Hybrid Systems for N-body Simulations

Spinnato, P.F.

Publication date
2003
Document Version
Final published version

Hybrid Systems for *N*-body Simulations

Piero Spinnato

Hybrid Systems for N-body Simulations

Piero Spinnato

Hybrid Systems for N-body Simulations

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. mr. P.F. van der Heijden
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit
op dinsdag 23 september 2003, te 12.00 uur

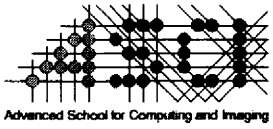
door

Pietro Fedele Spinnato

geboren te Sant'Agata di Militello (Italië)

PROMOTIE COMMISSIE

Promotor:	prof. dr. P.M.A. Sloot
Co-promotor:	dr. G.D. van Albada
Overige leden:	prof. drs. M. Boasson prof. dr. A.V. Bogdanov dr. ir. A.J.C. van Gemund prof. dr. E.P.J. van den Heuvel dr. S.F. Portegies Zwart
Faculteit:	Faculteit der Natuurwetenschappen, Wiskunde en Informatica



This work was carried out in the ASCI graduate school.
ASCI dissertation series number 87

ISBN 905-776-1092

Printed at Rotooffset Paganella, Trento

Detail of a picture taken in Leiden, at the corner of Pelikaanstraat with Oude Rijn.

Contents

Acknowledgments

It is with some degree of concern that I begin to write these acknoledgments, since I'm aware of the fact that this will be the first (and maybe the only) part of this dissertation that people will read. For this reason, I was tempted to put the entire text of the thesis under this section. This would have allowed me to break the world record for the longest acknowledgments ever, but it would have also induced the opposite effect, i.e., decreasing dramatically the number of my readers. Anyway, let's stop beating about the bush (sooner or later, it comes time for bushes to be beaten), and come to the point.

First of all, I'd like to thank Peter Sloot, my supervisor, for involving me in an exciting research field, at the crossroads of Algorithmics, Astrophysics, and Computer Science. The direction taken by my research owes much to his supervision. I'm also grateful to him for having involved me in teaching and refereeing activities. It was very stimulating and instructive to be an assistant in various courses, and also to serve as a referee for scientific journals and conferences where Peter occupies a steering role.

My thanks also go to Dick van Albada, my co-supervisor. His care for precision and rigorous attention to details influenced not only the research results, but also the final shape of this dissertation, which owes a great deal to his scrutinous reading of my drafts. I'm indebted to him and his family for their hospitality during my first days in the Netherlands.

I spent most of the last year and a half of my Ph.D. working with Simon Portegies Zwart. It has been a very enjoyable and fruitful collaboration, which resulted in the work presented in chapter 5 of this thesis. His enthusiasm and exuberance has always made it a pleasure to work with him; his inexhaustible energy has made it almost an ordeal! I also enjoyed helping him with the Stochastic Simulation 2003 course; our challenge of solving the Travelling Salesman Problem faster is still open. I'm sure my treecode solver will beat his Monte-Carlo one!

I gratefully acknowledge Ed van den Heuvel for having provided the funding that supported the last six months of my Ph.D.

I thank the members of my Ph.D. committee for agreeing to read my dissertation, and

my co-authors for granting me permission to include material from our joint papers in this thesis.

I'm grateful to Jun Makino from the University of Tokyo for having provided the two GRAPE-4 boards that I used for my performance analysis work, and the ASCI for the availability of the DAS cluster at the University of Asterdam, where the GRAPE boards were attached. I used the DAS extensively for my performance analysis work presented in chapter 2. Thanks to ASCI also for the availability of the new DAS-2 system, that was used to carry out part of the simulations upon which chapter 5 is based.

I'm grateful to Vittorio Rosato from the HPCN group of ENEA for his invitation, which resulted in a very instructive visit to the ENEA centre at Casaccia. There, I was introduced to the APE system, and had the chance to meet the members of the HPCN group, who I was very pleased to see again a few months later at the SIMAI conference in Ischia. During that visit, I also met Roberto Capuzzo Dolcetta from the University of Rome, who I'd also like to acknowledge for his activities towards strengthening and expanding the role of Computational Astrophysics in Italy.

I'm indebted to Michael Sipior for proofreading chapter 1 and part III of this dissertation during a joint train trip to Antwerp and back. His corrections were my only achievement of that day, for our great expectations of what brought us to Antwerp were completely frustrated. And, I'm afraid that Michael's work was also in vain, as the modifications that I introduced subsequently certainly spoiled Michael's efforts to de-italianise my English. I'm also profoundly grateful to Roeland Merks, who accepted, and heroically fulfilled the duty of amending my creative Dutch, and rendered the "samenvatting" of this thesis a decent piece of text. My reckless offer of compensating his efforts with a beer for each mistake he would find could have bankrupted me if Roeland had really counted them. I could have done better if I had bought an entire brewery for him! Roeland even raised the bid: he said he would buy a drink for each mistake he left in the "samenvatting". The error-hunting is still open! I'm also grateful to Amy Soller for revising the text of these acknowledgments, and Juan Heguiabehere for his prompt and remote help in obtaining the ISBN number of this book.

I'd like to acknowledge Marteen de Rijke for writing the ILLC dissertation style LaTeX package, which I used for this thesis, Roeland Merks (too many acknoledgements for you, Roeland!!) for providing me with his modified version of the bibliography style package used here, Rosella Gennari for the countless number of times when she helped me with LaTeX related problems, always reproving me for my troglodytic use of software tools. I express my gratitude to the great community of TeX and LaTeX developers for their contributions in constantly enhancing this formidable typesetting system, and to the initiator of this enterprise, Donald Knuth, for his love of beauty and elegance.

A dissertation is a place where scientific results of a Ph.D. are collected, hence emotional feelings are better kept out of it. My gratitude for the friendship and availability that my colleagues of the SCS group, the Institute for Informatics, the secretariat personnel, and my friends in Amsterdam and elsewhere showed me won't be acknowledged here, but personally to each of them.

Since I graduated in 1995 in Palermo, I stayed in different places around Europe, met

many interesting people, and got in touch with different cultures. Man's real home is not a house, but the Road, and life itself is a journey to be walked on foot, said Chatwin, and this could also be my motto. Meeting my travel mate for this journey was the greatest achievement of my stay in Amsterdam.

Piero Spinnato Trento, 5th August 2003

PS The cover image is inspired by Phaedrus' *De vulpe et uva*:

Fame coacta vulpes alta in vinea
Uvam appetebat summis saliens viribus;
Quam tangere ut non potuit, discedens ait:
'Nondum matura est; nolo acerbam sumere'.
Qui, facere quae non possunt verbis elevant,
Adscribere hoc debebunt exemplum sibi.

Guessing the metaphor that connects the above fable to this thesis is left as an exercise to the reader.

Chapter 1

Introduction

1.1 Preliminary remarks

Over the past 40 years, computer systems have shown an explosive growth in their computing power, pervading and influencing almost every aspect of our society. The scientific community has greatly benefited from this continuous increase in computer performance, which in a way is the reward for having provided the initial impetus for the pursuit of ever faster computer systems. Equally important for the scientific community is the development of faster and increasingly sophisticated software that has gradually expanded the role of computer systems in science from a mere support tool for numerical analysis to a fully-fledged environment to perform virtual experiments. The joint availability in so-called virtual laboratories of very powerful computer systems and very fast and accurate numerical algorithms nowadays permits the reproduction *in silico* of natural phenomena, and has resulted in the rise of Computational Science as a modern way of carrying out scientific research.

Traditionally, scientific investigation has been based on two pillars, theory, and experiments. The virtual laboratory, which provides the possibility of executing highly accurate simulations of complex natural phenomena, has led to the rise of simulation as a third pillar of scientific research. Natural phenomena as diverse as the interactions among the molecules that constitute a chemical solution, or the dynamics of stars that form a globular cluster, or the growth of a coral subject to environmental conditions, can be studied by means of computer simulations.

Computer simulations have the great advantage of allowing the investigation of phenomena that are very difficult, or even impossible to reproduce in a real laboratory, as in the cases cited above. A theoretical study of the dynamics of a chemical solution, or a star cluster, is not possible because the equations describing the system are unsolvable analytically. On the other hand, observing the dynamics of the constituents of the above systems is equally infeasible for an experimental scientist. The simulative approach is the only feasible means to tackle the study of such phenomena.

The simulative approach also shares many of the difficulties with the other two approaches. The tasks needed to set up a computer experiment are all prone to mistakes, and require both experimental and theoretical expertise. First, a mathematical model of the physical system under study needs to be developed. Then the set of equations that constitute the model needs to be discretised, and converted into a numerical algorithm, which is implemented by a computer code. Finally, the computer experiment can be performed, and the simulation results analysed as if they were obtained from measurements in a real laboratory.

The modelling of natural phenomena, the development of software for their simulation, together with the tasks of actually performing simulations and analysing the data output characterise the work of a computational scientist.

1.1.1 Thesis rationale

This thesis is concerned with the analysis of tools developed to make the simulation of so-called "N-body systems" fast and accurate. The molecules that constitute a chemical solution, or the stars that form a globular cluster are examples of N-body systems. Our focus in this thesis is on N-body systems subject to the force of gravity. The problem of solving the equations describing such systems is the *gravitational N-body problem* (see, e.g., Heggie & Hut, 2003; Hockney & Eastwood, 1988).

The N-body problem is analytically unsolvable, and its numerical solution needs high performance computing and sophisticated algorithms. In fact, the numerical solution of the gravitational N-body problem is so demanding in terms of computing power, that sophisticated fast algorithms have been devised to reduce the numerical complexity of the problem, trading higher speed for lower accuracy, and dedicated hardware has been developed to speed up N-body simulations requiring high numerical accuracy.

The central focus of this thesis is to explore the possibility of using dedicated hardware in connection with a powerful general purpose host, consisting of a parallel computer. We call these systems *hybrid architectures*. We try, by integrating a fast special purpose device into a parallel computer, to hybridise the two approaches, generalisation versus specialisation. People aiming at generalisation look more favourably on commodity systems, e.g. Beowulf systems or grid systems. The goal of the other approach is to obtain very high performance by means of hardware specialisation, developing, e.g., special purpose devices. Our research aims at bridging the gap between these two approaches, evaluating the viability of hybrid architectures, and their potential to solve large-scale simulation problems.

It is very important to understand the interplay of the parallel host, the dedicated hardware, and the application that runs on the hybrid architecture, in order to prevent bottlenecks, and find the optimal configuration. The tool we employ to study the interaction of a hybrid architecture with the software applications executed on it is *performance modelling*. By using performance modelling, we adopt a simulative approach to study systems that are used themselves to perform simulations. This meta-simulation is a core component of our research aimed at finding the optimal interaction between fast software and hardware in order

to devise a very high performance computational environment for the N-body simulation.

A main objective of this thesis is to show the potential of hybrid architectures to provide the optimal computing environment for the solution of specific problems. In view of this, we studied a numerical algorithm, and refined it for our hybrid architecture. This algorithm allows us to use fast N-body codes on dedicated hardware, with consequent computational performance benefits. This numerical algorithm is the software counterpart of our hybrid architecture, enabling a highly efficient computing environment for the N-body problem.

Finally, we look at the use of N-body simulations in astrophysical research. Specifically, we study the infall of a massive object to the Galactic centre. N-body simulations are an effective tool to study the time-scale of this infall, giving support to (or ruling out) theoretical models for the explanation of astronomical observations.

