:5 to 1:20 satisfactory explain the observed growth in size of compact galaxies. The growth of the merger remnant is always at least a factor of two higher than in the case of a major merger. This result is robust against variations in the initial density of the child galaxies. The mass of the minor galaxies tend to be accreted to the outside of the merger remnant. As a consequence the core of the merger remnant, formed by the original primary galaxy, will hardly be affected by the merging process. This 'inside-out-growth' is consistent with previous studies (Hilz et al. 2013).

If the number of child galaxies exceeds 20 the behavior of accretion changes, in the sense that the minor galaxies tend to accumulate in the central portion of the merger remnant. Evidence for this 'outside-in' growth is present in the density and velocity profiles of the merger remnant. The outside-in growth is mediated by self interactions among the child galaxies before they dissolve in the merger remnant.

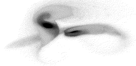
We conclude that the observed large massive elliptical galaxies can be evolved from compact galaxies at $z \gtrsim 2$ if they have grown in mass by accreting 5-10 minor galaxies with a mass ratio of 1:5 to 1:10. The majority of accreted mass will be deposited in the outskirts of the merger remnant.

Acknowledgments

We thank Marijn Franx and Steven Rieder for discussions. This work is supported by NWO grants (grants #643.000.802 (JB), #639.073.803 (VICI), and #612.071.305 (LGM)) and by the Netherlands Research School for Astronomy (NOVA).

5.A Resolution effects

As discussed in Sect. 5.2 we perform a range of simulations with various resolutions for different major and minor merger setups. Depending on the number of child galaxies we use low (LR) and high (HR) resolution configurations (see Tab. 5.1). In the LR configurations we adapted $N = 4.4 \times 10^5$ particles of which half belongs to the primary and the other half to the child galaxies. For the HR simulations we have $N = 4.4 \times 10^6$. To verify that the chosen resolutions are sufficient we ran a subset of our configurations with even higher resolutions. The results can be seen in Fig. 5.16. Here we present the size evolution of two major merger configurations, a 1:10 merger with $R = 200$ kpc and $R = 500$ kpc and one of the 1:80 mergers. Each configuration is run using the default



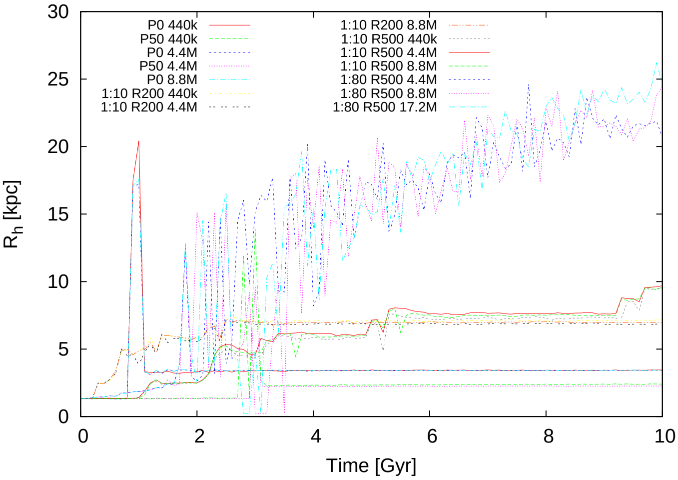


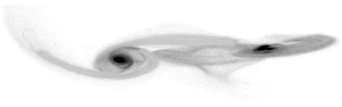
Figure 5.16: R_h versus time for a subset of configurations. Each configuration is run with three different resolutions to test dependence on the number of particles used. We present the results for two major merger configurations ($P = 0$ and 50 kpc), for two 1:10 minor merger configurations ($R = 200$ and ($R = 500$) and one 1:80 merger configuration.

resolution, as specified in Tab. 5.2, and using two higher resolutions For the 1:80 simulation this translates to $N = 4.4 \times 10^6$, $N = 8.8 \times 10^6$ and $N = 17.2 \times 10^6$. For the other configurations we use $N = 0.44 \times 10^6$, $N = 4.4 \times 10^6$ and $N = 8.8 \times 10^6$. For each of the resolutions we checked if the chosen time-step and softening was sufficient to keep the model stable in isolation using the method described in Sect. 5.2. In all but the 1:80 configurations there is little to no difference in the size of the merger remnant over the course of the simulation. Indicating that our choice of N is sufficiently large. The large amount of child galaxies in the 1:80 configurations cause child galaxies to interact with each other and not directly merge with the primary galaxy. The galaxies keep flying in and out of the remnant and makes it harder to determine R_h of the merger remnant which causes the large fluctuation in size of the 1:80 merger models. However, the trend of the R_h increase is independent of the used resolution. As final test we performed the 1:80 configuration with $N = 4.4 \times 10^5$ particles (not plotted) which quantitatively gives the same result, indicating that $N = 4.4 \times 10^6$ is adequate for the 1:80 configurations.

5.B The effect of child density

As described in Sect. 5.2 we do not change the density of the child galaxies when generating the initial conditions. Instead we change the cut-off radius to be a factor, f, smaller than that of the primary galaxy. This f scales with the mass ratio of the child galaxy. For example the cut-off radius of a child galaxy with mass ratio 1:10 is 10 times smaller than that of the primary. Together with the cut-off radius the mass is also reduced by a factor 10 giving the child galaxies a density lower than that of the primary galaxy (§ 5.2).

To test the effect of the density on the size growth of the merger product we generated two new 1:10 child galaxies. In addition to the `standard` child galaxy we created a `compact` and a `supercompact` child galaxy. These galaxies are generated such that their densities are higher than that of the `standard` child galaxy. The galaxy properties are



Model	Bulge mass [$M_\odot$]	Cut-off [kpc]	R_h [kpc]	ρ [10^{10} $M_\odot/kpc^3$]
Primary	2×10^{11}	100	1.36	1232
`standard`	2×10^{10}	10	1.26	142
`compact`	2×10^{10}	5.1	0.68	307
`supercompact`	2×10^{10}	0.9	0.25	4609

Table 5.3: Properties of the extra 1:10 child configurations. The first column indicates the model, either the primary or one of the three child configurations. The second column indicates the bulge mass. The third the cut-off radius for the dark matter halo, which is used to configure the galaxies. The fourth (fifth) column indicates the half-mass radius (density) of the galaxy in isolation at T=1 Gyr.

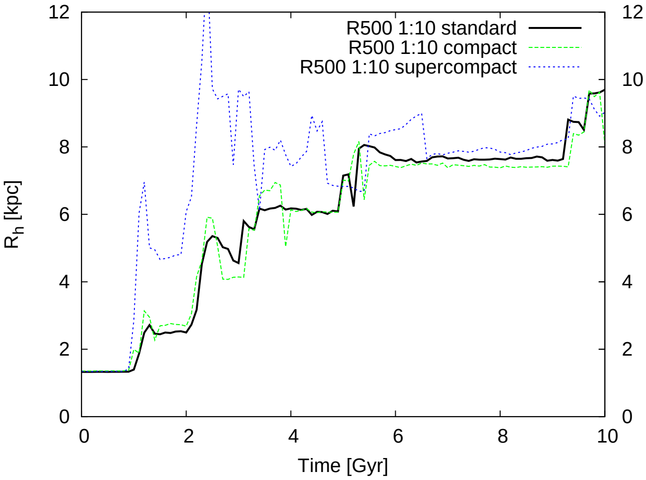
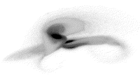


Figure 5.17: Size evolution. The configurations are based on the same primary galaxy and Plummer distribution, but the child galaxies have different size and density properties.

presented in Tab. 5.3.

With the new child galaxies in place we selected one of the 1:10 $R = 500$ kpc merger configurations and ran this realization with the new child galaxies. Each model is evolved for 10 Gyr using high resolution ($N = 4.4 \times 10^6$).

The effect of the child density on the size growth of the merger remnant is presented in Fig. 5.17, where we show R_h as a function of time. The `standard` and `compact` configurations show similar evolution. The `supercompact` configuration shows a different behaviour during the first 5 Gyr of the simulation. This is an artifact of the way we determine the size of the merger remnant which does not work perfect in this particular configuration (see § 5.4). The compactness of the galaxies causes the cumulative mass-profile to be irregular compared to the simulations with less dense child galaxies. As a consequence we are not always able to correctly measure the position in the profile that indicates the end of the merger remnant. After 5 Gyr the procedure works correct, because at this point the child galaxies have been involved in so many interactions that the mass-profile is smooth again. When the method starts working again we notice that all three simulations show similar size growth, independent of their density.



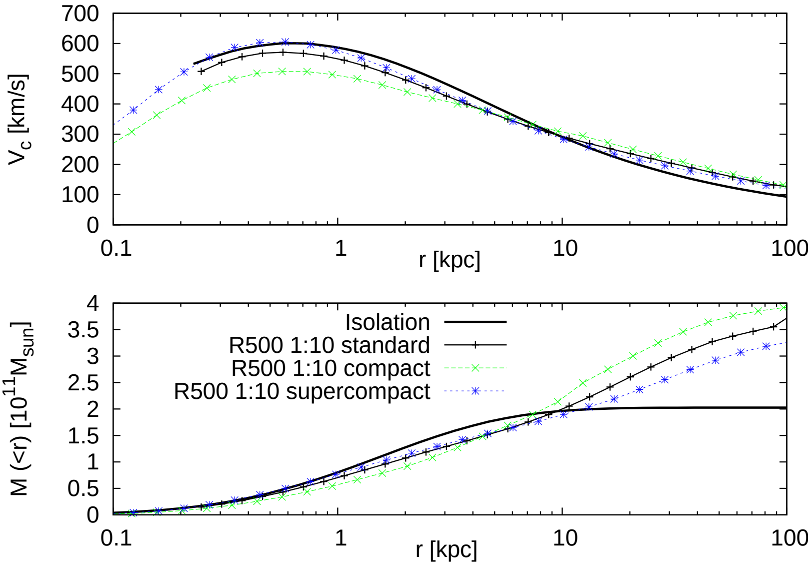
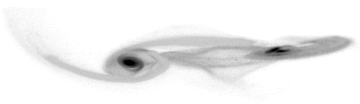


Figure 5.18: Circular velocity (top) and cumulative mass (bottom) after 10 Gyr for different merger configurations. The mergers are based on the same Plummer distribution but the child galaxies have different densities. The velocity and mass profile of the isolated model are indicated with the thick solid line.

5.B.1 Circular velocity

The circular velocity of the standard and high density child simulations is presented in Fig. 5.18 (top) we notice that the density of the child galaxy does affect the results. The simulation with the most dense child galaxies (striped line with stars) has a circular velocity similar to that of the isolated galaxy (thick solid line). The `compact` configuration has a circular velocity that is $\sim$ 100 km/s lower than the `supercompact` configuration. The `standard` configuration has a circular velocity in between the results of the `compact` and `supercompact` models. There is no clear trend in the effect the child density has on the circular velocity. A similar effect can be seen in the cumulative mass distribution of Fig. 5.18 (bottom). The difference between the runs is caused by the changed merger history. Even though the configurations are based on identical Plummer realisations the difference in density causes the merger history to be altered. Tidal stripping has less effect on the more compact galaxies when they move through the dark-matter halo of the primary than it has on the `standard` galaxy. Instead the higher density causes these galaxies to stay tightly bound resulting in interactions between other children and the core of the primary galaxy. Because of these interactions some of the child galaxies will be flung from the galaxy during the first passage instead of merging with the primary.

In Fig. 5.19 we present the distribution of accreted material throughout the merger remnant. In this figure we notice large differences between the different configurations caused by the differences in child density. The solid line represents the `standard` simulation where the mass is accreted on the outside of the primary (outside-in formation). For



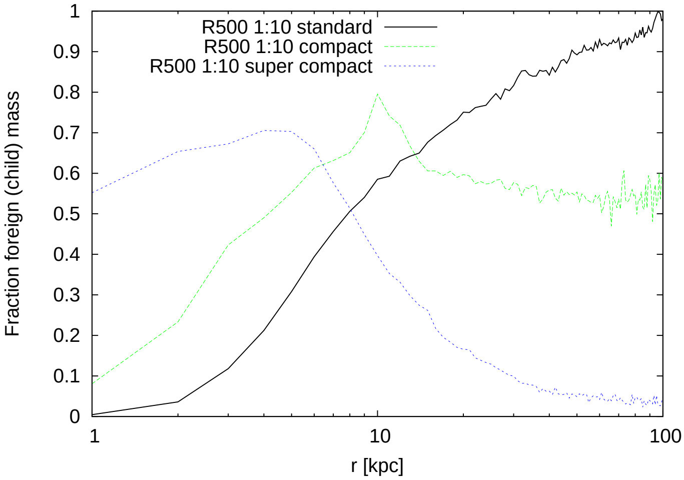
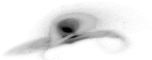


Figure 5.19: Origin of the mass in the merger remnant. Shown on the y-axis the fraction of mass that originates from the primary galaxy at a certain radius from the center. The mass is binned in 1 kpc sized bins. If the fraction is 0 then there is no contribution in mass from the child galaxies.

the denser children the mass is accreted closer to the core of the primary galaxy. This is the result of the less effective tidal-stripping described in the previous §. The dense child galaxies are able to progress further to the core of the merger remnant before they break up and are accreted. While the `standard` galaxies loose much of their material outside the core. This effect is best indicated by the `supercompact` configuration. In the merger remnant of this simulation most of the mass in the inner 8 kpc originates from the child galaxies. While in the other configurations most of the mass in this inner part of the merger remnant originates from the primary.

Concluding we can say that the density has no effect on the size of the merger remnant, but it does have an effect on how the remnant is build-up. With either mass being accreted on the outskirts (low density children) or in the inner parts of the merger remnant (high density children).



6

How to simulate the Milky Way Galaxy on a star-by-star basis

We report on the performance and optimizations carried out in our gravitational N-body tree-code `Bonsai` with which we plan to perform a simulation of the Milky Way Galaxy on a star-by-star basis. Here we report on the code optimizations and our preliminary simulations using 66 billion particles on the Titan supercomputer with up to 4096 GPUs and 65536 CPU-cores. Our code enables us to simulate the Galaxy on a star-by-star basis. To perform such simulations we completely redesigned the gravitational N-body tree algorithms to fully benefit from the parallel GPU and all the CPU-cores available per GPU. We have solved the load-balancing issues related to distributing the calculation effectively over many GPUs. As a consequence our calculations achieve a high sustained performance even with a relatively small number of particles. The GPU kernels compute at an aggregate rate of 7090 Tflops, while the sustained application performance is 4227 Tflops. At this speed we can simulate the 10 Gyr evolution of the entire Galaxy in 10 days.

Jeroen Bédorf, Evghenii Gaburov, Keigo Nitadori, Michiko S. Fujii, Tomoaki Ishiyama and
Simon Portegies Zwart

An extended and improved version of this chapter is published in the SC14 proceedings, 2014

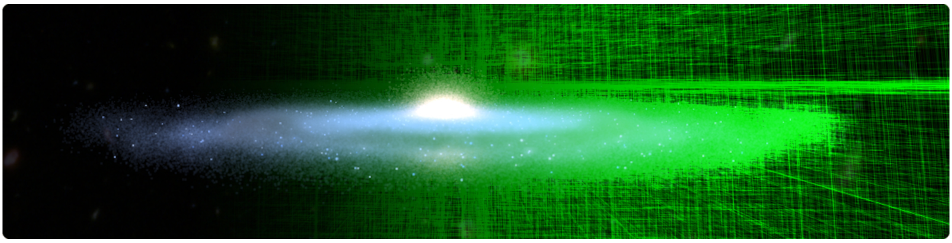


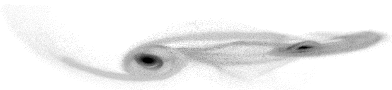
Figure 6.1: Edge-on view of the simulated Milky Way Galaxy. To the left is an optical representation, the computer code structure is illustrated to the right (in green).

6.1 Introduction

The Milky Way Galaxy is composed of about 100 billion stars that form a disk and nucleus surrounded by a dark matter halo. Each star in the Galaxy attracts each other star. Numerically such gravitational interactions are most accurately simulated via direct N-body force evaluations, in which each of the N objects requires the resolution of $N-1$ interactions per time-step (Heggie and Hut 2003). The N^2 time complexity of this algorithm, however, would render the time-to-solution for a galaxy simulation unrealistically long. Most Galaxy simulations therefore adopt the hierarchical Barnes-Hut tree algorithm (Barnes and Hut 1986), which has an $N\log(N)$ time complexity. Figure 6.1 depicts an image of the simulated Galaxy in which an 'observational' rendition is presented to the left that morphs into a numerical representation of the tree-structure in the N-body code towards the right.