1.1.2 Chapter outline

The remainder of this chapter gives a brief introduction to the subjects that will be discussed in the thesis. We begin by explaining in section 1.2 why N-body systems are an important research subject, what computational problems they present and how these can be approached using the special hardware described in section 1.3 and the software described in section 1.4. We also introduce the hardware and software systems that we study in particular in this thesis. Namely, in section 1.3.4 we describe *Hybrid Architectures*, then, in section 1.5, we introduce the code that we studied and refined to make optimal use of these architectures. Next, in section 1.6, we explain how the performance of this combination of hardware and software can be evaluated and how this evaluation leads to a performance model that can be used for prediction. Finally, in section 1.7, we present an example of the use of N-body simulations in astrophysical research.

1.2 The computational N-body problem

In the study of N-body systems, Computational Science clearly demonstrates the potential of the simulative approach to attain dramatic progress in the understanding of natural phenomena. In the most general formulation, an N-body system is a mathematical model, where N point-like constituents interact through a given force (see, e.g., Heggie & Hut, 2003, p. 15). The importance of N-body systems in the physical sciences comes from the fact that natural systems, as diverse as a stellar globular cluster or a chemical solute-solvent system, are instances of an N-body system. Our focus in this thesis is on N-body systems subject to the force of gravity, the so-called *gravitational N-body problem* (see, e.g., Heggie & Hut, 2003; Hockney & Eastwood, 1988).

The computational N-body problem can be stated as follows: given the positions and velocities of N point-like bodies, interacting with each other by means of a specified force, solve the equation of motion for each body. For the gravitational N-body problem, the interaction force between particles is described by Newton's inverse square law

```
C       OBTAIN THE CURRENT FORCE ON BODY I.
      DO 10 J = 1,N
      IF (J.EQ.I) GO TO 10
      A(1) = XX(J) - XI
      A(2) = XY(J) - YI
      A(3) = XZ(J) - ZI
      A(4) = A(1)*A(1) + A(2)*A(2) + A(3)*A(3) + EPS2
      A(5) = BODY(J)/(A(4)*SQRT (A(4)))
      F1(1) = F1(1) + A(1)*A(5)
      F1(2) = F1(2) + A(2)*A(5)
      F1(3) = F1(3) + A(3)*A(5)
   10 CONTINUE
```

Figure 1.1: A verbatim transcript of the direct code NBODY1 force computation loop. `XI`, `YI`, and `ZI` are the position coordinates of the particle on which force is currently computed (the so-called i-particle). `XX(J)`, `XY(J)`, `XZ(J)`, and `BODY(J)` are the position coordinates and the mass of the j-th force source particle, respectively. `EPS2` is the square of the softening parameter ϵ, a numerical parameter introduced to soften the interaction between very close pairs of particles. Modern versions of the code solve these close interactions with much more accurate and sophisticated methods (Funato *et al.*, 1996; Aarseth, 1999).

$$\mathbf{F}_i = G\frac{m_j m_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}(\mathbf{r}_j - \mathbf{r}_i) \quad . \tag{1.1}$$

Electrostatic interactions between electrically charged particles are described by the Coulomb force which, apart from a scaling factor, has the same form as the Newton force. In fact, an algorithm that computes all particle-particle interactions directly could be used in both cases equally well. But computing all interactions directly is a very expensive task, requiring $\mathcal{O}(N)$ operations per particle. Thus the computational complexity of the direct particle-particle method is $\mathcal{O}(N^2)$ per iteration. In fig. 1.1 we show the force computation loop of NBODY1 (Aarseth, 1963; Aarseth, 1985), one of the first direct particle-particle methods used to study the dynamics of astrophysical N-body systems. This code is the oldest of a class of algorithms developed by the computational stellar dynamics community for the study of systems requiring high computational accuracy (Aarseth, 1999). NBODY1 is one of our case-study codes; we describe it in more detail in section 1.4.1.

In astronomy, there is a large number of problems that can be studied as gravitational N-body systems. At the one extreme, cosmological problems are characterised by having a very large number of particles, but a relatively low density and very long time-scales for two-body interactions; at the other extreme we have the study of globular clusters and the formation of planetary systems, which are characterised by high densities and short time-scales for two-body interactions. An important parameter that characterises these systems is the ratio of the so-called relaxation time and the age of the system. The relaxation time is defined as the time in which the velocity of a star is significantly changed by the cumulative effects of two-body encounters with background stars. In Heggie & Hut (2003, p. 136) the

relaxation time for average quantities inside the half mass radius r_h of a star cluster is gives as:

$$t_r \simeq \frac{0.138 N^{1/2} r_h^{3/2}}{(Gm)^{1/2} \ln \gamma N} \tag{1.2}$$

Where m is the mass of the individual stars, and γ is a factor of order unity.

The importance of the relaxation time stems from the fact that this is the time-scale on which three-body encounters in the densely populated core of a star cluster can lead to the formation of binaries, or can cause existing binaries to become more tightly bound. The potential energy that is released in the formation or tightening of the binary, which causes the acceleration of the third star, is an important source of kinetic energy in the system, thus influencing the evolution of the system as a whole (Bhattacharya & van den Heuvel, 1991).

An N-body system is classified according to its dynamics as *collisional*, when its lifetime is greater than the relaxation time, *collisionless* otherwise. The relaxation time of a galaxy, which contains up to 10^{11} stars, is larger than the age of the Universe, and hence, *a fortiori*, larger than the age of the galaxy itself. Therefore, a galaxy is a collisionless system. Globular clusters include about one million stars. Their relaxation time is smaller than their age, which is also approximately equal to the age of the Universe. Therefore globular clusters are collisional systems. Approximate methods cannot be used for the simulation of such systems, since they do not provide the necessary accuracy needed to compute the effect of close encounters. In the case of collisionless systems, close encounters are not relevant for the long term dynamics of the system, hence approximate methods can be safely used. This leads to the apparently paradoxical situation that systems as large as galaxies or galaxy clusters including billions of particles can be routinely simulated, whereas simulations of globular clusters are yet limited to several hundred thousand particles. Section 5.3.4 contains a discussion on the relaxation time for the systems studied in our N-body simulations.

The $\mathcal{O}(N^2)$ scaling of the direct particle-particle method leads to execution times for realistic values of N that are unsustainable on ordinary computers, motivating the development of the special purpose devices described in section 1.3. Besides the $\mathcal{O}(N^2)$ scaling due to force computation, a further increase in computational complexity comes from time integration. N-body systems requiring high computational accuracy also require more time steps for time integration. See section 2.3.1 for a further discussion on this issue.

Several software techniques that have been developed in order to reduce this computational complexity will be discussed in 1.4. These methods, although reducing the computational complexity of the problem to $\mathcal{O}(N \log N)$, or even $\mathcal{O}(N)$, introduce approximations that inevitably decrease the computational accuracy. The simulation of collisional systems as globular clusters (see, e.g., Meylan & Heggie, 1997; Heggie & Hut, 2003) or proto-planetary clouds (see, e.g., Lissauer, 1993), requires a high accuracy that approximate methods do not provide.

The need to retain the direct $\mathcal{O}(N^2)$ method, which ensures exact force evaluation (obviously limited by machine precision) but at the cost of huge computation times, led to the development of a hardware solution. Instead of accelerating the computation by

means of faster software, an improvement of orders of magnitude has been attained by building a Special Purpose Device (SPD) devoted to the only task of computing gravitational interactions. This SPD, the GRAPE (GRAvity PipelinE), is the subject of the next section.

1.3 Hardware for the N-body problem

The GRAPE project (Sugimoto *et al.*, 1990; Makino & Taiji, 1998), undertaken by a small group of computational astrophysicists led by Jun Makino at the University of Tokyo, is one of the most successful enterprises in the development of an SPD for scientific computing. The Gordon Bell prize, awarded yearly to the fastest computer simulation in the world, has been won five times in recent years by simulations run on a GRAPE machine (Makino & Taiji, 1995; Fukushige & Makino, 1996; Kawai *et al.*, 1999; Makino *et al.*, 2000; Makino & Fukushige, 2001). The GRAPE-4, completed in 1995 (Makino *et al.*, 1997), was the first computer to break the Tflop/s peak speed barrier. The current configuration of the most recent machine of the series, GRAPE-6, reaches the 63 Tflop/s peak speed (Makino *et al.*, 2002). Developing an SPD has been rewarding from a price/performance perspective as well. The GRAPE-6 peak speed is substantially higher than that of the two fastest general purpose computers in the world, the Japanese Earth simulator,[†] developed for large scale climate and solid earth science simulation, which has a peak speed of 40 Tflop/s and a cost of 350 million dollars (Triendl, 2002), and the American ASCI-Q,[‡] used for nuclear weapons stockpile maintenance, whose peak speed is 30 Tflop/s and its cost 215 million dollars.[§] The total development cost of the GRAPE-6 is about five million dollars (Makino, 2001c), two orders of magnitude less than the cost of the two general purpose machines, see fig. 1.2.

The availability of GRAPE has allowed substantial progress in several fields of stellar dynamics, ranging from star cluster evolution (with the first clear evidence of so-called "gravothermal" oscillations in the core of a globular cluster (Makino, 1996)), to the understanding of black hole spiral-in in galaxy mergers (Makino & Ebisuzaki, 1996; Makino, 1997), to structure formation processes, as in the case of planet formation from proto-planetary clouds (Kokubo & Ida, 1996, 1998).

The impressive performance of the GRAPE comes mainly from three factors: first, the fact that the GRAPE has been developed with the purpose of performing only one specific task, trading generality for speed; secondly, the fact that this task consists of a small, but very demanding computational core, that can be implemented very efficiently in hardware as a pipeline of basic operations. The third reason is that this operation needs to be performed a very large number of times on a relatively small number of input values, in a manner that makes it very suitable for parallelisation.

The reasons stated above also explain why the GRAPE project has been able to stay ahead in the competition for processor performance against general purpose hardware. Since, according to Moore's famous law, commodity processors double their speed approximately

[†] `www.es.jamstec.go.jp/esc/eng/index.html`
[‡] `www.llnl.gov/asci/platforms/lanl_q/`
[§] `www.lanl.gov/worldview/news/pdf/HighPerf_Computing.pdf`

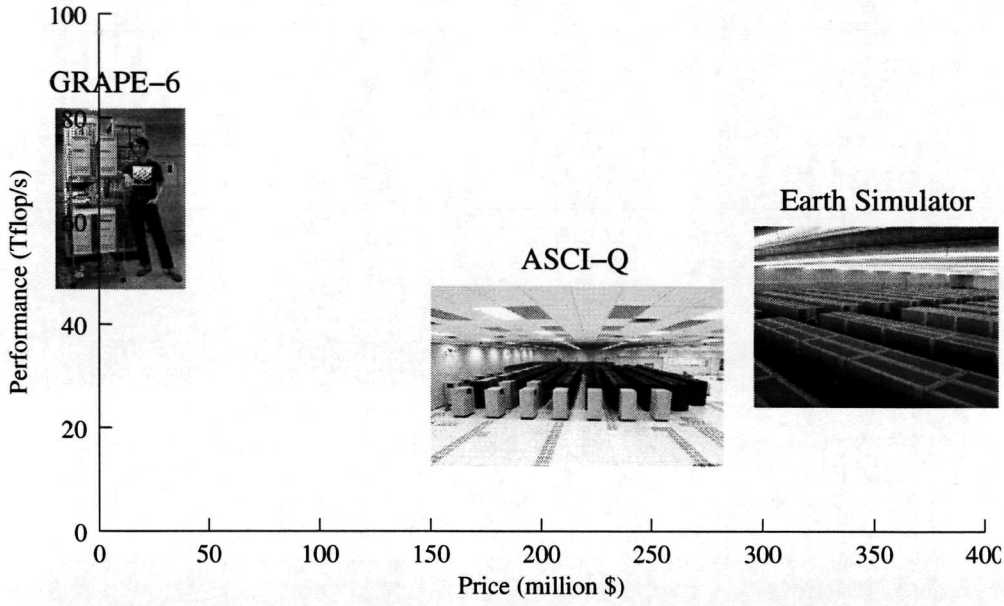


Figure 1.2: Price versus peak performance for the GRAPE-6 and the two fastest general purpose computers in the world. The physical size of the different systems can also be appreciated from the figure. The GRAPE-6 picture shows both the system and (on the right hand side) its main developer, Jun Makino.

every 18 months, the advantage in performance of an SPD would be soon obliterated by the progress in general purpose computer technology. The explanation why the GRAPE project is able to maintain its performance advantage comes from the relatively simple task that it implements in hardware. GRAPE developers are thus able to redesign a new GRAPE chip every three-four years, according to the most up-to-date microprocessor technology, keeping the GRAPE ahead in the performance competition.