Earlier simulations of galaxies have been performed using tree-codes with five hundred thousand (Besla et al. 2010) to tens of millions (Bédorf and Portegies Zwart 2013; Dubinski and Chakrabarty 2009) particles. Each particle in these simulations represents a total mass of $\gtrsim 5000\ \mathrm{M}_{\odot}$. These studies are used to understand the structure of the bar (Athanassoula 2012), spiral arm formation (Dubinski et al. 2009; D'Onghia et al. 2013) and dynamics (Fujii et al. 2011; Grand et al. 2012b; Baba et al. 2013), pitch angle and galactic shear (Grand et al. 2013) and the warping of the stellar disk (Dubinski and Chakrabarty 2009). The largest simulation of a disk galaxy so far has been performed for educational purposes, with 100 million particles (Dubinski 2008). Recently Fujii et al. (Fujii et al. 2011) and Sellwood (Sellwood 2013) demonstrated that the relatively small number of particles adopted in those simulations is not sufficient to accurately resolve relaxation processes and tend to overproduce the dynamical heating of the disk. They conclude that the number of particles in disk galaxy simulations should be increased by at least a factor 100 in order to accurately reproduce the intricate non-linear dynamics of the internal structure of a Milky Way sized galaxy (Sellwood 2013) (although this number is quite uncertain (Fujii et al. 2011)). This realization has lead to the requirement to further optimize tree-codes for accuracy and performance.

Previous tree-codes used for simulating galaxies are not as well optimized as the code we describe here. Some of these earlier codes are parallel but scale relatively poor to large



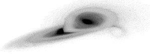
number of processors, typically with an $\lesssim 50\%$ efficiency for up-to 128 cores (Fortin et al. 2011). There is a clear need for improving the performance and concurrency before it becomes possible to simulate the Milky Way galaxy with sufficient resolution to resolve these issues. With the code described in this paper we are able to simulate a galaxy on a star-by-star basis, which should be sufficient resolution to address these questions and may therefore change our understanding of the dynamical evolution of disk galaxies.

We adopted the classic Barnes-Hut tree algorithm (Barnes and Hut 1986; Barnes 1990) for our simulations. In this algorithm the distribution of particles is recursively partitioned into octants until the number of particles in an octant is smaller than a critical value (we use 16). Once a tree-structre is built and its multipole moments are computed, the code proceeds with the force calculation step. For this, we adopt a geometric angle criterion, called the opening angle (or multipole acceptance criterion), θ, whose purpose is to decide whether or not the substructure in distant octants can be used as a whole. If the opening angle is infinitesimal the tree-code reduces to a rather inefficient direct N-body code with leap-frog time integrator. For a finite θ, however, sufficiently distant octants from a target particle can be used as a whole, and the partial forces from the constituent particles are calculated via a multipole expansion approximation. The time complexity for calculating the force between all particles in the entire system reduces with the tree-code to $\propto N \log(N)$.

The tree algorithm is quite efficient in calculating forces and parallelizes reasonably well to many cores. Tree-codes have therefore been used in various incarnations such as TreePM (Xu 1995; Bode et al. 2000; Bagla 2002; Dubinski et al. 2004; Springel 2005; Yoshikawa and Fukushige 2005; Ishiyama et al. 2009; Portegies Zwart et al. 2010). Most of the previous tree and TreePM codes have been tuned for massively parallel supercomputers without accelerators (Warren and Salmon 1991; Dubinski et al. 2004; Springel 2005; Ishiyama et al. 2009; Portegies Zwart et al. 2010; Ishiyama et al. 2012) except for codes optimized for relatively small clusters including accelerators such as GPUs and GRAPEs (Makino 2004; Yoshikawa and Fukushige 2005; Hamada et al. 2009b; Hamada and Nitadori 2010; Nakasato et al. 2012). The next logical step is to develop a tree-code which runs efficiently on massively parallel supercomputers with accelerators, because such systems are becoming mainstream for the current and next generation supercomputers as represented by Titan (at the Oak Ridge National Laboratory). The move from CPU-based to GPU-based supercomputers is motivated by smaller cost per flop and considerably smaller CO_2 footprint of the latter.

The new accelerator technology requires a complete redesign of the algorithm and optimization strategy in order to achieve high performance, but the reward is a tenfold increase of performance on a single GPU compared to the CPU. On Titan, which is the world's largest GPU supercomputer, at the time of writing, the theoretical peak performance is 3.95 Tflops per node (in single precision and excluding CPU performance), but it is 0.128 Tflops per node on K computer. The memory and network performance per flop is therefore much smaller on Titan than it is for K computer. While achieving high performance on the GPU, we increase the need for a high-speed network, which requires an even more aggressive strategy for minimizing the communication cost. In addition, the added GPU also causes an extra bottleneck to overcome in the communication with its host.

While accelerators greatly improve the efficiency per node, massive parallel comput-



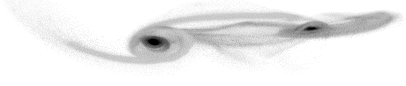
ing with GPUs has become one of the most challenging problems. One crucial bottleneck is the relatively slow communication between the GPU and its host, which directly limits the overall performance. Designing a strategy to minimize this communication will therefore result in a general improvement of a wide variety of algorithms because our optimization strategy is not specific to N-body simulations. Nor is our optimization strategy specific to Titan, but, in fact, can be applied on any large GPU-equipped supercomputer, such as the Tsubame (at the Tokyo institute of Technology's Global Scientific information Center) series of supercomputers, HA-PACS (at the University of Tsukuba), Tianhe (National Supercomputing Center of Tianjin), Nebulae (NSCS), PLX-GPU (CINECA) and LGM (Leiden). We have realized two important improvements compared to earlier work with parallel tree-codes, which are: 1) porting the tree-code, including the tree-building and tree-walk, entirely to the GPU and 2) utilizing the host CPU for orchestrating the communication, administrative purposes, force-feeding the GPU and data reduction. Improvement 1) allowed us to achieve high performance by efficiently utilizing each GPU and improvement 2) enables us to scale the code to 4096 nodes even with a rather modest number of particles.

6.2 Quantitative discussion of current state of the art

We adopted the classic Barnes-Hut tree algorithm (Barnes and Hut 1986; Barnes 1990) for our simulations. In this algorithm the distribution of particles is recursively partitioned into octants until the number of particles in an octant is smaller than a critical value (we use 16). Once a tree is built and its multipole moments are computed, the code proceeds with the force calculation step. For this, we adopt a geometric angle criterion, called the opening angle (or multipole acceptance criterion), θ, whose purpose is to decide whether or not the substructure in distant octants can be used as a whole. If the opening angle is infinitesimal the tree-code reduces to a rather inefficient direct N-body code with leap-frog time integrator. For a finite θ, however, sufficiently distant octants from a target particle can be used as a whole, and the partial forces from the constituent particles are calculated via a multipole expansion approximation. The time complexity for calculating the force between all particles in the entire system reduces with the tree-code to $\propto N \log(N)$.

The tree algorithm is quite efficient in calculating forces and parallelizes reasonably well to many cores. Tree-codes have therefore been used in various incarnations such as TreePM (Xu 1995; Bode et al. 2000; Bagla 2002; Dubinski et al. 2004; Springel 2005; Yoshikawa and Fukushige 2005; Ishiyama et al. 2009; Portegies Zwart et al. 2010).

Most of the previous tree and TreePM codes have been tuned for massively parallel supercomputers without accelerators (Warren and Salmon 1991; Dubinski et al. 2004; Springel 2005; Ishiyama et al. 2009; Portegies Zwart et al. 2010; Ishiyama et al. 2012) except for codes optimized for relatively small clusters including accelerators such as GPUs and GRAPEs (Makino 2004; Yoshikawa and Fukushige 2005; Hamada et al. 2009b; Hamada and Nitadori 2010; Nakasato et al. 2012). The next logical step is to develop a tree-code which runs efficiently on massively parallel supercomputers with accelerators,



because such systems are becoming mainstream for the current and next generation supercomputers as represented by Titan (at the Oak Ridge National Laboratory). The move from CPU-based to GPU-based supercomputers is motivated by smaller cost per flop and considerably smaller CO_2 footprint of the latter. For example, K computer energy efficiency is 892.86 Mflops/Watt compared to only 3333 Mflops/Watt for Titan[1]. We therefore expect that new exascale supercomputers will be equipped with some type of accelerator (see § 6.7).

The new accelerator technology requires a complete redesign of the algorithm and optimization strategy in order to achieve high performance, but the reward is a tenfold increase of performance on a single GPU compared to the CPU. On Titan, which is the world's largest GPU supercomputer, the theoretical peak performance is 3.95 Tflops per node (in single precision and excluding CPU performance), but it is 0.128 Tflops per node on K computer. The memory and network performance per flop is therefore much smaller on Titan than it is for K computer. While achieving high performance on the GPU, we increase the need for a high-speed network, which requires an even more aggressive strategy for minimizing the communication cost. In addition, the added GPU also causes an extra bottleneck to overcome in the communication with its host.

While accelerators greatly improve the efficiency per node, massive parallel computing with GPUs has become one of the most challenging problems. One crucial bottleneck is the relatively slow communication between the GPU and its host, which directly limits the overall performance. Designing a strategy to minimize this communication will therefore result in a general improvement of a wide variety of algorithms because our optimization strategy is not specific to N-body simulations. Nor is our optimization strategy specific to Titan, but, in fact, can be applied on any large GPU-equipped supercomputer, such as the Tsubame (at the Tokyo institute of Technology's Global Scientific information Center) series of supercomputers, HA-PACS (at the University of Tsukuba), Tianhe (National Supercomputing Center of Tianjin), Nebulae (NSCS), PLX-GPU (CINECA) and LGM (Leiden). We have realized two important improvements compared to earlier work with parallel tree-codes, which are: 1) porting the tree-code, including the tree-building and tree-walk, entirely to the GPU and 2) utilizing the node CPU for orchestrating the communication, administrative purposes, force-feeding the GPU and data reduction. Improvement 1) allowed us to achieve high performance by efficiently utilizing each GPU and improvement 2) enables us to scale the code to 4096 nodes even with a rather modest number of particles.

6.3 Implementation

Given the enormous computational capabilities of the Kepler GPU, communication between the MPI processes can easily become the major scalability barrier, especially when there are 1000+ nodes[2].

In our largest simulation of 65.5 billion particles on 4096 GPUs (16 million particles per GPU), the GPU tree-walk kernel on the local data completes in two seconds, within

---

[1] see `http://www.top500.org/`

[2] We use one GPU per MPI process.

which most of the communication among all 4096 GPUs must be hidden. Equivalent CPU code require more time, by an order of magnitude, for this operation and hiding communication behind the calculation is then considerably easier.

Here we describe our optimizations for K20X hardware of the GPU tree-walk kernel and our approach to minimizing and hiding communication. Our parallel GPU tree code is called `Bonsai` and is publicly available for download[3].

6.3.1 Tree-walk kernel optimizations

The single GPU code consists of three fundamental parts: tree-construction, computation of multipole moments, and tree-walk in which inter-particle gravitational forces are computed. In contrast to previous works (Hamada et al. 2009b; Hamada and Nitadori 2010), in `Bonsai` all of these steps are carried out on the GPU, leaving the CPU with lightweight tasks such as data management and kernel launches.

The assimilation of the tree-walk and force computations within a single GPU kernel allows us to achieve excellent computational efficiency by not wasting GPU bandwidth to store particle interaction lists into memory. Instead, the interaction lists are stored in registers and evaluated on-the-fly during the tree-walk, therefore delivering superb performance in excess of 1.7 Tflops on a single K20X. The details of the tree-walk on NVIDIA Fermi architecture are described in (Bédorf et al. 2012) or see Chapter 4.

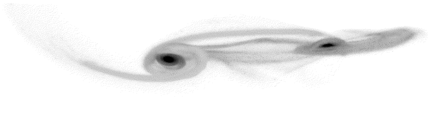
It is considerably more efficient to walk the tree on a GPU by a group of spatially nearby particles rather than by one particle at a time (Barnes 1990). These nearby particles have similar paths through the tree, and therefore similar interaction lists; by building an interaction list that is valid everywhere within the group, we reduce both the number of tree-walks and we make each of them efficient via thread cooperation inside a thread-block. In our previous work the grouping is based on the underlying tree-structure, such that tree-cells with the number of particles below a certain number (we use 64) are used as a group. However, due to the geometric nature of space partitioning by octrees, the average number of particles in such a group was much smaller than 64, which resulted in a waste of compute resources. We solved this by sorting the particles into a Peano-Hilbert space filling curve (PH-SFC) (Hilbert 1891) and splitting it into groups of 64 particles. The final criterion is to enforce a maximal geometric size of a group: if a group exceeds this size it is recursively split further into smaller groups.

The original gravity kernel was tailored to the Fermi architecture and contained a large number of prefix sum operations. These operations are used to communicate between threads in a thread-block, which is required for the on-the-fly construction and evaluation of the interaction lists during the tree-walk phase. This required on average about 1.2KB of shared memory per warp[4]. While efficiency of such a kernel on Fermi was good, we noticed that its performance on Kepler was below the expectations, in particular when one considers the ratio of the peak performance between the Kepler and Fermi chips. We suspected that the excessive use of shared memory is at fault for the lower than expected

---

[3] `https://github.com/treecode/Bonsai/tree/Titan`

[4] A warp is a group of threads which are executed in lock-step. On current NVIDIA hardware, a warp has 32 threads. Multiple warps make up one thread-block, and warps in a thread-block can be synchronized with the `__syncthreads()` intrinsic which allows inter-warp communication via share-memory.



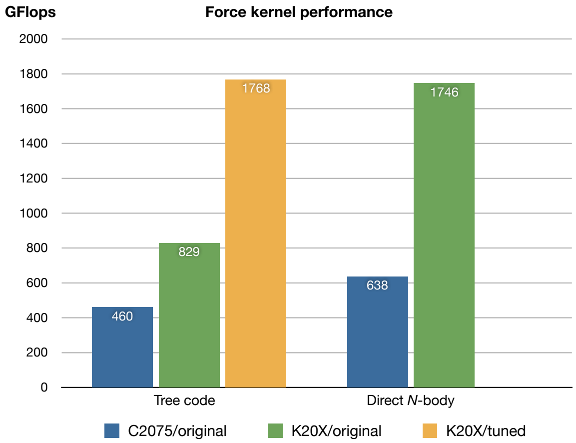


Figure 6.2: Performance of the gravitational force kernel. The blue bars indicate the performance of the Fermi kernel on the C2075. The green bars indicate the performance of the Fermi kernel on a K20X. The gold bar shows the performance of the K20X tuned kernel. With tuning, the K20X is twice as fast as the original kernel, and is 4x faster than the C2075. Also presented here is the performance of a direct N-body kernel running on the same hardware using NVIDIAs CUDA SDK 5.5.

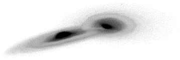
kernel performance, since it is much slower than registers, in particular on the Kepler chips in combination with 4-byte bank mode (all the data stored in shared memory are either floats or integers). Furthermore, because we already updated the Fermi kernel to be warp-synchronous, we can now eliminate most of the shared memory in favour of registers with help of the new Kepler `__shfl` instructions, which allow intra-warp communication without the need of shared memory. During this optimization step we also took advantage of the `__ballot` and `__popc` instructions to convert our binary prefix sums away from shared memory and `__shfl` (for a better performance we write some of the basic prefix-sum related code using inline PTX). These optimizations resulted in a more efficient code and a reduction of the shared memory requirement by 90% (128 bytes per warp). The kernel runtime was reduced by $2\times$ and is now consistent with our expectations based on the peak performance ratios. In Fig. 6.2 we show the performance of the different implementations.

6.3.2 Parallelization

Maintaining `Bonsai's` excellent single-GPU efficiency when scaled to many GPUs requires both the minimization of the amount of data traffic between different GPUs, and hiding the communication steps behind computations. We achieve this by carefully selecting, combining and modifying different well-known parallelization strategies. After experimenting we eventually settled on a combination of a Local Essential Tree (LET) and Space Filling Curve (SFC) methods.

In the original LET method, the physical domain is divided into rectangular sub-domains via a recursive multi-section algorithm. Each process uses these sub-domains to determine which part of its local data will be required by a remote process. This part is called the Local Essential Tree structure. After a process has received all the required LET structures, they are merged into the local tree to compute the gravitational forces.

In the SFC method, particles are ordered along an SFC curve which is split into equal pieces that define the sub-domain boundaries. The latter will no longer be rectangular; but they have fractal boundaries instead. In Fig. 6.3 we illustrate a SFC and its decomposition



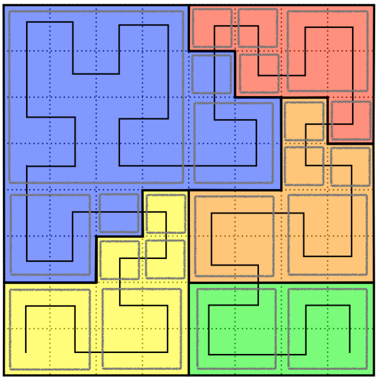


Figure 6.3: Example of domain decomposition with Peano-Hilbert space filling curve (black solid line). The colors indicate the five separate domains. The gray squares correspond to the tree-cells, which consist solely of particles belonging to a single process.

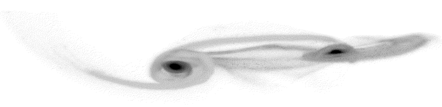
to 5 GPUs. This makes it harder to build a compact LET, and instead common SFC-based tree-codes either export particles to remote processes and import results back (Springel 2005), or request sub-trees from the remote processes (Warren and Salmon 1993; Winkel et al. 2012). Such methods generate multiple communication steps during the tree-walk.

The LET method requires the least amount of communication. Therefore we combined LET with the SFC domain decomposition which guarantees that sub-domain boundaries are tree-branches of a hypothetical global octree. This step allows us to skip merging the imported structures into our local-tree, but rather process them separately as soon as they arrive, therewith permitting to hide communication behind computations.

Domain Decomposition

Each GPU computes its corresponding local domain boundaries, and the CPUs determine global domain boundaries which are used for mapping particle coordinates into corresponding Peano-Hilbert (PH) keys (Hilbert 1891). Process 0 subsequently gathers a sample of PH-keys from the remote processes and combines these into a global PH-SFC. This SFC is cut into p equal pieces and the beginning and ending PH key determines the sub-domains of the global domain, which are broadcast to all of the processes. We use two load balancing strategies: one is using an approximately equal number of particles per process, and the other is weighting this number by the total number of flops by the GPU tree-walk kernel.

With the domain boundaries at hand, each GPU generates a list of particles that are not part of its local domain, and these particles are exchanged between processes. After the particle data exchange is completed, each GPU rebuilds its tree-structure and computes the corresponding multipole moments. At the end of this step, each process has all necessary information to proceed with the force calculation. We would like to stress that each local tree is a non-overlapping branch of a global octree, because we use the PH-SFC as a basis for the domain decomposition. This guarantees binary consistency of the domain decomposition independently of the number of processes, and allows us to hide LET communication behind computation.



Computing the gravity

Due to the long-range nature of Newton's universal law of gravitation (Newton 1687), the computation of mutual forces is by definition an all-to-all operation. Therefore, to compute forces on local particles, a target process requires communication with all other processes. We do this by forcing remote processes to send the required particle and cell data (LETs) to the target process. While the GPU on this target process is busy computing forces from the local particles, the CPU is busy preparing particle data for export, as well as sending and receiving data.

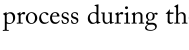
The preparation of particle data for export to remote processes is both floating point and memory bandwidth intensive, which must overlap with the communication between the processes. We achieve this via multi-threading, with which we split each MPI process into three thread-groups: one thread is responsible for MPI communication, to which we refer as the communication thread, another thread drives the GPU, which we call the driver thread, and the rest of the threads are busy computing, which we collectively call the compute threads.

In the first step the compute threads check whether it is sufficient to send only top level parts of the local tree to each remote process. If so, the necessary data is scheduled for export. When checks are completed, the communication thread exchanges the prepared data with the rest of the processes via a customized `all-to-all` communication, which uses a two-dimensional communicator splitting algorithm (Ishiyama et al. 2012) to reduce latency, but at the price of doubling the data traffic. This step communicates $\sim$90% of the LET trees.

In the second step, the compute threads prepare the LET structures to be exported to remote processes that require additional data to what was communicated in the first step. At the same time, the communication thread is busy sending prepared LET data as it arrives from the compute threads, as well as receiving LET data from remote processes which it passes to the driver thread. The driver thread merges these LET trees into a new tree, in which LET structures form branches. In other words, this builds the hypothetical global tree on the fly, but only using LET data currently received from remote processes. Whenever the GPU is ready with the gravitational force kernel, either on the local data or the already received remote data, the newly built LET tree is force-fed to the GPU.

Once the gravitational forces of all local particles are computed, the particles are advanced forward in time using a 2^{nd}-order leap-frog integration scheme (Hut et al. 1995) adopting a shared time-steps. Currently `Bonsai` supports block time-step integration mode (McMillan 1986) in the serial implementation, in which a subset of the groups will be advanced in time. A group is added to the subset if at least one of the particles belonging to the group has to be updated. This is determined using the time-step criteria, which is based on the particles crossing time and free-fall time. In the block time-step method the computation of the gravitational forces scales linearly with the number of active groups.

This block-time stepping can be extended to the parallel implementation. Currently the boundaries of the full-tree (all particles) are used to exchange the LET structures. If we use the boundaries of the active groups instead, in general the physical domain size, which is composed of active groups in the process, should be reduced. This in turn reduces the amount of data that needs to be pushed to the remote MPI process during the LET



exchange phase. The data still has to be sent to the processes containing active groups, but the amount of data is reduced. The long-range nature of the gravity makes it non-trivial to eliminate the communication phase entirely, but for most processes only the root-node of the local-tree (which is small) has to be communicated. This is mostly a latency bound operation.

6.4 Simulating the Milky Way Galaxy

We performed N-body simulations of the Milky Way Galaxy, which is composed of a disk of stars with a central bulge and a surrounding dark matter halo. We adopted a typical model for the Milky Way (Widrow et al. 2008), which has an NFW (Navarro et al. 1996) density distribution with a mass of 6.7×10^{11} $\mathrm{M}_\odot$ for the dark matter halo, an exponential disk of 4.0×10^{10} $\mathrm{M}_\odot$ for the galactic disk, and Sérsic density profile with a mass of 6.4×10^{9} $\mathrm{M}_\odot$ for the bulge.

For our largest simulation using these initial conditions, we adopted 1 billion particles, which is an order of magnitude larger than any simulations of the Milky Way Galaxy conducted before. We adopted an opening angle of $\theta = 0.4$ which is sufficiently small to resolve the fine structures of spiral arms. Generating these initial conditions is quite expensive, in terms of computational time. With a snapshot size of about 37 GByte, by extrapolation, an initial conditions snapshot with 66 billion particles will be 2.4 TByte in size and is impractically long to generate with the current galaxy generation codes. We therefore adopted a Plummer model (Plummer 1911) generated at runtime as initial conditions for our performance measurements.

A Plummer model is spherical and in virial equilibrium, but with a density gradient similar to the NFW model (Navarro et al. 1996), which we adopted for our Milky Way model. As we show in the following section, the performance of running a Plummer model is therefore quite similar to that of the NFW model.

Our weak-scaling measurements are performed using 16 million particles per node totaling about 66 billion particles for our maximum of 4096 nodes. The mass resolution of these runs is about $10\,\mathrm{M}_\odot$ per particle, which is more than a factor of 500 smaller than in earlier simulations of the Milky Way Galaxy, which achieved > 5000 $\mathrm{M}_\odot$ per particle.

Our simulations are unique in that we now can resolve all scales simultaneously and self-consistently, from the dynamics of individual stars to the global dynamics of the Milky Way Galaxy. We focus on the self-consistent gravitational evolution, which is well understood from first principles and does not require any underlying assumptions. The main numerical problems hide in the wide range of scales to cover and the massive computational requirements.

6.5 System and environment where performance was measured

The tree-code we use to perform our calculations was optimized to run on the NVIDIA Fermi and Kepler architectures. The highest performance is obtained on the K20X (GK110

Setup	LGM	HA-PACS	Titan
GPU model	GTX480	M2090	K20X
GPU/node	2	4	1
Total GPU	24	1024	18688
GPUs used	24	1024	4096
GPU RAM	1.5 GB	5.4 GB	5.4 GB
CPU model	Xeon E5620	Xeon E5-2670	Opteron 6274
CPU/node	2	2	1
Total CPU	24	512	18688
CPUs used	24	512	4096
Node RAM	24GB	128GB	32 GB
Network	IB-SDR	IB-QDRx2	Cray Gemini

Figure 6.4: Hardware used for our parallel simulations. The first two machines have Intel based architectures with multiple GPUs per node. The last machine is Titan with 1 GPU per node. Our code was initially developed on LGM, but the results reported here were produced using Titan.

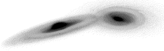
architecture), because of the hardware improvements introduced with the Kepler generation GPUs. The code operates on distributed memory systems with one or more GPUs per node. So long as there are enough CPU cores per GPU, we can run our code efficiently on several architectures, including the Little Green Machine at Leiden University, HA-PACS at the University of Tsukuba and on Titan at Oak Ridge National Laboratory. In Tab. 6.4 we summarize the specific hardware configuration for these machines.

Titan is by far the largest machine on which we have run our code and the results presented in this work are all obtained by runs on this machine. The Titan architecture allows us to use all available 16 CPU cores per GPU. Although other configurations will work similarly, we regularly refer back to this specific runtime architecture. In full operation Titan is composed of 18,688 nodes each containing an AMD Opteron 6274 16-core CPUs and one NVIDIA Tesla K20X GPU. The nodes are connected via a 3D torus network and are using the Cray Linux Environment distribution.

The HA-PACS architecture allows for better efficiency since the available CPU performance normalized per GPU performance is higher during the LET construction. With more than one GPU per node, the total amount of network communication is reduced since more communication can take place through shared memory. Furthermore, since the Fermi GPU is 4 times slower during the gravity phase it offers more time to hide the communication time. This improves the efficiency, but also delivers a lower application performance.

The Little Green Machine (LGM) in Leiden (the Netherlands), is a stereotypical low-budget GPU cluster as are built by many large academic research groups. The system contains two quad-core CPUs and 2 GPUs per node. For a total of about 20 nodes. Most of the initial development took place on this machine.

The `Bonsai` code is optimized for the latest NVIDIA CUDA architectures and for the measurements in this work we only take into account the version that is optimized for the K20X (see § 6.3). The code runs on any GPU-CPU-hybrid architecture and uses all available CPU cores per GPU. The CPUs are then used for network communication, for preparing data to be sent over the network and for force feeding the GPU. The GPUs handle the data processing and create the other data structures.



6.6 Performance results

We present the performance results by calculating floating point operations comprising only force calculations. We ignore contributions from tree construction, computation of multipole moments, multipole acceptance criteria during the tree-walk, all LET-related operations (on the CPU), initial condition generation, diagnostics, file I/O, etc. The timing of these operations however, are taken into account when we calculate the time-to-completion.

6.6.1 Operation counts

The accelerations and potential of a particle i, $\boldsymbol{a}_i$ and ϕ_i, which we collectively call the force, are computed by

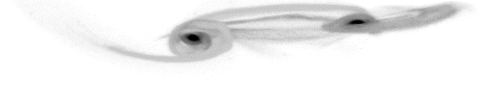
$$\begin{aligned} \phi_i &= \sum_j \left[-\frac{m_j}{|\boldsymbol{r}_{ij}|} + \frac{1}{2}\frac{\mathrm{tr}(\mathbf{Q}_j)}{|\boldsymbol{r}_{ij}|^3} - \frac{3}{2}\frac{\boldsymbol{r}_{ij}^T \mathbf{Q}_j \boldsymbol{r}_{ij}}{|\boldsymbol{r}_{ij}|^5} \right], \qquad (6.1) \\ \boldsymbol{a}_i &= \sum_j \left[\frac{m_j \boldsymbol{r}_{ij}}{|\boldsymbol{r}_{ij}|^3} - \frac{3}{2}\frac{\mathrm{tr}(\mathbf{Q}_j)\boldsymbol{r}_{ij}}{|\boldsymbol{r}_{ij}|^5} \right. \\ &\qquad \left. - \frac{3\mathbf{Q}_j \boldsymbol{r}_{ij}}{|\boldsymbol{r}_{ij}|^5} + \frac{15}{2}\frac{(\boldsymbol{r}_{ij}^T \mathbf{Q}_j \boldsymbol{r}_{ij})\boldsymbol{r}_{ij}}{|\boldsymbol{r}_{ij}|^7} \right], \qquad (6.2) \end{aligned}$$

with $\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i$. Here, m_j, $\boldsymbol{r}_j$, $\mathbf{Q}_j$ are mass, position, quadrupole moments (in a 3×3 symmetric matrix) of particle j respectively. Computation of one particle-particle interaction (p-p) without the quadrupole moment term consists of 4 subtraction (sub), 3 multiplication (mul), 6 fused-multiply-add (fma), and 1 reciprocal-square-root (rsqrt) instructions. In this article, we count 4 floating-point operations for the reciprocal-square-root, which results into a total of **23** operations for p-p. A particle-cell interaction (p-c) with quadrupole corrections consists of 4 sub, 6 add, 17 mul, 17 fma and 1 rsqrt, which results in **65** operations for p-c interaction[5]. The total number of flops is obtained by multiplying these numbers by the total number of p-p and p-c (as recorded during execution), and divided by the execution time.

Note that (Warren and Salmon 1992; Warren et al. 1998; Kawai et al. 1999; Hamada et al. 2009b; Hamada and Nitadori 2010) used **38** for the operation count of a p-p interaction. Although it is convenient to use the same operation count for comparing one record with the other, this can overcount the operations for hardware with fast rsqrt support. The last GBP winner (Ishiyama et al. 2012) counted **51**, within which about the half was paid for the calculation of a cut-off polynomial.

---

[5] The operation counts were verified with a disassembling command `cuobjdump -sass` in the CUDA toolkit.



Operation	computation unit		wall-clock time [s]		
	CPU	GPU	1 GPU	1024 GPUs	4096 GPUs
Sorting SFC	—	X	0.17	0.16	0.18
Domain Update	X	—	—	0.36	1.01
Tree-construction	—	X	0.13	0.14	0.14
Tree-properties	—	X	0.04	0.09	0.10
Compute gravity Local-tree	—	X	4.30	1.84	1.84
Compute gravity LETs	—	X	—	3.53	3.91
Compute gravity GPU	—	X	4.30	5.37	5.75
Gravity + LET	X	X	—	5.44	7.06
Other	X	X	0.20	1.34	1.01
Total			4.80	7.53	9.50

Interaction type	interaction count per particle		
	1 GPU	1024 GPUs	4096 GPUs
Particle-Particle	1724	1720	1649
Particle-Cell	6595	8338	8983

Table 6.1: Time breakdown for three different setups; single GPU, 1024 GPUs and 4096 GPUs. We use the weak-scaling results with 16M particles per GPU using a Plummer distribution and $\theta = 0.4$. The data presented shows the major parts of the algorithm and by which computation unit they are primarily executed. The first column indicates the function. The second and third the computation units used. The following three columns present the timings for 1 node, 1024 nodes and 4096 nodes respectively. The 'Gravity+LET' row indicates the time it takes to compute the gravity including communication. If this is equal to the 'Compute gravity GPU' then all the communication time of the gravity step is hidden. The 'Other' data includes the time to allocate memory, integrate particles and any waiting times induced by load-imbalance. The bottom two rows give the average number of particle-particle and particle-cell interactions.

6.6.2 Parallel performance

To evaluate the scalability and parallel performance of `Bonsai` we conduct both weak and strong scaling tests. In both cases we use the virialized Plummer model(Plummer 1911) as initial conditions (see § 6.4). In the weak scaling test we used 16 million particles per GPU, and strong scaling is tested with a 1 billion particle Plummer model. We explore weak scaling of `Bonsai` from 1 to 4096 GPUs, and strong scaling from 64 to 1024 GPUs. It is currently not possible to use fewer than 64 GPUs in the latter case due to limited amount of GPU memory: the parallel version of `Bonsai` is currently unable to hold more than 20 million particles using a K20X GPU. We also conducted a strong scaling test with a 1-billion-particle Milky Way Galaxy model. In both tests we use $\theta = 0.4$ as opening angle, which is satisfactory to properly model disk galaxies, such as the Milky Way (Bédorf and Portegies Zwart 2013).