As mentioned above, the task that the GRAPE accomplishes is the evaluation of the gravitational interaction between a pair of particles. The computation of the force exerted on a particle i by a particle j involves 18 mathematical operations, one of which is a division, and another one is a square root evaluation, as shown in the verbatim transcript of the force computation loop of NBODY1, fig. 1.1. In order to perform this computation, only four values have to be input, i.e. the position coordinates and the mass of particle j, while the position of i is stored in three local registers. This sequence of operations is repeated $N-1$ times for all the particles in the system except i.

The fact that a relatively high number of operations is performed on just four input values, and in a simple ordered sequence, makes the hardware implementation of this sequence as a pipeline relatively straightforward. Moreover, this task is easily parallelisable, because force on different i-particles can be computed concurrently using the same j-particle data (it is common practice in the N-body community to call the particle that exerts force the

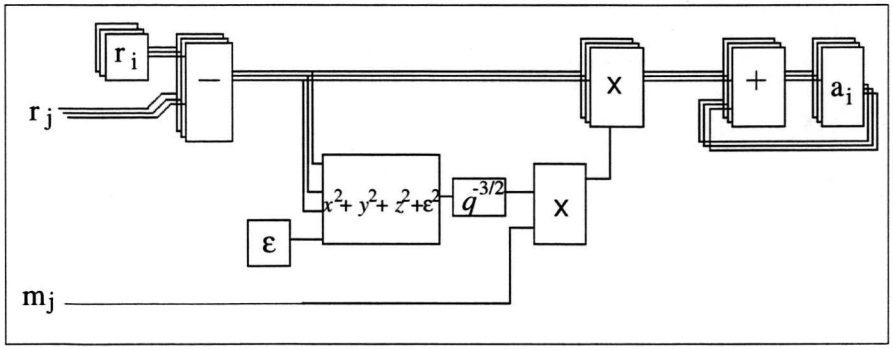


Figure 1.3: The pipeline for the force computation on the GRAPE. Figure adapted from (Makino, 2001b, fig. 4), reproduced here with author's permission. Here $\mathbf{r}_i$ is the position vector of the particle on which force is computed, stored in three pipeline registers. $\mathbf{r}_j$ and m_j are the position vector and mass of the j-th source particle, stored in the board memory, ε is the softening parameter mentioned in the caption of fig. 1.1, x, y, and z are the components of $\mathbf{r}_j - \mathbf{r}_i$, q is the sum of the squares of x, y, z and ε, and $\mathbf{a}_i$ is the partial accumulation of the gravitational acceleration on particle i. GRAPE-4 also includes a similar pipeline (not shown) for the computation of the acceleration derivative.

j-particle, and the one on which force is exerted the i-particle; we adopt this jargon here). In fig. 1.3 we show a sketch of the GRAPE acceleration pipeline,[†] which gives the name itself to the entire machine (GRAPE stands for GRAvity PipelinE, as mentioned above).

A pipeline also contains the circuitry to compute the gravitational potential for the particle i, and the time derivative of the acceleration, also called jerk, which is needed for the high accuracy time integration according to the Hermite method (Makino & Aarseth, 1992).

1.3.1 GRAPE-4

A GRAPE-4 board consists of an array of 96 such pipelines.[‡] A GRAPE-4 board also contains a pipeline for the extrapolation of the j-particle positions and velocities. The j-particle velocity is needed for the computation of the jerk. A board also contains memory to store data for about 44 000 j-particles (Kawai *et al.*, 1997). A sketch of a GRAPE-4 board is given in fig. 1.4.

[†]More precisely, the pipeline computes the force *field* at the i-particle position. This is equal to the particle acceleration in the case of gravitational interactions, but not in the case of, e.g., electrostatic interactions.

[‡]Actually, a GRAPE-4 board contains 48 physical pipelines, having a clock frequency twice the board clock frequency. In this way the board "sees" 96 virtual pipelines. Appropriate hardwiring manages the data exchange between the board and the pipeline (Makino *et al.*, 1997).

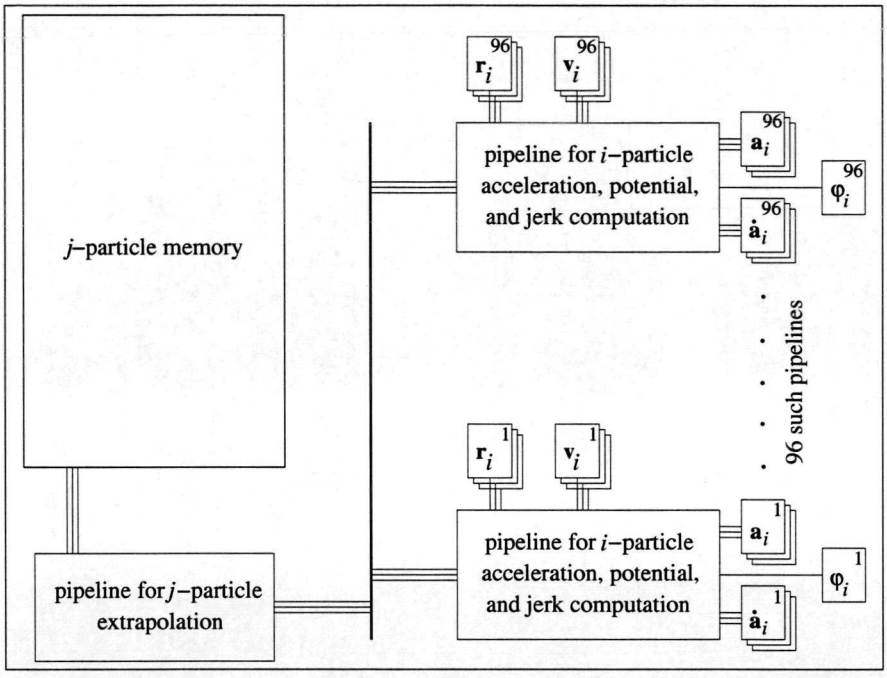


Figure 1.4: Sketch of a GRAPE-4 board. Force is computed on up to 96 i-particles simultaneously. At each cycle, a j-particle is loaded from the board memory, then its position and velocity are extrapolated to the i-particle time, these data are then fed into the acceleration and jerk (i.e. acceleration derivative) computation pipelines.

The performance of a GRAPE-4 board can be determined by considering the number of floating point operations executed by the pipelines that compute the acceleration and the jerk. Computing the 18 arithmetic operations of a single contribution requires 38 floating point operations (Karp, 1993; Warren *et al.*, 1997), and 19 more for the acceleration derivative (Makino *et al.*, 2000).

The total operation count is thus 57 floating point operations for a particle-particle interaction. The GRAPE-4 needs three clock cycles to perform a complete force calculation, whereas the GRAPE-6 performs it in a single clock cycle. The performance of a pipeline is obtained by multiplying the number of floating point operations per cycle by the clock frequency of the board. The 96 pipelines of the GRAPE-4 run at 16 MHz,[†] which gives $57/3 \cdot 16$ MHz $= 304$ Mflop/s per pipeline, and finally 304 Mflop/s $\cdot\, 96 \simeq 30$ Gflop/s per board. The GRAPE-4 system in Tokyo consists of 36 boards arranged in four clusters, which gives an aggregate peak performance exceeding one Tflop/s (Makino *et al.*, 1997). A sketch of the GRAPE-4 system is given in fig. 1.5. Sustained performance of 332 Gflop/s has been reached, and this was worth a Gordon Bell prize in 1996 (Fukushige & Makino, 1996) for a simulation of galaxy formation.

[†]In fact, the 48 physical pipelines have a clock frequency of 32 MHz.

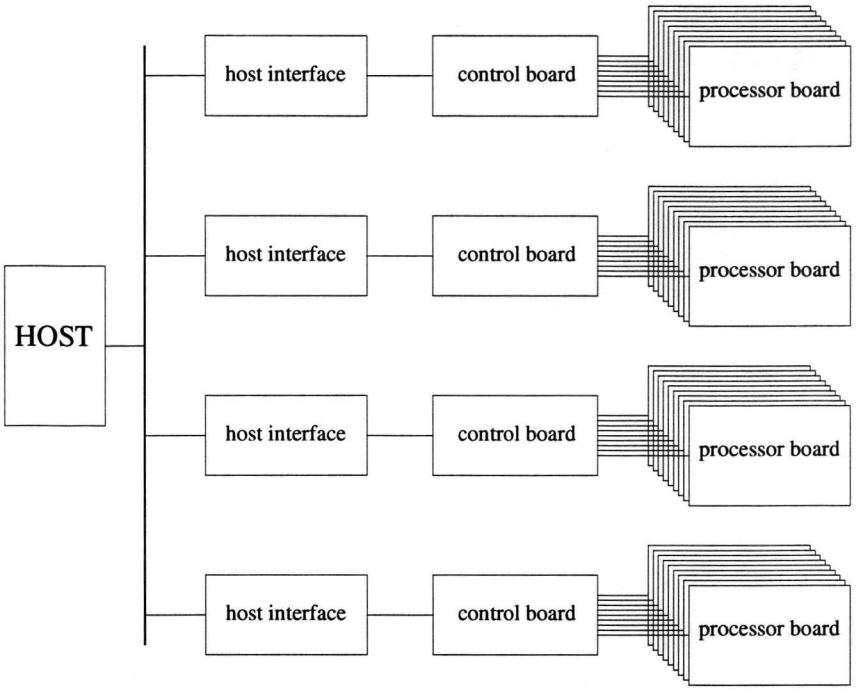


Figure 1.5: Sketch of the GRAPE-4 system at the University of Tokyo. The system consists of 36 processor boards as the one sketched in fig. 1.4, grouped in four clusters of nine boards each. The i-particle set is identical for each of the 36 boards. Each of the four control boards receives a different j-particle subset, and gives an equal part of this subset to each of the nine processor boards under its control. When the force computation is completed, the control board receives nine partial forces per i-particle from the processor boards, and sends the sum to the host, via the host interface. This reduces the communication bandwidth with the host, which communicates with only four peripherals, instead of 36. The host interface converts the internal GRAPE communication protocol to the host I/O protocol. This allows one to use the GRAPE with different hosts, changing only the host interface.

Our performance analysis and simulation studies, reported in chapters 2 and 3, are based on GRAPE-4 boards kindly made available to us by Jun Makino.

1.3.2 GRAPE-6

The progress in microelectronics made it possible to include in the GRAPE-6 chip six physical pipelines, able to compute a complete force contribution in a single clock cycle at a frequency of 90 MHz, whereas the GRAPE-4 needs three cycles. The peak performance of a GRAPE-6 chip is thus $6 \cdot 57 \cdot 90$ MHz $= 30.8$ Gflop/s, comparable to an entire GRAPE-4. In fact, a single GRAPE-6 chip implements all the operations implemented in a GRAPE-4 board, including j-particle position and velocity extrapolation. Similarly to the GRAPE-4,

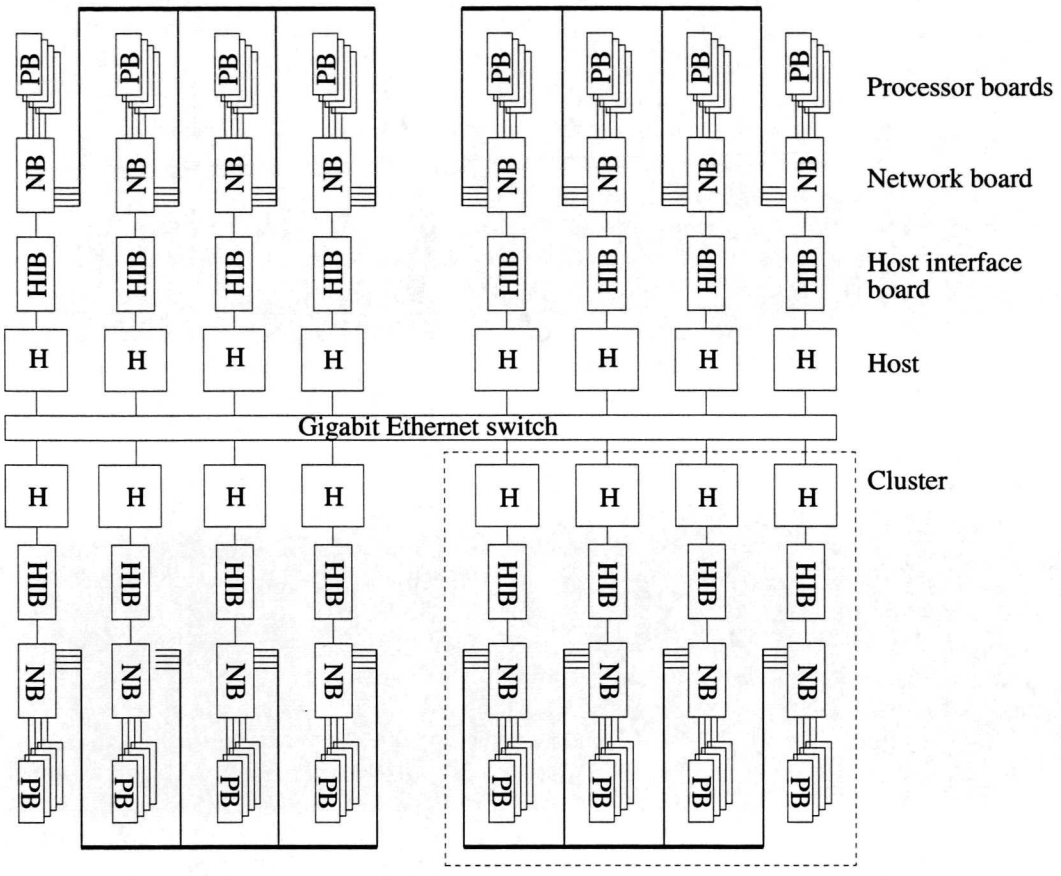


Figure 1.6: Sketch of the GRAPE-6 system at the University of Tokyo. Four clusters including four general purpose hosts and 16 processor boards are each connected by means of a Gigabit Ethernet switch. The processor boards in a cluster are able to communicate directly with each other by means of the network boards. Each network board controls four processor boards, and is directly linked with the other three boards in the cluster. In this way, each of the four hosts in a cluster, connected to a single network board via the host interface board, has direct access to all the processor boards in the cluster.

where 48 physical pipelines are seen as 96 virtual pipelines, the six pipelines of a GRAPE-6 chip are seen as 48 virtual pipelines, thus a GRAPE-6 chip is able to compute force on 48 different i-particles per clock cycle (Makino *et al.*, 2000).

A GRAPE-6 processor board includes 32 chips, which gives a peak performance of about 1 Tflop/s. The j-particle memory for a GRAPE-6 board is able to store data for 262 000 particles (Makino, 2003). The current configuration of the GRAPE-6 system in Tokyo includes 64 boards grouped in four clusters, for a total peak performance of 63 Tflop/s

(Makino *et al.*, 2002). A sustained performance of 22.72 Tflop/s has been reached for the simulation of planetesimal dynamics in the Uranus-Neptune region during the primordial phase of the Solar system's evolution (Makino *et al.*, 2002). A sketch of the GRAPE-6 system is given in fig. 1.6. It is much more complex than the GRAPE-4 system (shown in fig. 1.5). Now the general purpose front end of the system is a parallel computer, whose nodes are connected by means of a Gigabit channel. The front end nodes are Pentium-4 2.53 GHz, overclocked to 2.81 GHz (Makino 2003, private communication). The system is partitioned into clusters including four hosts, four host interface boards, four network boards, and 16 processor boards. A network board controls four processor boards, and is directly connected to the other three network boards of the cluster, allowing direct exchange of data among the boards, with no need to involve the host for communication.

The architecture of GRAPE-6, with its complex organisation of interconnected boards attached to a multiprocessor general purpose host, can be seen as an instantiation of the hybrid architecture model that we study in this thesis. This model is discussed in section 1.3.4 below.

1.3.3 GRAPEs in different fields

The impressive performance achievements of the GRAPE motivated the development of similar dedicated hardware in other contexts. The MD-GRAPE (Fukushige *et al.*, 1996) was developed with the purpose of implementing the computation of inter-particle forces depending on an arbitrary function of the particles' mutual distance. This also allows for the computation of the short-ranged van der Waals forces, which play a major role in Molecular Dynamics phenomena. MD-GRAPE also implements the hardware to compute inverse square law interactions with the Ewald method (Ewald, 1921), that is widely used in computational cosmology and computational chemistry to simulate systems with periodic boundary conditions. The recently developed MDM (Molecular Dynamics Machine) is an upgraded version of the MD-GRAPE, with a target peak-performance of 100 Tflop/s (Narumi *et al.*, 1999). MDM won a Gordon Bell prize for performance in 2000, shared with GRAPE-6, for a molecular dynamics simulation of 9 million NaCl ions (Narumi *et al.*, 2000).

The approach proposed in this thesis, of connecting a highly specialised SPD to a parallel general purpose computer, aims at expanding the range of applications of the special hardware in a different way. Instead of building a new dedicated hardware with new capabilities for performing those operations that, if executed on a serial host, would lead to a bottleneck, we still perform these operations on the *parallel* host of the hybrid architecture. The bottleneck is removed by distributing the computation on the nodes of the parallel host. In section 1.8 we discuss a specific case where our hybrid architecture approach could be effectively used.

We also expand the use of the GRAPE by means of software modifications. In section 1.5 we introduce a method that allows for the use of the GRAPE to compute the force from a multipole expansion of a particle distribution. Multipole expansions give force terms that have not an inverse square expression, thus the GRAPE could not be used for this computation. By converting the multipole expansion into a pseudo-particle distribution (Makino,

1999), we obtain a force expression that can be computed on the GRAPE. In the next section, we look more in detail at the potential role of hybrid architectures in Computational Science.

1.3.4 Hybrid architectures for the N-body problem

As already mentioned in section 1.1.1, hybrid architectures are systems consisting of a combination of a traditional parallel computer and special purpose devices. The need for such systems arises when the tasks that need to be performed by the host of the SPD begin to exceed the capacity of a single machine. This may be the case because the required communication bandwidth to the SPD exceeds that of a single host, or because the computational tasks that need to be performed on that host become too large. Host computing is needed, for instance, for the handling of special situations, such as the modelling of binaries and of three-star encounters, for the modelling of additional physical processes, such as stellar evolution, and especially, when a treecode is used, for the management of the tree structure.

In the sections above, we presented the N-body problem, and the hardware techniques that the Computational Astrophysics community has developed for its solution. In section 1.2 we briefly mentioned that software techniques have also been developed to speed up N-body simulations. These techniques, namely the treecode, the FMM, and the PM method, will be described in section 1.4 below. The tool that we use to study the interplay of hardware and software components in a computer system is performance modelling, which is introduced in section 1.6. Performance modelling allows us to analyse the system, and design the optimal architecture with the help of performance simulation.

One of our goals in this thesis is the study of architectures where a fast method, namely the treecode, described in section 1.4.2 below, effectively profits from the use of a fast dedicated SPD, namely the GRAPE. A parallel computer is planned as the SPD host, in order to provide computational power that is well-matched to the other tasks of the method. Otherwise the host computations would easily become the system bottleneck. The GRAPE-6 system in Tokyo, described in section 1.3, or the SIMD-MIMD architecture described by Palazzari *et al.* (2000); Capuzzo Dolcetta *et al.* (2001) are examples of this kind of architecture. Part I of this thesis is devoted to the description of the research carried out in developing our performance modelling environment, exploring the computational properties of the hardware and software tools described above, and analysing their interaction when integrated into the hybrid architecture discussed in this section.

1.4 Software for the N-body problem

Our research interest in this thesis is focussed on the interaction of algorithms developed for N-body simulations with the GRAPE hardware in hybrid architectures. Foremost among the numerical algorithms developed in computational astrophysics for the solution of the gravitational N-body problem, those that have the characteristics to exploit the computational power provided by the GRAPE are direct $\mathcal{O}(N^2)$ methods (Aarseth, 1999; Spurzem, 1999;

Portegies Zwart *et al.*, 2001), and the $\mathcal{O}(N \log N)$ treecode (Barnes & Hut, 1986; Barnes, 1990; Warren & Salmon, 1995; Springel *et al.*, 2001).

Among the direct codes, we focus on NBODY1 (Aarseth, 1963; Aarseth, 1985). NBODY1 is the progenitor of a class of direct codes, of which NBODY6 is its last offspring (Spurzem, 1999). The code's sophistication has grown dramatically from NBODY1 to NBODY6, primarily in the treatment of close encounters, and stellar evolution, allowing for increasingly refined and reliable simulations of globular clusters and other collisional systems.

The other main software environment developed for the simulation of collisional systems is `starlab` (Portegies Zwart *et al.*, 2001), originally written by Piet Hut, and currently maintained by Steve McMillan. It includes the high order integrator `kira`, the stellar evolution package `SeBa` developed by Simon Portegies Zwart, the three- and four-body scattering package `scatter`, and a number of routines for the pre- and post-processing of simulation data. All the above modules are implemented as independent programs, and share the same I/O data structure, so that they can easily be piped together to obtain the appropriate program flow for the problem under study.

The inner computational core of an N-body code, in which almost all the execution time is spent, consists of the few lines shown in fig. 1.1. They have not changed since the early days of NBODY1, and are "compiled" in hardware in the GRAPE pipeline. Our interest is in the interaction of N-body codes and GRAPE devices, which can conveniently be studied by using NBODY1.

Direct codes ensure high accuracy, but at the cost of very high compute times. As mentioned in section 1.2, approximate methods have been developed, that allow for the simulation of collisionless systems. The approach adopted by these schemes is to group particles according to their spatial proximity, then evaluate a truncated multipole expansion of the aggregate, and use this expansion to compute the force exerted by the aggregate, instead of evaluating directly the contribution of each single particle of the aggregate. This approach allows us to reduce the number of operations needed to compute the force on a particle to $\mathcal{O}(\log N)$.

Two main algorithms that implement this approach have been developed: The Fast Multipole Method (FMM) (Greengard, 1988; Carrier *et al.*, 1988; Greengard & Rokhlin, 1997; Cheng *et al.*, 1999) used for electrostatic computations, and the treecode (Barnes & Hut, 1986; Barnes, 1990; Warren & Salmon, 1995; Springel *et al.*, 2001) employed for gravitational problems. Although both methods are used to compute inverse square law interactions, neither is used in the field of the other. The FMM is more suited for systems where density is distributed homogeneously, like in plasmas, chemical solutions, and other Coulomb force-dominated systems. The treecode is inherently adaptive, and is well suited for highly clustered systems, such as those dominated by the force of gravity. This point is further discussed in section 1.4.3.