In Fig. 6.5 we show both weak scaling results and parallel efficiency. In particular, we would like to stress that the efficiency of `Bonsai` on 4096 GPUs is almost 70%, and the average time spent per iteration in this 65.5 billion particle simulation is 9.6 sec. In Tab. 6.1 we present the breakdown of the timing results.

In Fig. 6.6 and Fig. 6.7 we present the strong scaling results up to 1024 GPUs using

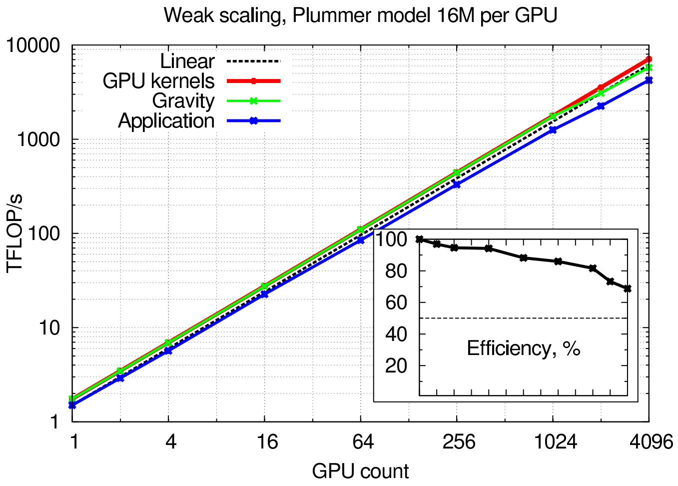


Figure 6.5: Weak scaling for 1 to 4096 nodes with 16 million particles per node. The red solid line shows the performance of the tree-walk kernel running on the GPU. The green solid line shows the performance of the gravity step, including the time to communicate LET structures. The blue solid line shows the performance of the full application. The dashed black line indicates linear scaling and finally the solid black line shows the parallel application efficiency with respect to a single GPU. The lower right inset gives the efficiency of the code from 1 (100%) to 4096 (67%) nodes.

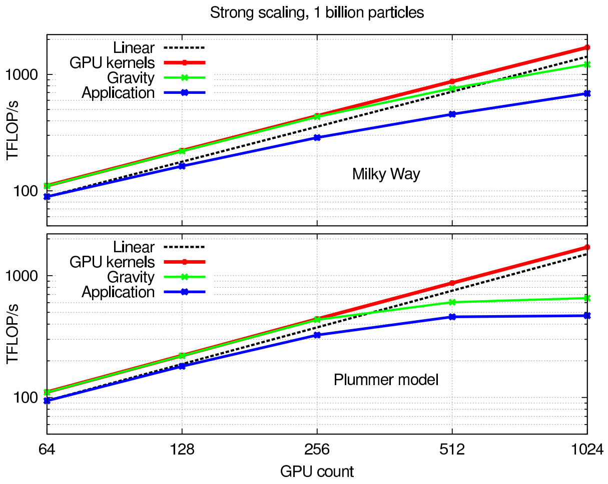
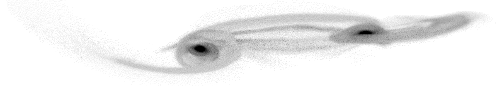


Figure 6.6: Strong scaling from 64 to 1024 nodes with 1 billion particles in total. The x- and y-axis show the number of GPUs used and the performance respectively. The top panel shows the results for a Milky Way model. The bottom panel shows the Plummer model, similar as used in the weak scaling simulations. The red solid lines show the performance of the tree-walk kernel running on the GPU. The green solid lines show the performance of the gravity step, including the time to communicate LET structures. The blue solid lines show the performance of the full application. The dashed black lines indicate linear scaling.



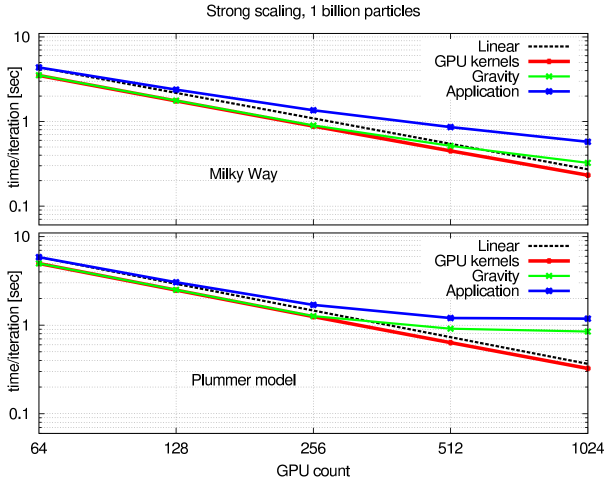


Figure 6.7: As Fig. 6.6, but showing the wall-clock time per iteration on the y-axis.

1 billion particles. The Milky Way initial conditions (top panel) continues to scale up to 1024 nodes, whereas the Plummer model (bottom panel) shows a decline. This difference originates from the size of the interaction list which for the Plummer model (7828 elements/particle) is nearly 50% larger than that of the Milky Way (5347 elements/particle). This, compounded by the need to import more data on 1024 than on 64 GPUs due to the increase of surface-to-volume ratio, explains the superior strong scaling properties of the Milky Way model. In particular, with as little as 1 million particles per GPU, the parallel efficiency of `Bonsai` is almost 50%, with an average time spent per iteration of just over half a second!

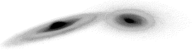
In both strong and weak scaling `Bonsai` is able to hide most of the communication time behind GPU computations. With about 4096 nodes, the communication time exceeds only slightly the GPU compute time. The domain decomposition, which has $\mathcal{O}(g)$ complexity with g the number of GPUs, is currently handled by a single process. Thus for some $g > g_{\rm crit}$ [6] the contributions from communication and serial parts of the code become noticeable.

6.6.3 Time-to-solution

We estimate the time-to-solution for a 66 billion particle simulation of the Milky Way Galaxy running on the Titan supercomputer. The strong scaling results indicate that the average time per iteration of the Milky Way model is shorter by a factor of ~ 0.67 compared to that of the Plummer model. Each step of the 66 billion particle simulation using a Plummer model running on 4096 nodes of Titan takes ~ 9.5 seconds. Such a step for the Milky Way Galaxy model would therefore last for about 6.4 seconds. With a soften-

---

[6] $g_{\rm crit} \approx 4096$ in the current version of the code



ing of 10 parsec, the minimal time step required for an accurate simulation is 0.075 Myr (this corresponds to the time that two particles pass each other within a softening length). This softening length was also used in our earlier calculations (Bédorf and Portegies Zwart 2013), and is considerably smaller than typically used in other Galaxy simulations (Dubinski and Chakrabarty 2009; Fujii et al. 2011; Athanassoula 2012; Grand et al. 2012a). If we desire to simulate the evolution of the Milky Way Galaxy for 10 Gyr, the simulation needs to make a total of 133 thousand time steps, each of which is 0.075 Myr. With an average of 6.4 seconds per time step when running on 4096 Titan nodes, the simulation will finish within 10 days.

6.6.4 Peak performance

We achieved a sustained application performance of 4.23 Pflops on a 66-billion-particle Plummer model with 4096 GPUs. When busy, the GPUs were processing at an aggregated rate of 7.09 Pflops. This translates to 1.73 Tflops per GPU and in excess of 1 Tflop for the overall application performance per node. The theoretical peak single precision performance of 4096 Titan nodes, excluding CPUs is 16.07 Pflops. The GPU kernels operate at 44% of this number, while the overall application achieves 26% of this peak performance.

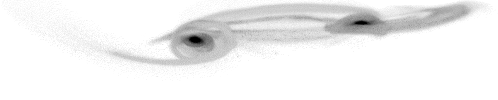
6.7 Discussion

One of the most important trends in parallel computing is the common availability of multi-threaded cores, which brought parallel computing to our desktop. One of the most important trends in many-core programming has been the availability of low-budget graphical accelerators, which brought high performance to our desktop. In the next step both trends are combined in a single machine, much in the same way as it is realized in Titan, together with a high-performance network. We expect that future large-scale scientific simulations will more frequently be carried out using such hybrid architectures.

With our attempt to perform a detailed simulation of the Milky Way Galaxy we demonstrate that hybrid architectures are ideally suited for scientific calculations. Our effort, however required us to completely redesign the N-body tree-code to make it operate efficiently on GPUs in parallel with CPUs. The accessibility of GPUs as part of a parallel multi-purpose platform would be enormously broadened if integrated compilers could detect the GPU-optimizable parts of the code and offload them to the attached multi-core hardware. Although this is unlikely to give the fine-tuned performance of `Bonsai`, it may provide acceptable speed-up compared to CPU only code, which probably is sufficient for most researchers. As it currently stands the performance of `Bonsai` is limited by the network and serial parts of the algorithm. With further improvements to these parts it is possible to let the algorithm scale even further on current generations of hardware. With future GPUs likely having more on-board memory[7] it will be easier to increase the amount of work done by the GPU relative to amount of CPU work and network communication for even further scaling.

---

[7] The NVIDIA Quadro K6000 has 12GB of on-board memory, twice that K20X as used in this work

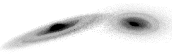


Our demonstration of the efficiency of running on a massively parallel heterogeneous architecture indicates that hybrid hardware can give excellent performance. Our achieved performance, the excellent scaling and the short time-to-solution has been realized by a fundamental redesign of the gravitational tree algorithm.

An application performance of 1 Tflop per node allows us to model a system with 16 million particles/node at a rate of about 6.4 seconds/step. This substantial speed is important for simulating a galaxy like the Milky Way, because the required simulation time is an order of magnitude longer than the dynamical time-scale of the system. For example, complete dynamical modeling of a Milky Way galaxy requires 10 Gyr of evolution, whereas its dynamical time scale is $\sim$ 250 Myr. With so little wall-clock time per step, it is now possible to model the history and future of the Milky Way within 10 day on a large GPU-equipped supercomputer, like Titan.

Acknowledgments

It is a pleasure to thank Arthur Trew, Richard Kenway and Jack Wells for arranging direct access via Director's time, Mark Harris, Stephen Jones and Simon Green from NVIDIA, for their assistance in optimizing `Bonsai` for the NVIDIA K20X, and Jun Makino for discussions. Part of our calculations were performed on Hector (Edinburgh), HA-PACS (University of Tsukuba), XC30 (National Astronomical Observatory of Japan) and Little Green Machine (Leiden University). This work was supported by the Netherlands Research Council NWO (grants #639.073.803 [VICI], #643.000.802 [JB], #614.061.608 [AMUSE] and #612.071.305 [LGM]), by the Netherlands Research School for Astronomy (NOVA), the Japan Society for the Promotion of Science (JSPS), MEXT HPCI strategic program and KAKENHI (under grand Number 24740115). This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.



7

Conclusions

The last few years have seen a huge increase in computational power in the form of special purpose hardware and new supercomputers. This is the direct result of the increasing amount of parallelism available in current day computer chips. However, in order to use this computational power, the user — or better, the developer — is forced to rethink the design and implementation of algorithms. Without taking advantage of the available multi-core technology there will hardly be any advantage of buying a new computer. We see this trend in the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU), but also for example in the mobile phone industry where quad-cores are the current day standard and octo-cores are slowly being introduced.

This thesis presents how we can benefit from the available processing power of these many-core chips, in our case GPUs, when performing astrophysical simulations. This can either be by implementing expensive, but accurate, algorithms such as direct N-body methods (see Chapter 2) or by taking it a step further and transforming the hierarchical Barnes-Hut tree-code method into a version that is suitable for many-core architectures (see Chapters 3, 4 and 6). The resulting simulation codes have a performance that is one to two orders of magnitude higher than previous versions. This allows for new kinds of science and wider parameter searches. For example, the work in Chapter 5 is the result of hundreds of simulations, while other works about the same topic usually do not perform more than a dozen simulations.

In this work we kept the direct N-body method and the tree-code method strictly separate, but in the future it might be beneficial to make use of both methods or make one of the methods part of a larger (existing) code. We will discuss this and more in the next paragraphs.

7.1 BRIDGE; Combining direct and hierarchical N-body methods

The difference between particle numbers used in collisional and collisionless methods has, because of their difference in scaling complexity, over the years only increased. Depending on the problem, scientists either choose for high precision direct N-body methods or for large particle numbers using approximation methods like the tree-code. Recently, however, methods have been introduced that try to combine the best of both worlds: the high

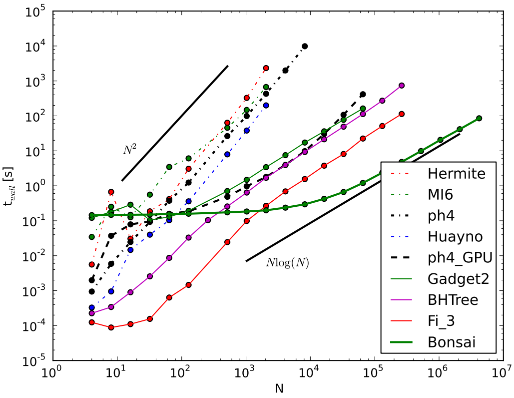


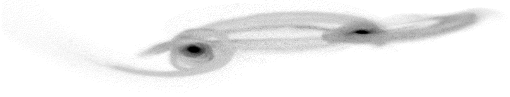
Figure 7.1: Performance comparison of a suite of N-body codes. These codes are included in the AMUSE software package. Visible are direct N-body codes that scale as $O(N^2)$ and hierarchical codes that scale as $O(Nlog(N))$. (Figure taken from (Portegies Zwart et al. 2013))

accuracy of direct methods and the speed of tree-codes. In the `BRIDGE` algorithm by Fujii et al. (2007) a direct N-body method is combined with a tree-code to integrate the evolution of star clusters (which requires direct N-body methods) embedded in their host galaxy (which requires an approximation method because of the large number of particles). This allows for detailed simulations in the area of interest while still being able to use large particle numbers. Since the method is based on two well known algorithms it is possible to use the methods presented in this thesis to accelerate `BRIDGE` with GPUs.

With simulation codes becoming more complex and containing more advanced features it becomes difficult to add new physics to existing codes without breaking other parts of the codes. This is a common problem in computational sciences and software development in general. Ideas often start off simple, but when something works you want to extend it, which complicates matters. In AMUSE (Portegies Zwart et al. (2009)) a different approach is taken. Codes that are written for different specific purposes are combined into one framework. This simplifies the development of the separate software products. The other advantage is that you can combine simulation codes that have support for GPUs with codes that do not and therefore still have the speed advantage of using GPUs. With AMUSE it is possible to use the same script using different simulation codes and thereby have the choice between speed, accuracy or available hardware. An example of this is shown in Fig. 7.1 where the execution speed of a set of N-body integration codes is demonstrated. The figure shows the results of 4 direct N-body codes (`Hermite, PhiGRAPE, Huayno and ph4`) and 3 tree-codes (`Gadget2, Octgrav, Bonsai`). Clearly visible is the difference in speed and scaling between the direct codes($\mathcal{O}(N^2)$ scaling) and the tree-codes ($\mathcal{O}(N\log N)$ scaling).

7.2 The future

With focus shifting to more complex methods and algorithms we see the advantage of the versatility of GPUs and the shift from fixed function methods in the early 90s (like the GRAPE) to programmable chips like GPUs. Even though Field Programmable Gate Arrays (FPGAs) have been around for decades, their programming is difficult and expensive,



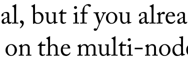
certainly compared to chips that are programmable by software. It is much easier to develop and acquire chips like GPUs, since they can be bought in many consumer computer stores. The availability and price makes the GPU one of the most attractive high performance computing devices currently available. It is of course still possible to develop faster chips that require less energy if you make them dedicated, but the development cost and specialized knowledge to build a chip that is competitive against the multi-billion dollar gaming industry is higher than a university research team can afford (Makino and Daisaka (2012)).

Also, simulation algorithms become more advanced and incorporate different techniques to overcome, for example, the painful $\mathcal{O}(N^2)$ scaling. An example of this is the `Pikachu` code by Iwasawa et al. (in preparation). In the `BRIDGE` method one has to indicate which particles will be integrated using the direct algorithm and which particles with the tree-code algorithm when the initial conditions are created. The `Pikachu` code improves on this by dynamically deciding which particles can be integrated using a tree-code and which need direct N-body integration.