The treecode, because of its reduced computational complexity, provides a dramatic speedup for the N-body simulation. Part of its computational core, as described below, is still the evaluation of direct particle-particle interactions described by Newton's law, eq. (1.1). This allows us to use the GRAPE to further accelerate this computation (see, e.g., Makino,

1991b). Yet only a fraction of the treecode force evaluations are computed as particle-particle interactions. This limits the speedup achievable by using the GRAPE. The treecode can be optimised in order to make full use of the GRAPE, as described later in section 1.5. We study and refine this optimised version of the treecode, in view of our research goal, where a GRAPE powered hybrid architecture is used to run a treecode optimised for the use of GRAPE. We give below an overview of the main features of NBODY1 and the treecode, in the context of their use with the GRAPE. We discuss the direct code and the treecode extensively, as they are the principal codes discussed for the remainder of this thesis. Then we give a brief description of the FMM and Particle-Mesh algorithms.

1.4.1 The direct code

NBODY1 was one of the first codes for N-body simulation to appear, developed by Sverre Aarseth at the Institute of Astronomy in Cambridge as early as 1963 (Aarseth, 1963). It consists of approximately 2000 lines of FORTRAN code (to be compared with the 34 000 lines of NBODY6 (Aarseth, 1999)), with a very simple program flow. A fundamental feature of NBODY1 is that it assigns individual times to each particle, as described below. As a consequence of this, particle data stored in memory refer to different moments in time.

At each iteration, the particle i with the smallest update time $t_i + \Delta t_i$ is selected for force computation; then positions and velocities of all the other particles are extrapolated to the update time $t_i + \Delta t_i$. The selection rule for the i-particle guarantees that the smallest update time $t_i + \Delta t_i$ is always in the future with respect to all individual times, so that all other particle positions are extrapolated forward in time. The j-particle extrapolation pipeline of the GRAPE serves precisely to this extrapolation task.

Then gravitational interactions are computed, determining the values of $\mathbf{a}_i$ and $\dot{\mathbf{a}}_i$ at time $t_i + \Delta t_i$. Finally the i-particle orbit is integrated and its new Δt_i is determined. It is clear how the GRAPE operational architecture reflects this algorithmic sequence.

Still, NBODY1 cannot efficiently make use of the GRAPE computing power. In this code, only one particle at a time is selected for force evaluation, whereas a GRAPE board is able to compute a number of force interactions concurrently, up to 96 for the GRAPE-4. In order to have a large number of particles that share the same individual time, the so-called *block time step* scheme has been developed (McMillan, 1986; Makino, 1991a). In this case, the time step value assigned to the particles can only be a (negative) integer power of 2. This allows particles to have the same time step value, which makes it possible to have many particles per iteration that require force computation, instead of only one.

Using this approach, force contributions on a large number of i-particles can be computed in parallel using the same extrapolated positions for the j-particles, i.e. the force-exerting particles. Then, when a GRAPE device is available, it is possible to make full use of the multiple pipelines provided by the hardware, since each pipeline can compute the force on a different i-particle. In this way, GRAPE provides orders of magnitude increase of performance for the direct N-body code execution. Simulation of globular clusters containing 10^5 particles or more are possible on the GRAPE-6, and even larger numbers can be reached

for the simulation of other systems (Makino, 2001a).

A detailed analysis of the direct code tasks, and its performance on the GRAPE-4 boards is given in chapter 2. N-body simulations carried out with the direct code on GRAPE-6 are reported and analysed in chapter 5.

1.4.2 The treecode

The treecode (Barnes & Hut, 1986; Barnes, 1990; Warren & Salmon, 1995; Springel *et al.*, 2001), introduced by Josh Barnes and Piet Hut from the Institute for Advanced Studies in Princeton, is one of the most popular numerical methods for particle simulation involving long range interactions. It is widely used in the Computational Astrophysics community to simulate systems like single galaxies or clusters of galaxies. It reduces the computational complexity of the N-body problem from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$, trading higher speed for lower accuracy. The $\mathcal{O}(N \log N)$ scaling of the treecode allows the study of very large systems exceeding 10^8 particles, as in the case of simulations of the large scale structure of the Universe (Warren *et al.*, 1997). Such simulations run on general purpose supercomputers. Can the use of GRAPE provide a further speedup to treecode simulations? In practice, using the GRAPE efficiently when executing the treecode is not an easy task, since particle-particle interactions, i.e. the computing task implemented on the GRAPE, are much less computationally relevant for the treecode, with respect to the direct code (see, e.g., Makino, 1991b). In fact, the superior $\mathcal{O}(N \log N)$ of the treecode is due to a decrease in the number of direct particle-particle computations performed to evaluate gravitational interactions. A description of the main treecode procedures is given below.

Procedure description. The treecode approach for computing forces on a given particle i is to group particles in larger and larger cells as their distance from i increases, and compute force contributions from these cells using truncated multipole expansions. The grouping is realised by inserting the particles one by one into the initially empty simulation cube. Each time two particles are in the same cube, that cube is divided into eight "child" cubes, whose linear size is one-half that of their parent's. This procedure is repeated until the two particles find themselves in different cubes. Hierarchically connecting such cubical cells according to their parental relation leads to a hierarchical tree data structure (see fig. 1.7).

When the force on a given particle i has to be computed, the tree is traversed searching for cells that satisfy an appropriate Multipole Acceptability Criterion (MAC). If a cell satisfies this criterion, the force from the entire particle distribution within the cell is computed using the cell multipole expansion, and the search skips the cell's children. Conversely, if the cell does not satisfy the MAC, then its children are examined. By applying this procedure recursively, starting from the tree root, i.e. the cell containing the whole system, all the cells satisfying the acceptability criterion are found. The most commonly used expression for the MAC (see, e.g., Barnes & Hut, 1986) is:

$$\frac{l}{d} < \theta \tag{1.3}$$

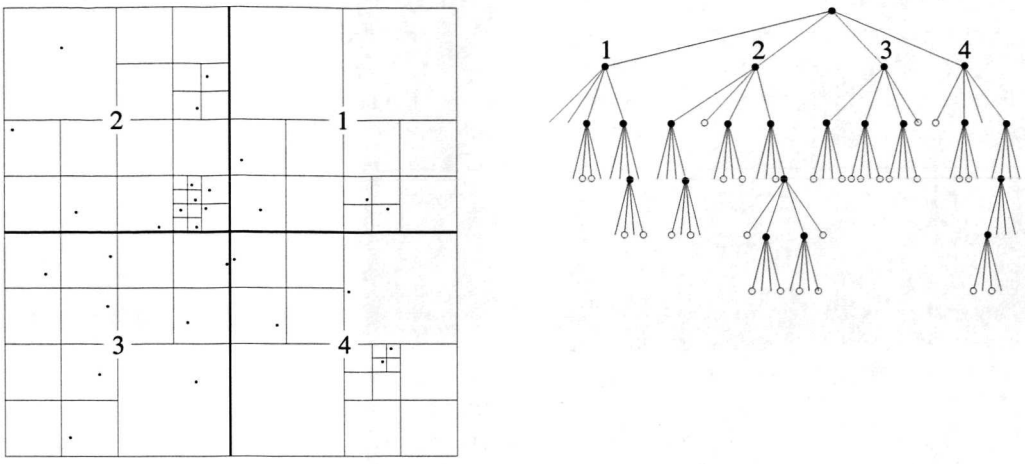


Figure 1.7: Sketch of the treecode space partition, and corresponding hierarchical tree data structure. The root cell, the one that encompasses the particle distribution, is recursively subdivided, until every particle is contained in a different cell. The corresponding tree data structure is shown on the right. The node corresponding to a given cell is marked with an empty circle if the cell is terminal (i.e. if it contains only one particle, and hence is not further split), or a full circle if the cell is not terminal. Cells containing no particles have no specific mark in the tree. The node corresponding to the root cell, in spite of the name, is on top of the tree, and is connected to the nodes corresponding to the root's daughter cells. Mapping from cells to tree nodes is shown for the first hierarchical level of the tree. This mapping is repeated recursively while traversing the tree downwards.

where l is the cell size, d is the distance between i and the cell's centre of mass and θ is an input parameter, usually $\theta \lesssim 1$. The MAC in eq. (1.3) has a simple physical interpretation. l/d can be seen as a measure of the opening angle under which an object of typical size l is seen from a distance d. Eq. (1.3) states that a cell is accepted if its opening angle is smaller than the threshold opening angle θ.

Theoretical complexity. The treecode force computation procedure scales as $N \log N$; in order to see that, suppose we increase N k-fold by replacing each particle with k particles having mass $1/k$ of the replaced particle mass. Then each cell will generate a number of new cells n, where $n \geq k$. The particle i "sees" this finer subdivision only within its nearest neighbourhood. The MAC is such that when a cell $\mathcal{C}$ is further from i than its own size divided by θ, i will still interact with $\mathcal{C}$, and not with the new "children" of $\mathcal{C}$. The total number of force evaluations on the i-particle as N increases is only dependent on the increase of particles in the neighbourhood of i. We want to find out how this increase affects the number of tree subdivisions, and show that the latter scales as $\log N$.

The increase of particles in the neighbourhood of i can be measured by the interparticle distance. In order to show that the latter is related to the number of cell subdivisions, assume that the particles are uniformly distributed. The interparticle distance is then proportional to $N^{-\frac{1}{3}}$. This can also be seen as a measure of the smallest cell size. But, since the cell size halves at each cell subdivision, the smallest cell size is proportional to $2^{-\lambda}$, where λ is the highest tree order. Equating the two quantities gives $2^{-\lambda} \propto N^{-\frac{1}{3}}$, and finally $\lambda \propto \log N$. Thus the cell subdivision and the number of cells opened during the force evaluation for a particle scale as $\log N$; so that the force computation scaling for the whole system is $\mathcal{O}(N \log N)$.

Interacting with the GRAPE. The tree building and traversal, that allows the algorithm to gain the $\mathcal{O}(N \log N)$ scaling, also dramatically changes the relative computational load of the different tasks of the program. Whereas in the direct method the force computation is by far the most demanding task, taking virtually 100% of the execution time, in the treecode this value decreases to approximately 50% (Makino, 1991b). Moreover, the force is usually computed as a multipole expansion up to the quadrupole term. The result of this is that the particle-particle interactions, i.e. the monopole term contributions, are less computationally demanding in the treecode, compared to the direct code. This decreases the effectiveness of using GRAPE to accelerate the treecode execution. In fact, GRAPE is used with the treecode with good results (Makino, 1991b; Athanassoula *et al.*, 1998), but in those cases the multipole expansion is limited to the monopole term, increasing the accuracy by reducing the value of θ in the MAC formula, eq. (1.3).

1.4.3 The Fast Multipole Method and Particle-Mesh methods

The FMM and the PM methods are the other main schemes used in N-body simulations of systems dominated by an inverse square law. The FMM subdivides the physical space by means of a regular grid, and repeats this subdivision recursively for each cell of the grid, terminating the recursion after a fixed number of steps. Multipole expansions for the lowest level cells are computed directly from the particles contained in them. Then expansions for the encompassing cells are computed recursively by propagating the daughter cell expansions upwards. Then cell-cell interactions are computed at the highest level for non nearest-neighbour sibling cells; the expression for the force exerted on each cell is then propagated downwards to the cell's daughters. This force term represents the far field force inside the daughter cells. The near field force is computed again as a sum of cell-cell interactions from non nearest-neighbour sibling and "cousin" cells (i.e. daughters of the parent cell's siblings). This process is repeated for each cell until the particle level, at which point the near field force is computed directly as a sum of particle-particle interactions.

This method is suitable for homogeneous systems, but does not perform well for inhomogeneous distributions. In this case adaptive methods, that refine the spatial subdivision according to the particle density, are better suited. The treecode has been developed to be adaptive. In this case, as described in section 1.4.2, particle-cell interactions are computed for the far field, and particle-particle interactions for the near field. There are no

cell-cell interactions. In this way, it is easy to continue the cell subdivision further in high density regions. The FMM is claimed to be $\mathcal{O}(N)$, even though discussion continues on this point (Aluru, 1996). In fact, asymptotic behaviour generally is not reached in FMM simulations, so that no real difference with a $\mathcal{O}(N \log N)$ scaling is usually experienced (see, e.g., Capuzzo Dolcetta & Miocchi, 1998).

Another scheme that is often used for N-body simulations is the particle-mesh (PM) method (Hockney, 1965; Hockney & Eastwood, 1988; Couchman *et al.*, 1996; Fellhauer *et al.*, 2000). In this case, the far field is not computed from multipole expansions, but by means of a regular grid. Density values are computed for each grid point from the particle distribution of its neighbour, then the Poisson equation is solved on the grid using fast Fourier transforms, so that the gravitational (or electrostatic) potential is known for each grid point. Finally, from the potential value on the nearest grid point, the potential on each particle is evaluated.

When needed for additional accuracy, the near field force can be computed by means of direct particle-particle interactions; in this case, the method is called P^3M (particle-particle particle-mesh). State-of-the-art codes use a recursive, spatially adaptive grid refinement (Couchman *et al.*, 1996; McFarland *et al.*, 1998; Fellhauer *et al.*, 2000), in order to cope with particle-particle computational bottlenecks arising in high density regions. We used a multi-grid PM code (Fellhauer *et al.*, 2000) in our comparative N-body simulations presented in chapter 5. The PM method scales as $\mathcal{O}(N \cdot n_c^3)$, where n_c is the number of cells per dimension. This clearly limits the possibility of increasing the PM accuracy by means of mesh refinement.

1.5 Software for hybrid architectures

The treecode, as described in section 1.4.2, provides a substantial speedup to N-body simulations. Using it on a hybrid architecture as the one discussed in section 1.3.4 could lead on a further substantial performance improvement. This perspective is generally applicable, and is not limited to the gravitational N-body problem. The fact that both fast software and dedicated hardware have been developed for its solution makes the N-body problem ideal for studying the potential of hybrid architectures in Computational Science. Our aim is also to make the techniques developed in Computational Astrophysics available to the much larger community that is involved in N-body simulations.

For instance, applications in science and engineering that involve Coulomb force computations could benefit from the computational environment provided by the hybrid architecture that we study. The FMM, as described in section 1.4.3, provides a robust mathematical structure, by means of which multipole expansions can be computed to any order, with an analytically bound accuracy error. The treecode is much more empirical in this sense. In fact, since the dominant force in astrophysical systems, gravity, is always attractive and cannot be shielded, a multipole expansion of such force will have a very large monopole term, and terms up to the quadrupole are usually sufficient to ensure acceptable accuracy in simulations where the treecode is used (see, e.g., McMillan & Aarseth, 1993). Multipole expansions in Coulomb force-dominated problems must include a larger number of terms, be-

cause the net effect produced by opposite-sign charges results in very small low order terms. The computation of higher multipole terms can be implemented in the treecode (McMillan & Aarseth, 1993), but then a problem arises: the computation of the force contribution from terms of the expansion other than the monopole cannot be done on the GRAPE, since the GRAPE only computes particle-particle interactions, i.e. monopole term contributions.

A solution to this problem comes from a technique originally introduced in the FMM framework by Chris Anderson of the University of California at Los Angeles (Anderson, 1992), and further developed by Atsushi Kawai and Jun Makino to be implemented on the GRAPE (Makino, 1999; Kawai & Makino, 1999). It consists of converting the multipole expansion into a pseudo-particle distribution; in other words, in finding a distribution of fictitious particles that produces the same force field as the original distribution, up to a given multipole term. Now, since the multipole expansion is expressed as a particle distribution, the GRAPE is also able to compute the contributions of higher order terms. This allows us to increase the accuracy of GRAPE based simulations performed with methods such as the treecode, and paves the way for using our hybrid architecture in fields like Molecular Dynamics. Chapter 4 is devoted to the description of the pseudo-particle approach, in the framework of our research concerned with multipole temporal expansion, and improvement in method accuracy.

Although with the pseudo-particle approach the use of the GRAPE by the treecode is optimised, the general purpose computer that hosts the GRAPE still has a large computational load, and can easily become the system bottleneck. In order to improve the host performance, so that the advantages provided by the treecode and the GRAPE can be fully enjoyed, it is important to understand the interplay between the GRAPE, the host and the treecode. The tool that we use for this study is performance modelling, as described in the following section.

1.6 Performance modelling for the N-body problem

Performance modelling (see, e.g., Jain, 1991; Sauer & Mani Chandri, 1981) is a useful tool for the study of computer system behaviour. It allows us to estimate the performance of a hardware or software architecture by means of an abstract model, in which each task of the system under study is specified in terms of its execution time as a function of a number of parameters.

For a hardware system, these parameters can be basic performance measures such as the processor clock speed, operations per second, or the bandwidth of a communication line. For a software application, a typical parameter is the problem size. For example, for NBODY1 or the treecode this is the number of particles N, or an input parameter of the code, such as the treecode opening angle θ. Building a performance model for the systems studied in this thesis involves the formal description of both the software and the hardware components.

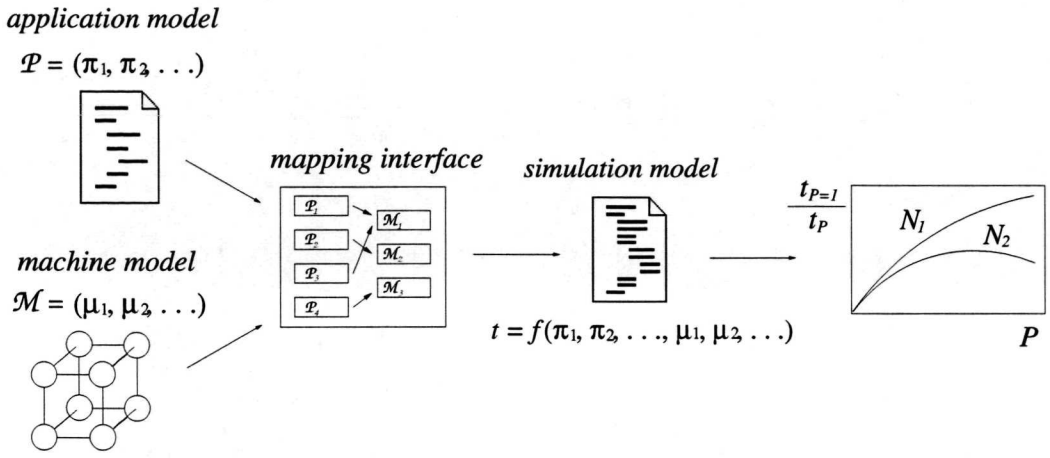


Figure 1.8: Performance modelling process. Figure adapted from the slides of the ASCI† course "Performance Modeling of Parallel Systems", taught by Arjan van Gemund, reproduced here with author's permission. The output of the simulation model, t, is the modelled execution time. The rightmost sketch represents the speedup in the execution of a parallel application, for two different values of a certain parameter N, as a function of the number of processors P. The meaning of the other symbols is explained in the text.

Our modelling approach is illustrated in fig. 1.8. In the *application model*, each task of the application is described in terms of the operations performed, and the workload that these operations produce. Workload is expressed as a function of the application parameters $\mathcal{P} = \{\pi_1, \pi_2, \ldots\}$. In the *machine model*, each architecture resource is specified in terms of the time spent accomplishing the task it was designed to perform, as a function of the machine parameters $\mathcal{M} = \{\mu_1, \mu_2, \ldots\}$. The calibration of this function is determined by timing sample runs of the real application. An important layer of the model is the mapping of the application tasks, each one depending on a subset $\mathcal{P}_i \subseteq \mathcal{P}$, to the appropriate machine resources, which depend on a subset $\mathcal{M}_i \subseteq \mathcal{M}$. The *mapping interface* specifies this. This formal description of the system is expressed in terms of a suitable language, which in our case is PAMELA, developed by Arjan van Gemund at the Delft University of Technology (van Gemund, 1993, 2003). A language interpreter converts this formal description into the machine executable *simulation model*. The output of the simulation model is the execution time of the application (which depends on the $\mathcal{P} \cup \mathcal{M}$ set of parameters), the utilisation of the various hardware components, and other performance measures.

†ASCI is the Advanced School for Computing and Imaging (see `http://www.asci.tudelft.nl`). It is unrelated, and predates, the homonym programme of the Department of Energy of the USA.

A metaphor for the performance modelling approach could be the sequence of actions performed when an executable is produced from a mathematical algorithm, as described below.

> The first step is to write a source code in the programming language of choice, that implements the algorithm. Each function of the algorithm is expressed as a sequence of commands in a software module. This is the analogue of building the machine model and the application model as a representation of the real machine and the real application.
>
> Then the various modules are linked to produce the executable. The analogue of this is the activity of the mapping model, and the resulting execution model.
>
> Finally, the executable is used to perform the computation that it was designed to do. Correspondingly, the execution model is run, by giving it appropriate values for the system parameters as input, and obtaining the execution time as output.

In order to show how performance modelling actually works, we describe here the modelling of the force evaluation task, i.e. the computation of the force exerted on a subset of particles, by the particles assigned to a computing element.[†] Fig. 1.9 shows how this task is modelled. A module in the application model calls the mapping interface, passing it the number of particles that exert force, N_j, and the number of particles for which the force is to be computed, N_i. The mapping interface selects the module that will accomplish the task, choosing the function that models the computation on the GRAPE when present (not unexpectedly this module is represented as a grape bunch in fig. 1.9), or the function that models the computation on a general purpose processor otherwise. The GRAPE model includes the actual force computation and the communication between the host and the GRAPE. The function that simulates the time spent by the GRAPE in performing the force computation depends on N_j and N_i. This time function also includes the communication delays between host and GRAPE. The general purpose machine model is much simpler, since no communications are involved. The force computation model in this case consists of a simple delay function.

Analysis of the simulation traces, in terms of appropriate metrics, e.g. speedup as in the example sketched in fig. 1.8, allows us to understand which parameters are relevant in affecting the system performance, and how a modification in the application or in the architecture influences the final performance.

Performance modelling is also a very effective tool for the design of computer architectures. The actual installation of a high performance computer system is a very expensive enterprise, in terms of both economic costs, and research and technology efforts for system planning and implementation. Obviously, nobody wants to incur the risk of embarking on such an enterprise, to sadly discover at the end that the system as constructed is inefficient. A tool that allows for fast and inexpensive prototyping is clearly desirable. Performance

[†]In the force evaluation task, a computing element is a GRAPE when the system includes it, or a common processor when no GRAPEs are available.

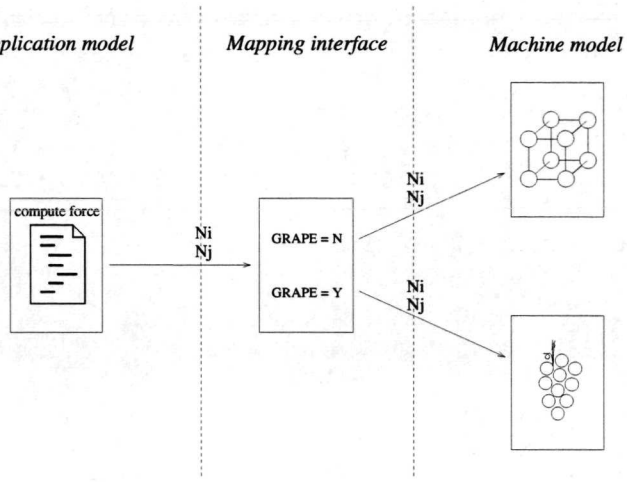


Figure 1.9: Sketch of the force evaluation task, as performed by our model. The application model module passes its parameters to the mapping module, that selects the architecture component that will perform the actual computation.

modelling provides a tool that simulates the planned architectures, allows one to discover inefficiencies in the interactions of the various system components, permits the exploration of different solutions to overcome such problems, and provides an environment in which the optimal architecture can be developed.