Even though approximation methods (tree-codes, FMM and Particle Mesh) are much faster than direct N-body methods, they do not reach the same level of accuracy. With the increase in computational power, direct N-body methods will always be used for new simulations with increasing N either to compare to previous results (e.g. performed with approximate methods) or for new science. The same is valid for the methods used to improve the performance of direct N-body simulations (block time-steps, neighbour schemes, etc.; see Section 1.3). These all have an influence on the precision. Although the difference is smaller than the difference between direct methods and approximation based methods it still might be of influence, especially considering the chaotic nature of the N-body problem (Miller (1964); Goodman et al. (1993)). Therefore, with the increased compute performance we will not only perform simulations with larger N, but also much more detailed simulations with relatively small N to validate previously obtained results. Simulations of globular clusters using high precision shared time-step algorithms are still far out of reach, but one day we will have the computational power to perform exactly this kind of simulation.

The increasing availability of GPUs in supercomputers and in small dedicated GPU clusters shows the potential, increased usage and the faith of researchers in GPUs over the last few years. And especially with the installation of GPUs in ordinary desktop computers, as is done, for example, at the Leiden Observatory, this computational power is available at everyone's fingertips without having to request time on expensive supercomputers.

However, as we demonstrated in Chapter 6, supercomputers are not obsolete. For many scientific questions we can increase the problem size indefinitely and by doing so we will run out of the available resources of our desktop computer and small scale clusters. At that point we have to transition to supercomputers. To make this transition as easy as possible for the user it is fundamental that supercomputers represent architectures that are popular in desktop and cluster-sized machines. This allows researchers to develop and optimize their single and multi-node implementations on their local hardware and then try to scale this up to thousands of nodes. This scaling is not trivial, but if you already have an optimized single-node implementation you only have to focus on the multi-node aspect of your code.



8

Samenvatting

De afgelopen jaren hebben we een vlucht gezien in de ontwikkeling van parallelle processoren. Tot een aantal jaar geleden werden processoren sneller door de kloksnelheid te verhogen. Hierdoor hoefde je als gebruiker alleen maar een nieuwe processor te kopen om sneller te kunnen rekenen. Als we het zouden vergelijken met een woon-werk-treinreis zou dit betekenen dat het treinstel elke paar maanden zou worden vervangen door een nieuwer model met een hogere maximumsnelheid. Je stapt nog steeds op dezelfde plekken in en uit, volgt dezelfde route, alleen deze route leg je elke keer een beetje sneller af.

Tegenwoordig gaat dit verhaal niet meer op: de kloksnelheid wordt niet meer verhoogd. In plaats daarvan komen er meer rekeneenheden, vaak met een maximale kloksnelheid die lager is dan wat we in het verleden konden behalen. Doordat er echter meerdere problemen tegelijk kunnen worden uitgerekend kan er toch snelheidswinst behaald worden. Als we een vergelijk trekken met onze treinreis, dan zou de rit zelf dus niet sneller worden. In plaats daarvan wordt er nog een set treinrails neergelegd. Hierdoor gaan er in plaats van één trein, twee treinen tegelijkertijd over hetzelfde traject. Daardoor kunnen er meer reizigers worden vervoerd en worden de wachttijden op het station korter. We hoeven namelijk minder lang op de volgende trein te wachten als de eerste (en enige) trein vol is. Daarom is met parallellisatie tijdwinst te behalen, mits het aantal rekenproblemen (aantal reizigers) groot genoeg is. Maar wanneer we maar één reiziger of rekenprobleem hebben zal er juist geen tijdwinst behaald worden, omdat de maximumsnelheid lager ligt dan voorheen.

Nu hebben we de mazzel dat de sterrenkundige problemen vaak bestaan uit duizenden objecten waarvan we bijvoorbeeld de zwaartekracht willen uitrekenen. Het uitrekenen van de zwaartekracht is geen sinecure. Als we twee objecten hebben is de oplossing exact wiskundig te bepalen, bij drie of meer objecten is dit echter niet langer mogelijk. Bij drie of meer objecten moet de zwaartekracht numeriek bepaald worden. Dit doen we door van elk object uit te rekenen hoeveel zwaartekracht wordt uitgeoefend tussen dat object en elk ander object in het te onderzoeken systeem. Dit is het N-body probleem, en om het op te lossen moet je de uitkomst van $N \times N$ interacties uitrekenen.

De enorme hoeveelheid interacties zorgt ervoor dat dit een zeer rekenintensief probleem is. Echter, doordat we de interactie voor elk deeltje afzonderlijk kunnen uitrekenen kunnen we dit verspreiden over verschillende rekenkernen. Hierdoor geldt dat hoe meer rekenkernen we hebben, hoe sneller we de simulatie kunnen uitvoeren, mits we efficiënt gebruik maken van deze kernen. In de computers die ik voor het onderzoek in dit proefschrift gebruik zitten twee soorten processoren die geschikt zijn voor het uitvoeren van

rekenwerk: de centrale processor (CPU) en de grafische processor (GPU). Traditioneel werd de GPU alleen gebruikt voor rekenwerk in spelletjes, maar tegenwoordig kunnen we hem inzetten voor het helpen oplossen van wetenschappelijke problemen.

In dit proefschrift toon ik aan hoe we de GPU kunnen inzetten voor het uitvoeren van zwaartekracht simulaties. Hierbij maken we onderscheid tussen simulaties die gebruik maken van directe N-body methoden en simulaties die gebruik maken van hiërarchische methoden. De directe N-body methode is de nauwkeurigste, maar omdat deze methode schaalt als $O(N^2)$ vereist deze ook het meeste rekenwerk. Door dit vele rekenwerk kan deze methode wel efficiënt worden geïmplementeerd op de GPU, zoals ik aantoon in hoofdstuk 2. In de hiërarchische methode maken we gebruik van een boomstuctuur om onze data op te slaan. Deze methode is minder rekenintensief dan de directe N-body methode, maar daardoor ook minder nauwkeurig. Door de ingewikkelde datastructuren in deze methode is het een stuk ingewikkelder om gebruik te maken van de GPU. In hoofstukken 3 en 4 laat ik zien dat het mogelijk is om de GPU te gebruiken, mits we de implementatie van deze methode geheel herschrijven. Tijdens het implementeren houden we rekening met het feit dat de GPU duizenden rekenkernen heeft, daarom zullen we zoveel mogelijk onderdelen van het algoritme parallel uitvoeren. Vervolgens toon ik in hoofdstuk 5 wat er mogelijk is met de in de voorgaande hoofdstukken ontwikkelde simulatie code.

Tenslotte laat ik in het laatste hoofdstuk zien wat er nodig is om, op een efficiënte wijze, simulaties uit te voeren die miljarden deeltjes bevatten.

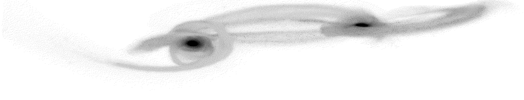
8.1 De hoofdstukken in dit proefschrift

8.1.1 Hoofdstuk 2

Dit hoofdstuk gaat over het versnellen van simulaties die gebruik maken van de directe N-body integratiemethode met behulp van de GPU. We introduceren de speciale softwarebibliotheek `Sapporo2`, welke het eenvoudig maakt om de GPU te gebruiken voor het uitvoeren van sterrenkundige simulaties. In het hoofdstuk bespreken we hoe de GPU werkt en hoe code het best te paralleliseren is. Vervolgens tonen we hoe je kunt bepalen welke configuraties de beste prestaties leveren. De uitkomsten van deze configuraties gebruiken we vervolgens om de prestaties van de bibliotheek te testen op verschillende GPUs. Hierdoor hebben we direct een beeld van welke GPU het meest geschikt is voor het simuleren van onze sterrenkundige problemen. Dit hangt namelijk sterk af van het aantal deeltjes en de benodigde precisie, dit alles wordt weergeven in de resultatensectie van het hoofdstuk.

8.1.2 Hoofdstuk 3

Door gebruik te maken van de GPU is het mogelijk om simulaties die de directe N-body integratiemethode methode gebruiken velen malen sneller uit te voeren dan met de CPU. Tevens is het mogelijk om, door gebruik te maken van bestaande methoden, de GPU te gebruiken voor hiërarchische algoritmen. Dit soort algoritmen vereist veel minder rekenkracht voor het uitrekenen van de zwaartekrachtinteracties die plaats vinden tussen verschillende deeltjes. Dit gaat wel ten koste van de nauwkeurigheid, maar dat is over het



algemeen geen probleem. De bestaande methoden zijn echter niet geoptimaliseerd voor de snelheid en mogelijkheden van de GPU. Daarom introduceren we in hoofdstuk 3 een nieuwe methode die ervoor zorgt dat de hoeveelheid communicatie tussen de grafische kaart en de CPU velen malen minder is dan in voorgaande implementaties. Hierdoor kunnen hiërarchische methoden veel efficiënter gebruik maken van de GPU dan voorheen mogelijk was. In dit hoofdstuk tonen we de eigenschappen van deze nieuwe methoden. Ook geven we aan waar er nog beperkingen zitten, en of dit ligt aan de methoden of aan de GPU.

8.1.3 Hoofdstuk 4

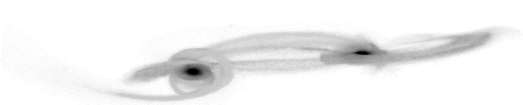
De beperkingen die we in het vorige hoofdstuk geïdentificeerd hebben worden in dit hoofdstuk onder handen genomen. We ontwikkelen een volledig nieuwe methode om in parallel hiërarchische datastructuren op te bouwen uit een set van datapunten. Doordat deze methode parallel is, werkt dit zeer efficiënt op de parallelle grafische kaart. Om verdere beperkingen te voorkomen hebben we tevens alle overige onderdelen van het hiërarchisch algoritme opnieuw geïmplementeerd en geschikt gemaakt voor de GPU. Hierdoor is de communicatie tussen de GPU en CPU bijna volledig geëlimineerd. Als gevolg van al deze verbeteringen is deze implementatie, die we `Bonsai` noemen, velen malen sneller dan de bestaande CPU én GPU implementaties. In dit hoofdstuk tonen we hoe de algoritmen schalen en hoe snel ze zijn op verschillende GPU architecturen. Dit doen we met synthetische datasets, maar ook met een simulatie van twee sterrenstelsels waarin zich zwarte gaten bevinden.

8.1.4 Hoofdstuk 5

De in het vorige hoofdstuk ontwikkelde `Bonsai` code gebruiken we in dit hoofdstuk voor een concreet probleem. We proberen antwoord te vinden op de vraag waarom we op foto's uit het verre verleden van het universum wel zeer compacte en zware sterrenstelsels zien, maar op foto's van het meer recente verleden er geen spoor te vinden is van dit soort sterrenstelsels. We onderzoeken in dit hoofdstuk hoe de eigenschappen van deze zware sterrenstelsels veranderen als ze samensmelten met kleinere en lichtere sterrenstelsels. We stellen daarbij de vraag of ze sneller groeien in volume dan in massa. Uit de simulaties blijkt dit inderdaad het geval te zijn. Als we een zwaar en compact sterrenstelsel laten samensmelten met tien sterrenstelsels die tien keer lichter zijn dan het zware stelsel wordt het nieuw gevormde sterrenstelsel twee keer zo zwaar, maar tevens zeven keer zo groot. De snelheid van de `Bonsai` code stelde ons in staat om vele simulaties uit te voeren met verschillende instellingen en configuraties. Door alle resultaten te onderzoeken komen we tot de volgende conclusie: als een zwaar en compact sterrenstelsel samensmelt met vijf tot tien kleinere sterrenstelsels ontstaat er een nieuw sterrenstelsel dat vergelijkbaar is met wat we observeren op foto's die het recente verleden van het universum tonen. Deze kleine stelsels moeten dan wel vijf tot tien keer lichter zijn dan het grote sterrenstelsel. Als dat het geval is verdubbelt het originele compacte sterrenstelsel in massa, maar wordt het tevens tussen de vijf en zeven keer groter.

8.1.5 Hoofdstuk 6

De simulatiesoftware `Bonsai` die we in hoofdstuk 4 hebben geïntroduceerd wordt in dit hoofdstuk uitgebreid. In de eerste versie was het niet mogelijk om meer dan één GPU te gebruiken. Hierdoor was het aantal deeltjes dat je kon simuleren beperkt. Door de uitbreidingen die we in dit hoofdstuk introduceren is het mogelijk om `Bonsai` op meerdere GPUs tegelijk te gebruiken. De methodes zijn gebaseerd op bestaande publicaties, maar worden in een unieke manier samengevoegd waardoor het mogelijk is de nieuwe implementatie te laten schalen naar duizenden grafische kaarten en miljarden deeltjes. Dit tonen we aan door de code uit te voeren op de duizenden GPUs die in de Titan supercomputer zitten. Dit is de snelste GPU supercomputer die op dit moment beschikbaar is. De mogelijkheid om miljarden deeltjes te simuleren, wat orders van grote meer is dan hiervoor, stelt ons in staat om gedetailleerdere modellen door te rekenen dan in het verleden mogelijk was. Hierdoor zijn we instaat simulaties te draaien die we kunnen vergelijken met de data die de nieuwste satellieten zullen produceren.



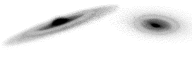
List of publications

Journal Papers

- R. G. Belleman, **J. Bédorf**, and S. F. Portegies Zwart. High performance direct gravitational N-body simulations on graphics processing units II: An implementation in CUDA. *New Astronomy*, 13:103–112, February 2008. doi10.1016/j.newast.2007.07.004.
- **J. Bédorf**, E. Gaburov, and S. Portegies Zwart. A sparse octree gravitational N-body code that runs entirely on the GPU processor. *Journal of Computational Physics*, 231:2825–2839, April 2012. doi10.1016/j.jcp.2011.12.024.
- **J. Bédorf** and S. Portegies Zwart. The effect of many minor mergers on the size growth of compact quiescent galaxies. *MNRAS*, 431:767–780, May 2013. doi10.1093/mnras/stt208.
- **J. Bédorf**, E. Gaburov, K. Nitadori, M.S. Fujii, T. Ishiyama and S. Portegies Zwart. How to simulate the Milky Way Galaxy on a star-by-star basis. *New Astronomy*, submitted.
- S. Portegies Zwart and **J. Bédorf**. Computational Gravitational Dynamics with Modern Numerical Accelerators. *Computer*, submitted.
- **J. Bédorf**, E. Gaburov and S. Portegies Zwart. Sapporo2: A versatile direct N-body library. *Computational Astrophysics and Cosmology*, submitted.

Peer-reviewed Conference Proceedings

- E. Gaburov, **J. Bédorf**, and S. Portegies Zwart. Gravitational Tree-Code on Graphics Processing Units: Implementation in CUDA. In *International Conference on Computational Science 2010*. Elsevier, 2010.
- **J. Bédorf** and S. Portegies Zwart. A pilgrimage to gravity on GPUs. *European Physical Journal Special Topics*, 210:201–216, August 2012. doi10.1140/epjst/e2012-1647-6.



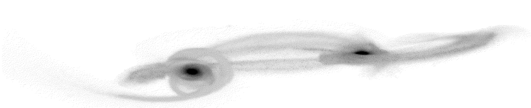
- **J. Bédorf**, E. Gaburov, M.S. Fujii, K. Nitadori, T. Ishiyama and S. Portegies Zwart. 24.77 Pflops on a Gravitational Tree-Code to Simulate the Milky Way Galaxy with 18600 GPUs. *SC'14 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ACM, 2014.

Conference Proceedings

- **J. Bédorf**, E. Gaburov, and S. Portegies Zwart. Bonsai: A GPU Tree-Code. In R. Capuzzo-Dolcetta, M. Limongi, and A. Tornambè, editors, *Advances in Computational Astrophysics: Methods, Tools, and Outcome*, volume 453 of *Astronomical Society of the Pacific Conference Series*, page 325, July 2012.

- **J. Bédorf** and S. Portegies Zwart. Parallel gravity: From embarrassingly parallel to hierarchical. In *Proceedings of the 2012 Workshop on High-Performance Computing for Astronomy Date*, Astro-HPC '12, pages 7–8, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1338-4.
  doi10.1145/2286976.2286980.

- J.A.C. van Toorenburg, N. Kijk in de Vegte, and **J. Bédorf**. On-line and off-line simulation of large motorway networks. In *Proceedings of 2nd International Conference on Models and Technologies for Intelligent Transportation Systems*, 2011.



Bibliography

S. J. Aarseth. Dynamical evolution of clusters of galaxies, I. *MNRAS*, 126:223–+, 1963.

S. J. Aarseth. From NBODY1 to NBODY6: The Growth of an Industry. *PASP*, 111:1333–1346, November 1999. doi: 10.1086/316455.

S. J. Aarseth. *Gravitational N-Body Simulations*. Gravitational N-Body Simulations, by Sverre J. Aarseth, pp. 430. ISBN 0521432723. Cambridge, UK: Cambridge University Press, November 2003., November 2003.

S. J. Aarseth. Mergers and ejections of black holes in globular clusters. *MNRAS*, 422:841–848, May 2012. doi: 10.1111/j.1365-2966.2012.20666.x.

S. J. Aarseth and D. C. Heggie. A 6000-Body Simulation with Primordial Binaries. In G. H. Smith & J. P. Brodie, editor, *The Globular Cluster-Galaxy Connection*, volume 48 of *Astronomical Society of the Pacific Conference Series*, page 701, January 1993.

A. Ahmad and L. Cohen. A numerical integration scheme for the N-body gravitational problem. *Journal of Computational Physics*, 12:389–402, 1973. doi: 10.1016/0021-9991(73)90160-5.

E. Athanassoula. Manifold-driven spirals in N-body barred galaxy simulations. *MNRAS*, 426: L46–L50, October 2012. doi: 10.1111/j.1745-3933.2012.01320.x.

E. Athanassoula, A. Bosma, J.-C. Lambert, and J. Makino. Performance and accuracy of a GRAPE-3 system for collisionless N-body simulations. *MNRAS*, 293:369–380, February 1998. doi: 10.1046/j.1365-8711.1998.01102.x.

E. Athanassoula, E. Fady, J. C. Lambert, and A. Bosma. Optimal softening for force calculations in collisionless N-body simulations. *MNRAS*, 314:475–488, May 2000. doi: 10.1046/j.1365-8711.2000.03316.x.

J. Baba, T. R. Saitoh, and K. Wada. Dynamics of Non-steady Spiral Arms in Disk Galaxies. *ApJ*, 763:46, January 2013. doi: 10.1088/0004-637X/763/1/46.

J. S. Bagla. TreePM: A Code for Cosmological N-Body Simulations. *Journal of Astrophysics and Astronomy*, 23:185–196, December 2002. doi: 10.1007/BF02702282.

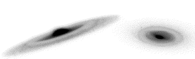
J. Barnes and P. Hut. A Hierarchical O(NlogN) Force-Calculation Algorithm. *Nature*, 324:446–449, December 1986.

J. E. Barnes. A Modified Tree Code Don't Laugh: It Runs. *Journal of Computational Physics*, 87: 161–+, March 1990.

J.E. Barnes. *Computational Astrophysics*. Springer-Verlag, Berlin, 1994.

J. Bédorf and S. Portegies Zwart. A pilgrimage to gravity on GPUs. *European Physical Journal Special Topics*, 210:201–216, August 2012. doi: 10.1140/epjst/e2012-1647-6.

J. Bédorf and S. Portegies Zwart. The effect of many minor mergers on the size growth of compact quiescent galaxies. *MNRAS*, 431:767–780, May 2013. doi: 10.1093/mnras/stt208.



J. Bédorf, E. Gaburov, and S. Portegies Zwart. A sparse octree gravitational N-body code that runs entirely on the GPU processor. *Journal of Computational Physics*, 231:2825–2839, April 2012. doi: 10.1016/j.jcp.2011.12.024.

R. G. Belleman, J. Bédorf, and S. F. Portegies Zwart. High performance direct gravitational N-body simulations on graphics processing units II: An implementation in CUDA. *New Astronomy*, 13: 103–112, February 2008. doi: 10.1016/j.newast.2007.07.004.

G. Besla, N. Kallivayalil, L. Hernquist, R. P. van der Marel, T. J. Cox, and D. Kereš. Simulations of the Magellanic Stream in a First Infall Scenario. *ApJ*, 721:L97–L101, October 2010. doi: 10.1088/2041-8205/721/2/L97.

R. Bezanson, P. G. van Dokkum, T. Tal, D. Marchesini, M. Kriek, M. Franx, and P. Coppi. The Relation Between Compact, Quiescent High-redshift Galaxies and Massive Nearby Elliptical Galaxies: Evidence for Hierarchical, Inside-Out Growth. *ApJ*, 697:1290–1298, June 2009. doi: 10.1088/0004-637X/697/2/1290.

Markus Billeter, Ola Olsson, and Ulf Assarsson. Efficient stream compaction on wide simd many-core architectures. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, pages 159–166, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-603-8. doi: http://doi.acm.org/10.1145/1572769.1572795.

P. Bode, J. P. Ostriker, and G. Xu. The Tree Particle-Mesh N-Body Gravity Solver. *ApJS*, 128: 561–569, June 2000. doi: 10.1086/313398.

I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for gpus: stream computing on graphics hardware. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 777–786, New York, NY, USA, 2004a. ACM. doi: http://doi.acm.org/10.1145/1186562.1015800.

Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for gpus: stream computing on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 777–786, New York, NY, USA, 2004b. ACM. doi: http://doi.acm.org/10.1145/1186562.1015800.

Martin Burtscher and Keshav Pingali. *GPU Computing Gems Emerald Edition*, chapter 6: An Efficient CUDA Implementation of the Tree-based Barnes Hut n-Body Algorithm, pages 75–92. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011. ISBN 0123849888, 9780123849885.

A. Cimatti, C. Nipoti, and P. Cassata. Fast evolving size of early-type galaxies at z > 2 and the role of dissipationless (dry) merging. *MNRAS*, 422:L62, May 2012. doi: 10.1111/j.1745-3933.2012.01237.x.

M.A. Clark, Pc La Plante, and L.J. Greenhill. Accelerating radio astronomy cross-correlation with graphics processing units. *Int. J. High Perform. Comput. Appl.*, 27(2):178–192, May 2013. ISSN 1094-3420. doi: 10.1177/1094342012444794. URL `http://dx.doi.org/10.1177/1094342012444794`.

E. Daddi, A. Renzini, N. Pirzkal, A. Cimatti, S. Malhotra, M. Stiavelli, C. Xu, A. Pasquali, J. E. Rhoads, M. Brusa, S. di Serego Alighieri, H. C. Ferguson, A. M. Koekemoer, L. A. Moustakas, N. Panagia, and R. A. Windhorst. Passively Evolving Early-Type Galaxies at 1.4 <˜ z <˜ 2.5 in the Hubble Ultra Deep Field. *ApJ*, 626:680–697, June 2005. doi: 10.1086/430104.

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*, chapter 5 and 16. Springer-Verlag, second edition, 2000.

W. Dehnen. Towards optimal softening in three-dimensional N-body codes - I. Minimizing the force error. *MNRAS*, 324:273–291, June 2001. doi: 10.1046/j.1365-8711.2001.04237.x.

W. Dehnen. A Hierarchical O(N) Force Calculation Algorithm. *Journal of Computational Physics*,

179:27–42, June 2002.

W. Dehnen and J. I. Read. N-body simulations of gravitational dynamics. *European Physical Journal Plus*, 126:55, May 2011. doi: 10.1140/epjp/i2011-11055-3.

S. Dindar, E. B. Ford, M. Juric, Y. I. Yeo, J. Gao, A. C. Boley, B. Nelson, and J. Peters. Swarm-NG: A CUDA library for Parallel n-body Integrations with focus on simulations of planetary systems. *New A*, 23:6–18, October 2013. doi: 10.1016/j.newast.2013.01.002.

E. D'Onghia, M. Vogelsberger, and L. Hernquist. Self-perpetuating Spiral Arms in Disk Galaxies. *ApJ*, 766:34, March 2013. doi: 10.1088/0004-637X/766/1/34.

E. N. Dorband, M. Hemsendorf, and D. Merritt. Systolic and hyper-systolic algorithms for the gravitational N-body problem, with an application to Brownian motion. *Journal of Computational Physics*, 185:484–511, March 2003. doi: 10.1016/S0021-9991(02)00067-0.

J. Dubinski. A parallel tree code. *New Astronomy*, 1:133–147, October 1996. doi: 10.1016/S1384-1076(96)00009-7.

J. Dubinski. Visualizing astrophysical N-body systems. *New Journal of Physics*, 10(12):125002, December 2008. doi: 10.1088/1367-2630/10/12/125002.

J. Dubinski and D. Chakrabarty. Warps and Bars from the External Tidal Torques of Tumbling Dark Halos. *ApJ*, 703:2068–2081, October 2009. doi: 10.1088/0004-637X/703/2/2068.

J. Dubinski, J. Kim, C. Park, and R. Humble. GOTPM: a parallel hybrid particle-mesh treecode. *New A*, 9:111–126, February 2004. doi: 10.1016/j.newast.2003.08.002.

J. Dubinski, I. Berentzen, and I. Shlosman. Anatomy of the Bar Instability in Cuspy Dark Matter Halos. *ApJ*, 697:293–310, May 2009. doi: 10.1088/0004-637X/697/1/293.

E. Elsen, M. Houston, V. Vishal, E. Darve, P. Hanrahan, and V. Pande. N-body simulation on gpus. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM. ISBN 0-7695-2700-0. doi: http://doi.acm.org/10.1145/1188455.1188649.

C. J. Fluke. Accelerating the Rate of Astronomical Discovery with GPU-Powered Clusters. In P. Ballester, D. Egret, and N. P. F. Lorente, editors, *Astronomical Data Analysis Software and Systems XXI*, volume 461 of *Astronomical Society of the Pacific Conference Series*, page 3, September 2012.

P. Fortin, E. Athanassoula, and J.-C. Lambert. Comparisons of different codes for galactic N-body simulations. *A&A*, 531:A120, July 2011. doi: 10.1051/0004-6361/201015933.

M. Franx, P. G. van Dokkum, N. M. F. Schreiber, S. Wuyts, I. Labbé, and S. Toft. Structure and Star Formation in Galaxies out to z = 3: Evidence for Surface Density Dependent Evolution and Upsizing. *ApJ*, 688:770–788, December 2008. doi: 10.1086/592431.

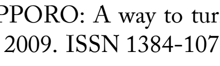
Daan Frenkel and B. Smit. *Understanding Molecular Simulation, Second Edition: From Algorithms to Applications (Computational Science Series, Vol 1)*. Academic Press, 2 edition, November 2001. ISBN 0122673514.

M. Fujii, M. Iwasawa, Y. Funato, and J. Makino. BRIDGE: A Direct-Tree Hybrid N-Body Algorithm for Fully Self-Consistent Simulations of Star Clusters and Their Parent Galaxies. *PASJ*, 59:1095–, December 2007.

M. S. Fujii, J. Baba, T. R. Saitoh, J. Makino, E. Kokubo, and K. Wada. The Dynamics of Spiral Arms in Pure Stellar Disks. *ApJ*, 730:109, April 2011. doi: 10.1088/0004-637X/730/2/109.

T. Fukushige, T. Ito, J. Makino, T. Ebisuzaki, D. Sugimoto, and M. Umemura. GRAPE-1A: Special-Purpose Computer for N-body Simulation with a Tree Code. *Publ. Astr. Soc. Japan*, 43: 841–858, December 1991.

Evghenii Gaburov, Stefan Harfst, and Simon Portegies Zwart. SAPPORO: A way to turn your graphics cards into a GRAPE-6. *New Astronomy*, 14(7):630 – 637, 2009. ISSN 1384-1076. doi: DOI:10.1016/j.newast.2009.03.002.



Evghenii Gaburov, Jeroen Bédorf, and Simon Portegies Zwart. Gravitational Tree-Code on Graphics Processing Units: Implementation in CUDA. In *International Conference on Computational Science 2010*. Elsevier, 2010.

J. Goodman, D. C. Heggie, and P. Hut. On the Exponential Instability of N-Body Systems. *ApJ*, 415:715, October 1993. doi: 10.1086/173196.

R. J. J. Grand, D. Kawata, and M. Cropper. The dynamics of stars around spiral arms. *MNRAS*, 421:1529–1538, April 2012a. doi: 10.1111/j.1365-2966.2012.20411.x.

R. J. J. Grand, D. Kawata, and M. Cropper. Dynamics of stars around spiral arms in an N-body/SPH simulated barred spiral galaxy. *MNRAS*, 426:167–180, October 2012b. doi: 10.1111/j.1365-2966.2012.21733.x.

R. J. J. Grand, D. Kawata, and M. Cropper. Spiral arm pitch angle and galactic shear rate in N-body simulations of disc galaxies. *A&A*, 553:A77, May 2013. doi: 10.1051/0004-6361/201321308.

A. Gualandris, S. Portegies Zwart, and A. Tirado-Ramos. Performance analysis of direct N-body algorithms for astrophysical simulations on distributed systems. *Parallel Computing*, 33(3):159 – 173, 2007. ISSN 0167-8191. doi: 10.1016/j.parco.2007.01.001.

Alessia Gualandris, Simon Portegies Zwart, and Alfredo Tirado-Ramos. Performance analysis of direct n-body algorithms for astrophysical simulations on distributed systems. *Parallel Comput.*, 33(3):159–173, 2007. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2007.01.001.

Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, November 1990.

T. Hamada and T. Iitaka. The Chamomile Scheme: An Optimized Algorithm for N-body simulations on Programmable Graphics Processing Units. *ArXiv Astrophysics e-prints*, March 2007.

Tsuyoshi Hamada and Keigo Nitadori. 190 tflops astrophysical n-body simulation on a cluster of gpus. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–9, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-1-4244-7559-9. doi: http://dx.doi.org/10.1109/SC.2010.1. URL `http://dx.doi.org/10.1109/SC.2010.1`.

Tsuyoshi Hamada, Tetsu Narumi, Rio Yokota, Kenji Yasuoka, Keigo Nitadori, and Makoto Taiji. 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009a. ACM. ISBN 978-1-60558-744-8. doi: http://doi.acm.org/10.1145/1654059.1654123.

Tsuyoshi Hamada, Tetsu Narumi, Rio Yokota, Kenji Yasuoka, Keigo Nitadori, and Makoto Taiji. 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 62:1–62:12, New York, NY, USA, 2009b. ACM. ISBN 978-1-60558-744-8. doi: 10.1145/1654059.1654123. URL `http://doi.acm.org/10.1145/1654059.1654123`.

Tsuyoshi Hamada, Keigo Nitadori, Khaled Benkrid, Yousuke Ohno, Gentaro Morimoto, Tomonari Masada, Yuichiro Shibata, Kiyoshi Oguri, and Makoto Taiji. A novel multiple-walk parallel algorithm for the barnes–hut treecode on gpus – towards cost effective, high performance n-body simulation. *Computer Science - Research and Development*, 24(1-2):21–31, 2009c. ISSN 1865-2034. doi: 10.1007/s00450-009-0089-1. URL `http://dx.doi.org/10.1007/s00450-009-0089-1`.

S. Harfst, A. Gualandris, D. Merritt, R. Spurzem, S. Portegies Zwart, and P. Berczik. Performance analysis of direct N-body algorithms on special-purpose supercomputers. *New Astronomy*, 12: 357–377, July 2007. doi: 10.1016/j.newast.2006.11.003.

A. H. Hassan, C. J. Fluke, D. G. Barnes, and V. A. Kilborn. Tera-scale astronomical data analysis and visualization. *MNRAS*, 429:2442–2455, March 2013. doi: 10.1093/mnras/sts513.

D. Heggie and P. Hut. *The Gravitational Million-Body Problem: A Multidisciplinary Approach to Star Cluster Dynamics*. Cambridge University Press, 2003. ISBN 9780521774864. URL `http://books.google.nl/books?id=dQH7NJRhCvMC`.

D. C. Heggie and R. D. Mathieu. Standardised Units and Time Scales. In P. Hut and S. L. W. McMillan, editors, *The Use of Supercomputers in Stellar Dynamics*, volume 267 of *Lecture Notes in Physics, Berlin Springer Verlag*, page 233, 1986. doi: 10.1007/BFb0116419.

P. Hertz and S. L. W. McMillan. Application of a massively parallel computer to the N-body problem. *Celestial Mechanics*, 45:77–80, March 1988. doi: 10.1007/BF01228981.

David Hilbert. Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891. doi: 10.1007/bf01199431. URL `http://dx.doi.org/10.1007/bf01199431`.

M. Hilz, T. Naab, J. P. Ostriker, J. Thomas, A. Burkert, and R. Jesseit. Relaxation and stripping - The evolution of sizes, dispersions and dark matter fractions in major and minor mergers of elliptical galaxies. *MNRAS*, 425:3119–3136, October 2012. doi: 10.1111/j.1365-2966.2012.21541.x.

M. Hilz, T. Naab, and J. P. Ostriker. How do minor mergers promote inside-out growth of ellipticals, transforming the size, density profile and dark matter fraction? *MNRAS*, 429:2924–2933, March 2013. doi: 10.1093/mnras/sts501.

R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. 1981.