The aim of our performance modelling work is to realise an environment where the interplay of fast special hardware, general purpose host, and advanced software can be studied to determine the optimal interaction; i.e. an architecture where hardware and software are integrated to provide a very efficient tool for the simulation of N-body systems. We describe our envisaged architecture in the following section.

1.7 N-body simulations: the reason for it all

In chapter 5 we use the direct method, the treecode, and the particle- mesh code to perform N-body simulations of dynamic astronomical phenomena. Specifically, we study the infall of a black hole towards the Galactic centre. This infall is due to dynamical friction (Chandrasekhar, 1943), a drag force experienced by a massive body moving within a background populated by lighter bodies, and interacting with them by means of the force of gravity (see, e.g., Binney & Tremaine, 1987, sect. 7.1). The net effect of this interaction on the massive body is a force opposite to its velocity, which effectively acts as a friction force. When the body is orbiting around a centre of gravity, as the Galactic centre in our case-study, the deceleration of the body results in a spiral-in orbit towards that centre. This process can explain the presence of very young stars in the inner core of the Galaxy. These young stars

are not likely to be born in the Galactic centre because it is a hostile place for star formation, due to the strong tidal field that prevents interstellar gas from collapsing and forming a star. A possible explanation (see, e.g., Gerhard, 2001) is that dense young clusters, formed outside the Galactic inner core, spiral towards the Galactic centre due to dynamical friction, thus bringing the young stars in the cluster nuclei into the Galactic core. In the work presented in chapter 5 we estimate the typical infall time of an inspiraling object, which provides a constraint to this model. In fact, for this model to work, the cluster must reach the Galactic core before it evaporates, i.e. before the dynamical evolution of the cluster causes all the stars to escape from its gravitational potential well.

We study the infall process for a single massive particle, which actually models the spiral-in of a black hole. We carry out a comparative study of this spiral-in process, using a direct code (see section 1.4.1), a treecode (see section 1.4.2), and a PM code (see section 1.4.3). The direct code simulations are accurate, but highly granular, i.e. limited in the number of particles, because of the direct code $\mathcal{O}(N^2)$ computational complexity. The other methods are inherently less accurate, but allow us to use many more particles. We compare the accurate results of the direct method with the approximate results of the other two methods, in order to understand how granularity and inaccuracy affect our simulation results.

1.8 Hybrid codes on hybrid systems

Our case-study is also a first step in the direction of simulating the infall of a star cluster. In order to simulate the infall of a cluster on a star-by-star basis, the use of a direct code is essential. In fact, an approximate-method simulation is not able to follow the internal dynamics of the cluster accurately enough during its spiral-in; the cluster would evaporate much faster than is expected from theory (see, e.g., Kim & Morris, 2002). On the other hand, a complete direct code simulation of a cluster infall, that includes the background stars of the Galactic centre, is unfeasible because of the very large number of particles involved. Our intention is to develop a hybrid code, where a direct code simulates the cluster, and a treecode simulates the Galactic centre. The cluster is represented as a particle with variable mass in the treecode. The mass change is a consequence of the internal dynamics of the cluster. The cluster mass is an input value for the treecode co-simulation, and results from the direct code simulation. The input of the direct code co-simulation is the current value of the tidal field of the Galaxy, which is computed by the treecode as the force acting on the cluster particle.

This hybrid code not only represents a challenge with respect to its development, but is also quite demanding in terms of hardware performance. In order to run it efficiently, we need an architecture which is very powerful, both to compute the gravitational interactions, and to perform the other general purpose tasks of the hybrid code. Our envisaged hybrid architecture (cf. section 1.3.4) would be an ideal computational platform for this application, since it would efficiently run both the direct code and the treecode "phases" of the hybrid.

1.9 Thesis outline

This dissertation is divided into three parts. The first part is devoted to performance modelling and simulation, and consists of two chapters. Chapter 2 reports on the performance analysis of the direct code NBODY1 on our case-study architecture, which includes two GRAPE-4 boards connected to a distributed computer. It contains a detailed description of NBODY1 tasks, and presents performance measurements and analysis of various parallelised versions of NBODY1, running on our hybrid architecture. These measurements are the basis for our performance modelling and simulation of different architectures where direct N-body codes and treecodes are executed. This performance modelling and simulation work is presented in chapter 3.

The second part of this dissertation is also divided into two chapters. Chapter 4 is devoted to accuracy analysis and optimisation of the pseudo-particle treecode, which has been developed for optimal use with the GRAPE. We study the error behaviour of the pseudo-particle treecode with different particle distributions, and improve the code accuracy in the presence of highly inhomogeneous distributions. We also study an optimisation of the pseudo-particle scheme, introducing pseudo-particle velocity, which allows us to retain the pseudo-particle distributions for several time steps, whereas the standard scheme recomputes the pseudo-particles at each step.

Then, in chapter 5, we present our comparative multi-method N-body simulations, aimed at estimating quantitatively the efficiency of the spiral-in of a black hole towards the Galactic centre, and understanding the effect of particle granularity and code inaccuracy on the infall efficiency. Finally, in part III we summarise our work and discuss its future developments.

Part I

Performance Modelling and Simulation

Chapter 2

N-body Codes on Hybrid Architectures[†]

In this chapter we analyse NBODY1, the direct particle-particle code introduced in section 1.4.1, and study the performance of this N-body code on hybrid architectures, which were presented in section 1.3.4. A detailed analysis of the N-body code performance, in terms of the relative weight of each task of the code, and how this weight is influenced by software or hardware modifications, is essential to understand the interaction of the code with the hardware platform that executes it. Especially the interaction with the GRAPE, the dedicated device for N-body simulation introduced in section 1.3, requires a careful performance analysis. The use of GRAPE results in a dramatic performance leap for N-body simulations, as it provides a very high performance for the computation of gravity interactions, the most expensive computational task of an N-body code. The interaction of the GRAPE, its general purpose host, and the N-body code run on the machine, gives rise to complex execution patterns that need to be studied and understood to find the optimal configuration. We need this performance analysis in order to acquire the necessary experimental data, for our performance modelling and simulation research to devise a very high performance computational environment for N-body simulations.

2.1 Introduction

The importance of N-body codes for the simulation of the dynamics of astrophysical systems has been discussed in chapter 1. The core of an N-body code is the computation of

[†]This chapter is based on work published in:

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Analysis of Parallel N-Body Codes*, in M. Bubak; H. Afsarmanesh; R.D. Williams and L.O. Hertzberger, editors, Proceedings of the HPCN2000 Conference, LNCS vol. 1823, pp. 249–260. Springer-Verlag, 2000.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance of N-body Codes on Hybrid Machines*, Future Generation Computer Systems, **17**, 951-959, 2001.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Modelling of Distributed Hybrid Architectures*, IEEE Transactions on Parallel and Distributed Systems, in press, 2003.

the gravitational interactions between all pairs of particles that compose the system. In sections 1.2 and 1.4 we discussed the main algorithms developed for the computation of gravity interactions between a given particle i and the rest of the system. In this chapter we study the performance of the direct particle-particle method (Aarseth, 1985, 1999), which exactly computes the gravity force that every particle in the system exerts on i. The high accuracy of the direct method is obtained at the cost of a computational load which grows as N^2 per time step.

The huge computational requirements of the direct N-body code led to the development of the GRAPE, a special purpose device for gravity force computation, described in section 1.3. A principal objective of our research is the efficient integration of GRAPE boards with a parallel general purpose host, to realise a hybrid architecture for N-body simulations, as discussed in section 1.3.4.

The performance analysis research presented in this chapter aims at understanding how such architectures interact with the N-body code. For this purpose, we use NBODY1 (Aarseth, 1963; Aarseth, 1985) as a reference code. NBODY1 was introduced in section 1.4.1. We use it to determine the scaling properties of various parallel versions of the code, running on a hybrid architecture which includes two GRAPE-4 boards connected to a distributed computer (see fig. 2.1). The performance data obtained will be used in chapter 3 for the realisation and calibration of a performance model that we use to study hybrid architectures for N-body simulations, and their interaction with various types of N-body codes.

2.2 System description

2.2.1 Architecture

Our hybrid architecture, sketched in fig. 2.1, is composed of a parallel general purpose multicomputer, DAS (Bal *et al.*, 2000), and an SPD, GRAPE (see, e.g., Makino *et al.*, 1997; Makino & Taiji, 1998). The DAS multicomputer is a wide-area distributed computer including 200 nodes in total, grouped into four clusters located at different locations in the Netherlands. The cluster at the University of Amsterdam, which served as a testbed for our model, comprises 24 processors. Technical characteristics of our testbed system are summarised in table 2.1.

In January 2002 the new DAS-2 came into service. DAS-2 is also a wide-area distributed computer including in total 200 1-GHz dual Pentium-III nodes grouped in five local clusters interconnected via the Dutch university Internet backbone. Local clusters are connected by a fast Myrinet network having a bandwidth of 250 GBytes/s peak-performance. We used the DAS-2 for the N-body simulations presented in chapter 5.

The GRAPE project, as described in section 1.3, started in the late eighties, and has produced a series of very high performance devices, mainly for the computation of the gravitational force. The GRAPE-4 system, completed in 1995, was the first computer to reach the TFlop/s peak speed (Makino *et al.*, 1997). The current peak performance of the latest machine, the GRAPE-6, is 63.6 TFlop/s (Makino *et al.*, 2002).

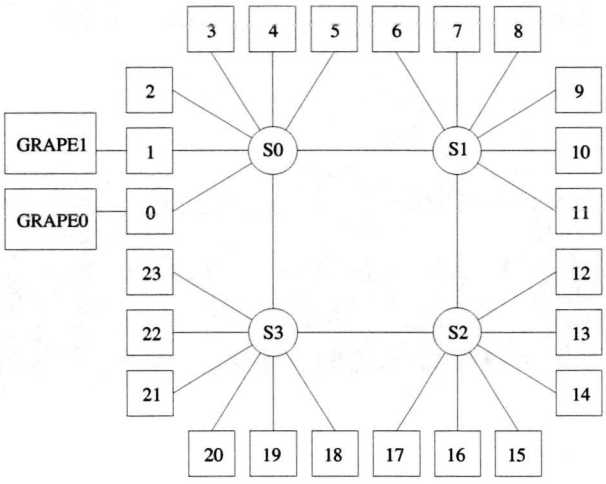


Figure 2.1: The DAS cluster at University of Amsterdam, including 24 processor nodes, two GRAPEs, and 4 network switches.

We study the performance of a system consisting of two GRAPE-4 boards, each one attached to a host processor via a PCI channel. The performance of a single GRAPE-4 board can reach 30 GFlop/s. A single board comprises an array of pipelines (up to 96 per board). Each pipeline performs, at each clock-cycle, the computation of the gravitational (or electrostatic) interaction between a pair of particles. The main technical characteristics of our system are summarised in table 2.1 below:

local network	Myrinet	150 MBytes/s peak-performance	40 μs latency
host	PPro 200 MHz	64 MB RAM	2.5 GB disk
GRAPE board	up to 320 MFlop/s per pipeline	62 resp. 94 pipelines	on-board memory for $\sim$ 44 000 particles
host-GRAPE channel	PCI9080	33 MHz clock	133 MBytes/s

Table 2.1: Technical data concerning our testbed architecture.