F. Hohl and R. W. Hockney. A Computer Model of Disks of Stars. *Journal of Computational Physics*, 4:306, October 1969. doi: 10.1016/0021-9991(69)90002-3.

E. Holmberg. On the Clustering Tendencies among the Nebulae. II. a Study of Encounters Between Laboratory Models of Stellar Systems by a New Integration Procedure. *ApJ*, 94:385–+, November 1941. doi: 10.1086/144344.

P. F. Hopkins, L. Hernquist, T. J. Cox, D. Keres, and S. Wuyts. Dissipation and Extra Light in Galactic Nuclei. IV. Evolution in the Scaling Relations of Spheroids. *ApJ*, 691:1424–1458, February 2009. doi: 10.1088/0004-637X/691/2/1424.

P. F. Hopkins, D. Croton, K. Bundy, S. Khochfar, F. van den Bosch, R. S. Somerville, A. Wetzel, D. Keres, L. Hernquist, K. Stewart, J. D. Younger, S. Genel, and C.-P. Ma. Mergers in ΛCDM: Uncertainties in Theoretical Predictions and Interpretations of the Merger Rate. *ApJ*, 724:915–945, December 2010. doi: 10.1088/0004-637X/724/2/915.

J. R. Hurley and A. D. Mackey. N-body models of extended star clusters. *MNRAS*, 408:2353–2363, November 2010. doi: 10.1111/j.1365-2966.2010.17285.x.

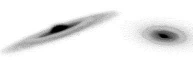
P. Hut. Dense stellar systems as laboratories for fundamental physics. *New A Rev.*, 54:163–172, March 2010. doi: 10.1016/j.newar.2010.09.009.

P. Hut, J. Makino, and S. McMillan. Building a better leapfrog. *ApJ*, 443:L93–L96, April 1995. doi: 10.1086/187844.

S. Inagaki. Post-collapse evolution of globular clusters with finite number of stars in the core. *PASJ*, 38:853–863, 1986.

T. Ishiyama, T. Fukushige, and J. Makino. GreeM: Massively Parallel TreePM Code for Large Cosmological N -body Simulations. *PASJ*, 61:1319–, December 2009.

Tomoaki Ishiyama, Keigo Nitadori, and Junichiro Makino. 4.45 pflops astrophysical n-body simulation on k computer: The gravitational trillion-body problem. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 5:1–5:10, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press. ISBN 978-1-4673-0804-5. URL `http://dl.acm.org/citation.cfm?id=2388996.2389003`.



T. Ito, J. Makino, T. Ebisuzaki, and D. Sugimoto. A special-purpose N-body machine GRAPE-1. *Computer Physics Communications*, 60:187–194, September 1990. doi: 10.1016/0010-4655(90)90003-J.

A. Kawai, T. Fukushige, J. Makino, and M. Taiji. GRAPE-5: A Special-Purpose Computer for N-Body Simulations. *PASJ*, 52:659–676, August 2000.

A. Kawai, J. Makino, and T. Ebisuzaki. Performance Analysis of High-Accuracy Tree Code Based on the Pseudoparticle Multipole Method. *ApJS*, 151:13–33, March 2004. doi: 10.1086/381391.

Atsushi Kawai, Toshiyuki Fukushige, and Junichiro Makino. $7.0/mflops astrophysical n-body simulation with treecode on grape-5. In *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, Supercomputing '99, New York, NY, USA, 1999. ACM. ISBN 1-58113-091-0. doi: 10.1145/331532.331598. URL `http://doi.acm.org/10.1145/331532.331598`.

Khronos Group Std. The OpenCL Specification Version 1.0 Rev.48 , 2010. URL `http://khronos.org/registry/cl/specs/opencl-1.0.48.pdf`.

Donald E. Knuth. *The Art of Computer Programming, Volume 3: (3rd ed.) Sorting and Searching*, chapter Sorting by Distribution, pages 168–179. Addison-Wesley, 1997. ISBN 0-201-89685-0.

K. Kuijken and J. Dubinski. Nearly Self-Consistent Disc / Bulge / Halo Models for Galaxies. *MNRAS*, 277:1341–+, December 1995.

C. Lacey and S. Cole. Merger rates in hierarchical models of galaxy formation. *MNRAS*, 262: 627–649, June 1993.

Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. Fast bvh construction on gpus. *Comput. Graph. Forum*, 28(2):375–384, 2009.

D. Lynden-Bell. Statistical mechanics of violent relaxation in stellar systems. *MNRAS*, 136:101, 1967.

J. Makino. Postcollapse Evolution of Globular Clusters. *ApJ*, 471:796, November 1996. doi: 10.1086/178007.

J. Makino. Direct Simulation of Dense Stellar Systems with GRAPE-6. In S. Deiters, B. Fuchs, A. Just, R. Spurzem, and R. Wielen, editors, *Dynamics of Star Clusters and the Milky Way*, volume 228 of *Astronomical Society of the Pacific Conference Series*, pages 87–+, 2001.

J. Makino. A Fast Parallel Treecode with GRAPE. *Publications of the Astronomical Society of Japan*, 56:521–531, June 2004.

J. Makino and S. J. Aarseth. On a Hermite integrator with Ahmad-Cohen scheme for gravitational many-body problems. *PASJ*, 44:141–151, April 1992.

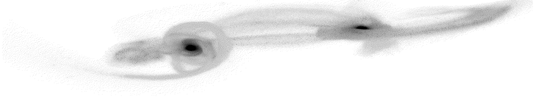
J. Makino and M. Taiji. *Scientific simulations with special-purpose computers : The GRAPE systems*. Scientific simulations with special-purpose computers : The GRAPE systems /by Junichiro Makino & Makoto Taiji. Chichester ; Toronto : John Wiley & Sons, c1998., 1998.

J. Makino, K. Hiraki, and M. Inaba. GRAPE-DR: 2-Pflops massively-parallel computer with 512-core, 512-Gflops processor chips for scientific computing. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 18:1–18:11, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-764-3. doi: http://doi.acm.org/10.1145/1362622.1362647.

Junichiro Makino and Hiroshi Daisaka. Grape-8: An accelerator for gravitational n-body simulation with 20.5gflops/w performance. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 104:1–104:10, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press. ISBN 978-1-4673-0804-5. URL `http://dl.acm.org/citation.cfm?id=2388996.2389137`.

Mark Harris (NVIDIA). Parallel Prefix Sum (Scan) with CUDA. *CUDA Scan Whitepaper*, 2009.

J. Martinez-Manso, R. Guzman, G. Barro, J. Cenarro, P. Perez-Gonzalez, P. Sanchez-Blazquez, I. Trujillo, M. Balcells, N. Cardiel, J. Gallego, A. Hempel, and M. Prieto. Velocity Dispersions



and Stellar Populations of the Most Compact and Massive Early-type Galaxies at Redshift ˜1. *ApJ*, 738:L22, September 2011. doi: 10.1088/2041-8205/738/2/L22.

S. L. W. McMillan. The Vectorization of Small-N Integrators. In P. Hut & S. L. W. McMillan, editor, *The Use of Supercomputers in Stellar Dynamics*, volume 267 of *Lecture Notes in Physics, Berlin Springer Verlag*, page 156, 1986. doi: 10.1007/BFb0116406.

S. L. W. McMillan and S. J. Aarseth. An O(N log N) integration scheme for collisional stellar systems. *ApJ*, 414:200–212, September 1993. doi: 10.1086/173068.

D. Merill and A. Grimshaw. Revisiting sorting for gpgpu stream architectures. Technical Report CS2010-03, Department of Computer Science, University of Virginia, February 2010.

D. Merritt. Optimal Smoothing for N-Body Codes. *AJ*, 111:2462, June 1996. doi: 10.1086/117980.

R. H. Miller. Irreversibility in Small Stellar Dynamical Systems. *ApJ*, 140:250, July 1964. doi: 10.1086/147911.

R. H. Miller and B. F. Smith. Galaxy collisions - A preliminary study. *ApJ*, 235:421–436, January 1980. doi: 10.1086/157646.

G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 19 1965.

G. M. Morton. A computer oriented geodetic data base; and a new technique in file sequencing. Technical report, Ottawa, Canada: IBM Ltd., 1966.

T. Naab, S. Khochfar, and A. Burkert. Properties of Early-Type, Dry Galaxy Mergers and the Origin of Massive Elliptical Galaxies. *ApJ*, 636:L81–L84, January 2006. doi: 10.1086/500205.

T. Naab, P. H. Johansson, and J. P. Ostriker. Minor Mergers and the Size Evolution of Elliptical Galaxies. *ApJ*, 699:L178–L182, July 2009. doi: 10.1088/0004-637X/699/2/L178.

N. Nakasato, G. Ogiya, Y. Miki, M. Mori, and K. Nomoto. Astrophysical Particle Simulations on Heterogeneous CPU-GPU Systems. *ArXiv e-prints*, June 2012.

J. F. Navarro, C. S. Frenk, and S. D. M. White. The Structure of Cold Dark Matter Halos. *ApJ*, 462:563, May 1996. doi: 10.1086/177173.

I. Newton. *Philosophiae naturalis principia mathematica*. J. Societatis Regiae ac Typis J. Streater, 1687. URL `http://books.google.nl/books?id=-dVKAQAAIAAJ`.

C. Nipoti, T. Treu, M. W. Auger, and A. S. Bolton. Can Dry Merging Explain the Size Evolution of Early-Type Galaxies? *ApJ*, 706:L86–L90, November 2009a. doi: 10.1088/0004-637X/706/1/L86.

C. Nipoti, T. Treu, and A. S. Bolton. Dry Mergers and the Formation of Early-Type Galaxies: Constraints from Lensing and Dynamics. *ApJ*, 703:1531–1544, October 2009b. doi: 10.1088/0004-637X/703/2/1531.

C. Nipoti, T. Treu, A. Leauthaud, K. Bundy, A. B. Newman, and M. W. Auger. Size and velocity-dispersion evolution of early-type galaxies in a Λ cold dark matter universe. *MNRAS*, 422:1714–1731, May 2012. doi: 10.1111/j.1365-2966.2012.20749.x.

K. Nitadori and S. J. Aarseth. Accelerating NBODY6 with graphics processing units. *MNRAS*, 424:545–552, July 2012. doi: 10.1111/j.1365-2966.2012.21227.x.

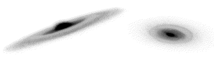
K. Nitadori and J. Makino. Sixth- and eighth-order Hermite integrator for N-body simulations. *New A*, 13:498–507, October 2008. doi: 10.1016/j.newast.2008.01.010.

K. Nitadori, J. Makino, and P. Hut. Performance tuning of N-body codes on modern microprocessors: I. Direct integration with a hermite scheme on x86_64 architecture. *New A*, 12:169–181, December 2006. doi: 10.1016/j.newast.2006.07.007.

NVIDIA. *NVIDIA CUDA Programming Guide 3.2*. 2010.

NVIDIA. *NVIDIA CUDA Programming Guide 5.5*. 2013a.

NVIDIA. *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110*. 2013b.



NVIDIA Corp. CUDA Programming manual. *CUDA Programming manual*, 1:200–212, September 2007. doi: 10.1086/173068.

L. Nyland, M. Harris, and J. Prins. The rapid evaluation of potential fields using programmable graphics hardware, 2004.

L. Nyland, M. Harris, and J. Prins. Fast N-body simulation with CUDA. *GPU Gems*, 3:677–695, 2007.

T. Oogi and A. Habe. Dry minor mergers and size evolution of high-z compact massive early-type galaxies. *MNRAS*, 428:641–657, January 2013. doi: 10.1093/mnras/sts047.

L. Oser, J. P. Ostriker, T. Naab, P. H. Johansson, and A. Burkert. The Two Phases of Galaxy Formation. *ApJ*, 725:2312–2323, December 2010. doi: 10.1088/0004-637X/725/2/2312.

L. Oser, T. Naab, J. P. Ostriker, and P. H. Johansson. The Cosmological Size and Velocity Dispersion Evolution of Massive Early-type Galaxies. *ApJ*, 744:63, January 2012. doi: 10.1088/0004-637X/744/1/63.

J. Pantaleoni and D. Luebke. Hlbvh: hierarchical lbvh construction for real-time ray tracing of dynamic geometry. In *Proceedings of the Conference on High Performance Graphics*, HPG '10, pages 87–95, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

P. J. E. Peebles. *The large-scale structure of the universe*. 1980.

H. C. Plummer. On the problem of distribution in globular star clusters. *MNRAS*, 71:460–470, March 1911.

H. C. Plummer. The distribution of stars in globular clusters. *MNRAS*, 76:107–121, December 1915.

S. Portegies Zwart and T. Boekholt. On the minimal accuracy required for simulating self-gravitating systems by means of direct N-body methods. *ArXiv e-prints*, February 2014.

S. Portegies Zwart, S. McMillan, D. Groen, A. Gualandris, M. Sipior, and W. Vermin. A parallel gravitational N-body kernel. *New A*, 13:285–295, July 2008. doi: 10.1016/j.newast.2007.11.002.

S. Portegies Zwart, S. McMillan, S. Harfst, D. Groen, M. Fujii, B. Ó. Nualláin, E. Glebbeek, D. Heggie, J. Lombardi, P. Hut, V. Angelou, S. Banerjee, H. Belkus, T. Fragos, J. Fregeau, E. Gaburov, R. Izzard, M. Jurić, S. Justham, A. Sottoriva, P. Teuben, J. van Bever, O. Yaron, and M. Zemp. A multiphysics and multiscale software environment for modeling astrophysical systems. *New Astronomy*, 14:369–378, May 2009. doi: 10.1016/j.newast.2008.10.006.

S. Portegies Zwart, T. Ishiyama, D. Groen, K. Nitadori, J. Makino, C. de Laat, S. McMillan, K. Hiraki, S. Harfst, and P. Grosso. Simulating the universe on an intercontinental grid of supercomputers. *IEEE Computer, v.43, No.8, p.63-70*, 43:63–70, October 2010. doi: 10.1109/MC.2009.419.

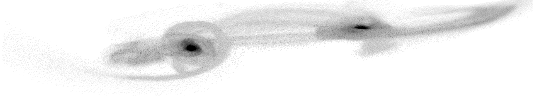
S. F. Portegies Zwart, S. L. W. McMillan, P. Hut, and J. Makino. Star cluster ecology - IV. Dissection of an open star cluster: photometry. *MNRAS*, 321:199–226, February 2001.

S. F. Portegies Zwart, R. G. Belleman, and P. M. Geldof. High-performance direct gravitational N-body simulations on graphics processing units. *New Astronomy*, 12:641–650, November 2007. doi: 10.1016/j.newast.2007.05.004.

S. F. Portegies Zwart, S. L. W. McMillan, A. van Elteren, F. I. Pelupessy, and N. de Vries. Multiphysics simulations using a hierarchical interchangeable software interface. *Computer Physics Communications*, 184:456–468, March 2013. doi: 10.1016/j.cpc.2012.09.024.

Rajeev Raman and David Stephen Wise. Converting to and from dilated integers. *IEEE Trans. Comput.*, 57:567–573, April 2008. ISSN 0018-9340. doi: 10.1109/TC.2007.70814. URL `http://dl.acm.org/citation.cfm?id=1345867.1345918`.

J. K. Salmon and M. S. Warren. Skeletons from the treecode closet. *Journal of Computational Physics*, 111:136–155, March 1994. doi: 10.1006/jcph.1994.1050.



Nadathur Satish, Mark Harris, and Michael Garland. Designing efficient sorting algorithms for manycore gpus. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3751-1. doi: http://dx.doi.org/10.1109/IPDPS.2009.5161005.

J. A. Sellwood. Relaxation in N-body Simulations of Disk Galaxies. *ApJ*, 769:L24, June 2013. doi: 10.1088/2041-8205/769/2/L24.

S. Sengupta, M. Harris, and M. Garland. Efficient parallel scan algorithms for gpus. Technical Report NVR-2008-003, NVIDIA, December 2008.

V. Springel. The cosmological simulation code GADGET-2. *MNRAS*, 364:1105–1134, December 2005. doi: 10.1111/j.1365-2966.2005.09655.x.

V. Springel, N. Yoshida, and S. D. M. White. GADGET: a code for collisionless and gasdynamical cosmological simulations. *New Astronomy*, 6:79–117, April 2001. doi: 10.1016/S1384-1076(01) 00042-2.

R. Spurzem and S. J. Aarseth. Direct collisional simulation of 10000 particles past core collapse. *MNRAS*, 282:19, September 1996.

R. Spurzem, P. Berczik, I. Berentzen, D. Merritt, N. Nakasato, H. M. Adorf, T. Brüsemeister, P. Schwekendiek, J. Steinacker, J. Wambsganß, G. M. Martinez, G. Lienhart, A. Kugel, R. Männer, A. Burkert, T. Naab, H. Vasquez, and M. Wetzstein. From Newton to Einstein N-body dynamics in galactic nuclei and SPH using new special hardware and astrogrid-D. *Journal of Physics Conference Series*, 78(1):012071, July 2007. doi: 10.1088/1742-6596/78/1/012071.

R. Spurzem, P. Berczik, I. Berentzen, K. Nitadori, T. Hamada, G. Marcus, A. Kugel, R. Männer, J. Fiestas, R. Banerjee, and R. Klessen. Astrophysical particle simulations with large custom gpu clusters on three continents. *Computer Science Research and Development*, 26(3-4):145–151, 2011. URL `http://www.springerlink.com/index/10.1007/s00450-011-0173-1`.

J. G. Stadel. *Cosmological N-body simulations and their analysis*. PhD thesis, AA(UNIVERSITY OF WASHINGTON), 2001.

D. Szomoru, M. Franx, and P. G. van Dokkum. Sizes and Surface Brightness Profiles of Quiescent Galaxies at z ˜ 2. *ApJ*, 749:121, April 2012. doi: 10.1088/0004-637X/749/2/121.

A. Tanikawa, K. Yoshikawa, T. Okamoto, and K. Nitadori. N-body simulation for self-gravitating collisional systems with a new SIMD instruction set extension to the x86 architecture, Advanced Vector eXtensions. *New A*, 17:82–92, February 2012. doi: 10.1016/j.newast.2011.07.001.

A. Tanikawa, K. Yoshikawa, K. Nitadori, and T. Okamoto. Phantom-GRAPE: Numerical software library to accelerate collisionless N-body simulation with SIMD instruction set on x86 architecture. *New A*, 19:74–88, February 2013. doi: 10.1016/j.newast.2012.08.009.

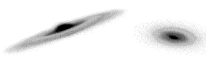
E. N. Taylor, M. Franx, K. Glazebrook, J. Brinchmann, A. van der Wel, and P. G. van Dokkum. On the Dearth of Compact, Massive, Red Sequence Galaxies in the Local Universe. *ApJ*, 720: 723–741, September 2010. doi: 10.1088/0004-637X/720/1/723.

E. Terlevich. N-Body Simulations of Open Clusters. In J. E. Hesser, editor, *Star Clusters*, volume 85 of *IAU Symposium*, page 165, 1980.

P. Teuben. The Stellar Dynamics Toolbox NEMO. In *Astronomical Data Analysis Software and Systems IV*, volume 77 of *Astronomical Society of the Pacific Conference Series*, pages 398–+, 1995.

S. Toft, P. van Dokkum, M. Franx, I. Labbe, N. M. Förster Schreiber, S. Wuyts, T. Webb, G. Rudnick, A. Zirm, M. Kriek, P. van der Werf, J. P. Blakeslee, G. Illingworth, H.-W. Rix, C. Papovich, and A. Moorwood. Hubble Space Telescope and Spitzer Imaging of Red and Blue Galaxies at z ˜ 2.5: A Correlation between Size and Star Formation Activity from Compact Quiescent Galaxies to Extended Star-forming Galaxies. *ApJ*, 671:285–302, December 2007. doi: 10.1086/521810.

I. Trujillo, N. M. Förster Schreiber, G. Rudnick, M. Barden, M. Franx, H.-W. Rix, J. A. R. Caldwell,



D. H. McIntosh, S. Toft, B. Häussler, A. Zirm, P. G. van Dokkum, I. Labbé, A. Moorwood, H. Röttgering, and et al. The Size Evolution of Galaxies since z˜3: Combining SDSS, GEMS, and FIRES. *ApJ*, 650:18–41, October 2006. doi: 10.1086/506464.

I. Trujillo, A. J. Cenarro, A. de Lorenzo-Cáceres, A. Vazdekis, I. G. de la Rosa, and A. Cava. Superdense Massive Galaxies in the Nearby Universe. *ApJ*, 692:L118–L122, February 2009. doi: 10.1088/0004-637X/692/2/L118.

I. Trujillo, I. Ferreras, and I. G. de La Rosa. Dissecting the size evolution of elliptical galaxies since z 1: puffing-up versus minor-merging scenarios. *MNRAS*, 415:3903–3913, August 2011. doi: 10.1111/j.1365-2966.2011.19017.x.

I. Trujillo, E. R. Carrasco, and A. Ferré-Mateu. Ultra-deep Sub-kiloparsec View of nearby Massive Compact Galaxies. *ApJ*, 751:45, May 2012. doi: 10.1088/0004-637X/751/1/45.

T. Valentinuzzi, B. M. Poggianti, R. P. Saglia, A. Aragón-Salamanca, L. Simard, P. Sánchez-Blázquez, M. D'onofrio, A. Cava, W. J. Couch, J. Fritz, A. Moretti, and B. Vulcani. Superdense Massive Galaxies in the ESO Distant Cluster Survey (EDisCS). *ApJ*, 721:L19–L23, September 2010. doi: 10.1088/2041-8205/721/1/L19.

T. S. van Albada. Numerical integrations of the N-body problem. *Bull. Astron. Inst. Netherlands*, 19: 479–+, January 1968.

J. van de Sande, M. Kriek, M. Franx, P. G. van Dokkum, R. Bezanson, K. E. Whitaker, G. Brammer, I. Labbé, P. J. Groot, and L. Kaper. The Stellar Velocity Dispersion of a Compact Massive Galaxy at z = 1.80 Using X-Shooter: Confirmation of the Evolution in the Mass-Size and Mass-Dispersion Relations. *ApJ*, 736:L9, July 2011. doi: 10.1088/2041-8205/736/1/L9.

A. van der Wel, B. P. Holden, A. W. Zirm, M. Franx, A. Rettura, G. D. Illingworth, and H. C. Ford. Recent Structural Evolution of Early-Type Galaxies: Size Growth from z = 1 to z = 0. *ApJ*, 688:48–58, November 2008. doi: 10.1086/592267.

P. G. van Dokkum, M. Franx, M. Kriek, B. Holden, G. D. Illingworth, D. Magee, R. Bouwens, D. Marchesini, R. Quadri, G. Rudnick, E. N. Taylor, and S. Toft. Confirmation of the Remarkable Compactness of Massive Quiescent Galaxies at z ˜ 2.3: Early-Type Galaxies Did not Form in a Simple Monolithic Collapse. *ApJ*, 677:L5–L8, April 2008. doi: 10.1086/587874.

P. G. van Dokkum, M. Kriek, and M. Franx. A high stellar velocity dispersion for a compact massive galaxy at redshift z = 2.186. *Nature*, 460:717–719, August 2009. doi: 10.1038/nature08220.

P. G. van Dokkum, K. E. Whitaker, G. Brammer, M. Franx, M. Kriek, I. Labbé, D. Marchesini, R. Quadri, R. Bezanson, G. D. Illingworth, A. Muzzin, G. Rudnick, T. Tal, and D. Wake. The Growth of Massive Galaxies Since z = 2. *ApJ*, 709:1018–1041, February 2010. doi: 10.1088/0004-637X/709/2/1018.

S. von Hoerner. Die numerische Integration des N-Körper-Problemes für Sternhaufen. I. *ZAp*, 50: 184–214, 1960.

M. S. Warren and J. K. Salmon. FOREST: A Parallel Treecode for Gravitational N-Body Simulations with up to 20 Million Particles. In *Bulletin of the American Astronomical Society*, volume 23 of *Bulletin of the American Astronomical Society*, page 1345, September 1991.

M. S. Warren and J. K. Salmon. Astrophysical n-body simulations using hierarchical tree data structures. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, Supercomputing '92, pages 570–576, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. ISBN 0-8186-2630-5. URL `http://dl.acm.org/citation.cfm?id=147877.148090`.

M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 12–21, New York, NY, USA, 1993. ACM. ISBN 0-8186-4340-4. doi: http://doi.acm.org/10.1145/169627.169640.

Michael S. Warren, Timothy C. Germann, Peter S. Lomdahl, David M. Beazley, and John K.

Salmon. Avalon: An alpha/linux cluster achieves 10 gflops for $15k. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, Supercomputing '98, pages 1–11, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-89791-984-X. URL `http://dl.acm.org/citation.cfm?id=509058.509130`.

M. S. Warren et al. Parallel supercomputing with commodity components. In H. R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, pages 1372–1381, 1997.

L. M. Widrow and J. Dubinski. Equilibrium Disk-Bulge-Halo Models for the Milky Way and Andromeda Galaxies. *ApJ*, 631:838–855, October 2005. doi: 10.1086/432710.

L. M. Widrow, B. Pym, and J. Dubinski. Dynamical Blueprints for Galaxies. *ApJ*, 679:1239–1259, June 2008. doi: 10.1086/587636.

Mathias Winkel, Robert Speck, Helge Hübner, Lukas Arnold, Rolf Krause, and Paul Gibbon. A massively parallel, multi-disciplinary barnes–hut tree code for extreme-scale n-body simulations. *Computer Physics Communications*, 183(4):880 – 889, 2012. ISSN 0010-4655. doi: http://dx.doi.org/10.1016/j.cpc.2011.12.013. URL `http://www.sciencedirect.com/science/article/pii/S0010465511004012`.

G. Xu. A New Parallel N-Body Gravity Solver: TPM. *ApJS*, 98:355, May 1995. doi: 10.1086/192166.

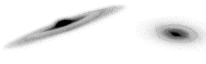
R. Yokota, J. P. Bardhan, M. G. Knepley, L. A. Barba, and T. Hamada. Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns. *Computer Physics Communications*, 182:1272–1283, June 2011. doi: 10.1016/j.cpc.2011.02.013.

Rio Yokota and Lorena A Barba. *GPU Computing Gems Emerald Edition*, chapter 9: Treecode and fast multipole method for N-body simulation with CUDA, pages 113–132. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011. ISBN 0123849888, 9780123849885.

Rio Yokota and Lorena A. Barba. Hierarchical n-body simulations with autotuning for heterogeneous systems. *Computing in Science and Engineering*, 14(3):30–39, 2012. ISSN 1521-9615. doi: http://doi.ieeecomputersociety.org/10.1109/MCSE.2012.1.

K. Yoshikawa and T. Fukushige. PPPM and TreePM Methods on GRAPE Systems for Cosmological N-Body Simulations. *PASJ*, 57:849–860, December 2005.

K. Zhou, M. Gong, X. Huang, and B. Guo. Highly parallel surface reconstruction. Technical Report MSR-TR-2008-53, Microsoft Research, April 2008.



Curriculum Vitae

Ik ben geboren op 1 mei 1984 te Alkmaar en begon mijn opleiding in 1988 op de Kleine en Grote Beer te Heerhugowaard. Om vervolgens in 1996 te beginnen aan het VWO op O.S.G Huygenwaard, ook in Heerhugowaard. In mijn vierde jaar van het VWO begon ik aan het “Economie & Maatschappij” profiel dat behoorde bij het pas ingevoerde “tweede fase” studie systeem. Aan het eind van het vierde jaar kwam ik er achter dat ik informatica interessant vond en heb toen Wiskunde A vervangen door Wiskunde B zodat ik een universitaire informatica opleiding kon volgen.

In 2002 heb ik mijn VWO opleiding afgerond en begon ik aan de opleiding “Informatica” aan de Universiteit van Amsterdam. Na het afronden van mijn bacheloronderzoek ben ik verder gegaan met de master opleiding in “Grid Computing” met als specialisatie “Computational Science”. Voor mijn masterscriptie heb ik gewerkt aan directe N-body simulaties op de grafische kaart onder leiding van Robert Belleman en Simon Portegies Zwart. Ik heb deze opleiding in november 2007 (cum laude) afgerond.

Vervolgens ben ik fulltime gaan werken voor Transpute B.V. te Amersfoort. Door het verkrijgen van een IsFast NWO grant eind 2008 ben ik in 2009 begonnen aan mijn promotieonderzoek binnen de computationale sterrenkunde groep van Simon Portegies Zwart aan de Universiteit van Amsterdam. Na twee maanden is de hele groep verhuisd naar de Sterrewacht Leiden waar ik het onderzoek heb voortgezet. In al die tijd heb 4 dagen per week aan mijn promotie onderzoek gewerkt en ben ik 1 dag per week blijven werken voor Transpute.

Ik heb de resultaten van mijn onderzoek gepresenteerd op conferenties in Nederland, Duitsland, Zweden, Italië en de Verenigde Staten. Tevens heb ik eind 2013 drie maanden stage gelopen bij NVIDIA in Californië, VS.

Acknowledgements

Met de meeste projecten sta je er zelden alleen voor en voor het maken van mijn proefschrift was dat niet anders, daarom gebruik ik graag de laatste pagina's om de mensen om mij heen te bedanken voor de hulp en steun die ze mij de afgelopen jaren gegeven hebben.

Allereerst wil ik mijn promotor Simon bedanken voor de mogelijkheid om dit onderzoek te kunnen uitvoeren en al het advies, steun en geboden mogelijkheden door de jaren heen. And directly connected to that I would like to thank all the current and former members of the "Computational Astrophysics" group for forming such a diverse group of interests, expertise and entertainment. Some of them I would like to thank in particular. Derek, bedankt voor alle (informatica) discussies die we gevoerd hebben in dit bastion van de sterrenkunde. Evghenii thanks for your ingenious insights in parallel computing and amazing conference trips. Arjen, Nathan and Inti bedankt voor jullie gezamenlijke kennis van algorithmen en software ontwikkeling waar ik in de afgelopen jaren dat we het kantoor deelden veelvuldig gebruik van maakte.

Furthermore I would like to thank the administrative staff that keeps the Sterrewacht up and running, thanks for all the help Alexandra, Anita, David, Debbie, Els, Erik, Evelijn, Jan, Liesbeth and Niels. And of course all my (former) collegues and friends here at the Sterrewacht that make it such a dynamic work environment and who made it worthwhile to make the long daily commute to Leiden. Some of them I would like to thank in particular. Bernadetta (for all the boardgame parties), Daniel R. & Carmen (for teaching me (some) Spanish, yo te estoy viendo!), Giorgia (for the wonderful tiramisu), Markus & Eva & Lars (for all the amazing barbecues), Michiko (for the great sushi), Stefania & Rasmus, Steven, Thibaut & Emilie & Samuel, Daniel H. and Yuri (for always offering a good time at parties and the many coffee and tea breaks).

Dan & Alex, thanks for all the great moments that we enjoyed in the office, during conferences and outside the working zone and most of all thanks for agreeing to be my paranymphs. Sam & Ellie & Jonas, Dan couldn't have wished for a better family, thanks for all the dinners, TV-show and Super Bowl parties that I've had the pleasure of being a part of at your place.

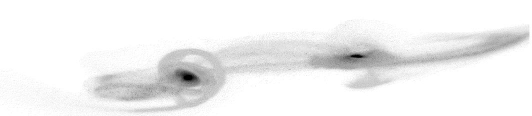
Naast mijn tijd op de universiteit was ik altijd 1 dag in de week aan het werk bij Transpute; Albert, Annemiek, Bart, Cees, Jaap en Natascha, bedankt voor de geweldige jaren en dat jullie altijd konden omgaan met mijn rare werkrooster.

During my three months at NVIDIA in California I met many people, a few I would

like to thank in particular for making my internship such a great experience. Thank you Sarah, Cyril, Justin, Simon, Paulius, Steve, Nicolai and Peng it was great working with you. Thanks to Mark Harris, Simon Green (both NVIDIA) and Stephen Jones (SpaceX) for their assistance in improving Bonsai's performance and visualization engine.

Over the years I met many amazing people in and outside the university. Constanze, thanks for your insights into astronomy, sushi, classical music and your deep knowledge of the Berlin public transport system. It was a great pleasure working with you! Sanne, bedankt voor je hulp met de omslag en het Nederlands. Met jou praten was altijd een welkome afleiding van de academische wereld en gaf mij altijd weer een andere en vooral betere kijk op de wereld.

Als laatste wil ik mijn familie bedanken welke mij de afgelopen jaren altijd onvoorwaardelijk gesteund heeft. Oma, Patrick, Natasja, Chantal, Marvin, Naomi, Lana, Smokey jullie stonden altijd klaar om mij te helpen en advies te geven. Papa en mama, bedankt dat jullie er altijd voor mij zijn.



Fewer than two dozen GPUs have died during the creation of this thesis.