2.2.2 Application

The direct N-body method

A formal solution for the N-body problem is known only for $N = 2$, making a numerical approach necessary when a solution for a larger system is desired. As discussed in section 1.4, a range of techniques has been developed to implement a numerical solution for the N-body problem (Aarseth, 1999; Barnes & Hut, 1986; Cheng *et al.*, 1999; Hockney & Eastwood, 1988). We are concerned with the direct method, which computes gravitational interactions exactly, and with the treecode, which approximates this force evaluation, gaining in performance, at the cost of a lower accuracy. The treecode (Barnes & Hut, 1986) is able to reach a $\mathcal{O}(N \log N)$ scaling, compared to the $\mathcal{O}(N^2)$ scaling of the direct code. Other codes, as the FMM (Cheng *et al.*, 1999) of the Particle-Mesh (PM) code (Hockney & Eastwood, 1988), reach $\mathcal{O}(N)$ (see section 1.2). The FMM is routinely used in applications where the Coulomb force plays a central role. The PM code is primarily used in Computational Cosmology.

In chapter 3 we discuss our simulations of direct code and treecode performance on hybrid architectures. In section 3.4.4 we compare the direct code with two different parallel versions of the treecode. In the sequel we describe the main tasks of the direct code that we analyse in this chapter, i.e. NBODY1 (Aarseth, 1999), introduced in section 1.4.1 . The original serial code has been parallelised, and a number of modifications have been made, to obtain an optimal use of the GRAPE's capabilities, as described in section 2.3 below.

Code tasks

In the N-body computations, the particles that exert the force are commonly called j-particles, and the particles that experience the force are the i-particles. As discussed below, for each iteration, force is computed only on a small subset of particles, so that only a few particles are used as i-particles. On the other hand, since all particles in the system exert force, every particle plays the role of a j-particle, including the i-particles.

As mentioned in section 1.4.1, NBODY1 implements the *individual time step* scheme: particles experiencing a strong or rapidly changing force field need to be updated more frequently than particles moving through a quiet, nearly constant potential region. NBODY1 computes forces, and integrates orbits for each particle at the rate required by the particle dynamics itself. The individual time step is described in more technical detail in section 2.3 below.

A basic task graph of NBODY1 code-flow is given in fig. 2.2, together with the mapping of each task on the appropriate hardware resource, in case the GRAPE is available. If GRAPE is not available, all tasks are executed on the general purpose machine. The tasks shown in the figure are described in the following.

1. The first task of the main cycle is to find the i-particles. This is implemented as a search through a list of candidates, which is scrolled at each iteration, and rebuilt every `DTLIST` time units, where `DTLIST` is the average particle time step. The i-particles are selected simply by picking those particles that need to be updated first.

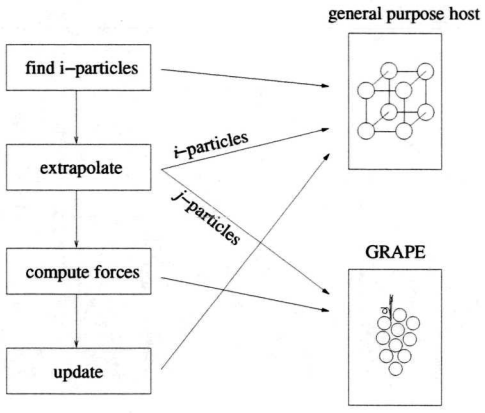


Figure 2.2: Basic task graph of NBODY1, and mapping of the tasks on the hardware resources. If the GRAPE is not available, all tasks are executed on the general purpose machine.

2. Then, since stored values of positions and velocities of different particles refer to different times because of the individual time step, an extrapolation of the position values for the entire set of particles is done, to "synchronise" the system to the time value of the i-particles. The GRAPE also contains a pipeline to perform the extrapolation of the j-particle positions (see fig. 1.4). Hence, when the GRAPE is available, this task is executed on it. Still, the host has to extrapolate the i-particle positions, therefore in fig. 2.2 the extrapolation task is mapped on the host for the i-particle extrapolation, and on the GRAPE for the j-particle extrapolation.

3. Now accelerations are computed; when the GRAPE is available, i-particle data are sent to it, and it will return the accelerations.

4. Finally, orbits are integrated using the forces computed in the previous task, and relevant physical quantities are evaluated and updated.

Code parallelisation

In the parallel application, we distributed the j-particles equally between two GRAPEs, i.e. we loaded the j-particle memory of each GRAPE with half of the particle set. All GRAPE hosts have a copy of the entire set of particles. Each SPD computes the partial force exerted on the i-particles by the j-particles that it stores; these values are then communicated to the host. A global sum done by the hosts makes the total force on each i-particle available to all processors, that finally integrate the i-particle orbits. When the GRAPE boards are not available, the algorithm works in a very similar fashion. In this case the j-particles are distributed by assigning each processor a different subset of particles, so that a processor will evaluate only forces exerted by its own j-particles. In section 2.3 below, we describe in detail the parallel codes that we studied.

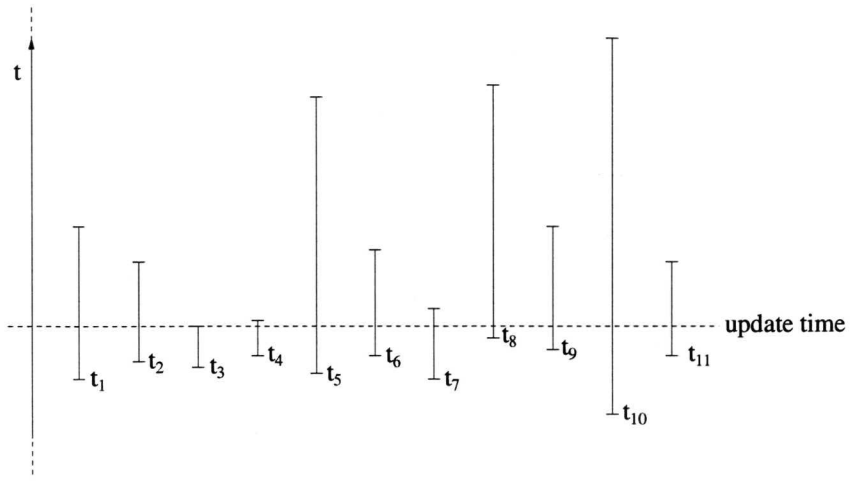


Figure 2.3: Sketch depicting the individual time step machinery. The update time is determined as the smallest $t_i + \Delta t_i$ (which in this figure is the value of particle #3). Particle positions are then extrapolated from t_i to the update time, in order to compute force on particle #3, and integrate its orbit to the update time. Finally, the new Δt for particle #3 is computed, and the next update time is determined. In the figure above, the next update time will probably be $t_4 + \Delta t_4$, unless the new Δt_3 is very small.

2.3 Code parallelisation

In this section, we describe the parallelisation of the various flavours of the direct N-body code used for our performance analysis and simulation. We chose NBODY1 (Aarseth, 1963) as the instantiation of a direct N-body code to experiment with, because it is a rather simple code, but uses almost all the functionalities of GRAPE. This allows us to evaluate the performance of our system. A number of modifications have been made to the code, in order to parallelise it, and to let it make full use of the functionalities of GRAPE.

An overview on the code is given below. We made use of MPI communication primitives (Message Passing Interface Forum, 1997) to parallelise it.

2.3.1 The basic: individual time-step

As already mentioned in section 1.4.1, NBODY1 uses individual time-steps. Each particle is assigned a different time at which the force will be computed. Fig. 2.3 depicts this procedure. The time-step value of each particle Δt, sketched in fig. 2.3 for each particle as a segment, depends on the particle dynamics (Aarseth, 1999). Smaller Δt values are assigned to particles having faster dynamics (i.e. those particles which have large values in the higher order time

derivatives of their acceleration) according to the formula (see, e.g., Aarseth, 2001)

$$\Delta t_i = \sqrt{\eta \frac{|\mathbf{a}_i||\ddot{\mathbf{a}}_i| + |\dot{\mathbf{a}}_i|^2}{|\dot{\mathbf{a}}_i||\dddot{\mathbf{a}}_i| + |\ddot{\mathbf{a}}_i|^2}} \ ,$$

where η is an accuracy parameter of order unity. At each iteration, the code selects that particle having the smallest $t + \Delta t$ value (particle 3 in fig. 2.3), and integrates only the orbit of that particle. This reduces the computational complexity, with respect to a code where a single global time step is used. The individual time step approach reduces the temporal complexity to $\mathcal{O}(N^{1/3})$, whereas the global time step approach is $\mathcal{O}(N^{2/3})$ (Makino & Hut, 1988).[1] This temporal complexity refers to the computational effort needed to integrate the system for a dynamical time, i.e. the average time taken by a particle to cross the system.

An effect of individual times is that, for each particle, values stored in memory refer to a different moment in time, i.e. the moment of the particle's last orbit integration. This means that, before force on particle i is computed, an extrapolation of the other particle positions to time $t_i + \Delta t_i$ is needed. The time value $t_i + \Delta t_i$ is marked in fig. 2.3 by the "update time" line.

Parallelisation

Since contributions to the gravity force on a given particle i are computed from all the other particles using eq. (1.1), regardless of their distance from i, a uniform distribution of particles to each processing element (PE), i.e. to each DAS node, suffices to assure load balancing. The force computation is done by broadcasting the coordinates of the currently selected particle i. Then each PE computes the partial component to the force on i, by accumulating contributions from its own particles. Finally such components are sent back to the PE which hosts i, where the force resultant is computed, the particle's orbit is integrated, and the new values are stored.

To identify the particle i on which force will be computed, a global reduction operation is done, in order to find which particle has the least $t_i + \Delta t_i$ value, and which PE owns it. This information is broadcast to all PEs, since they must know the extrapolation time, and the i-particle owner.

2.3.2 Towards a GRAPE code: block time-step

Since its introduction, NBODY1 has evolved to newer versions, which include several refinements and improvements (see, e.g., Aarseth, 1999). In the version of NBODY1 used in our study we implemented the so called *hierarchical block time-step* scheme (McMillan, 1986; Makino, 1991a). In this case, after computing the new Δt_i, the value actually assigned is the value of the largest power of 2 smaller than Δt_i. This allows for more than one particle to have the same Δt, which makes it possible to have many i-particles per time step, instead of

[1] These figures for the temporal complexity are valid for a uniformly distributed configuration. More realistic distributions show a more complicated dependence on N, although quantitatively only slightly different.

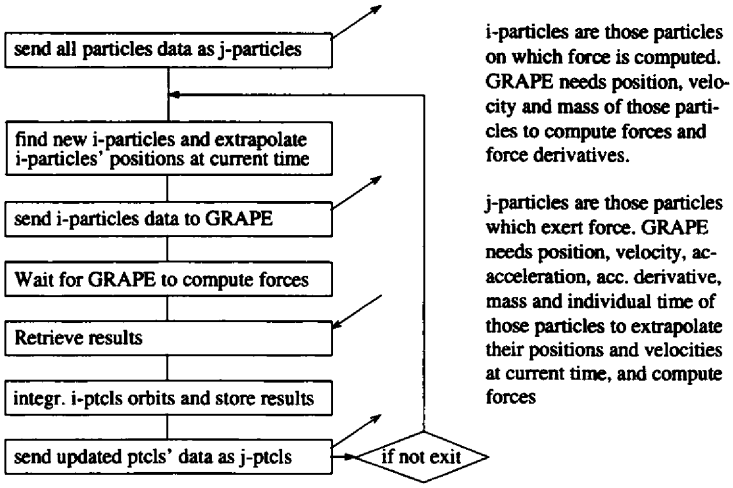


Figure 2.4: Basic sketch of NBODY1 interfaced with GRAPE. Diagonal arrows symbolise communication with GRAPE.

only one. Using this approach, force contributions on a, possibly large, number of i-particles can be computed in parallel using the same extrapolated positions for the force-exerting particles, hereafter called j-particles. Moreover, when a GRAPE device is available, it is possible to make full use of its array of pipelines, since each pipeline can compute the force on a different particle concurrently.

Parallelisation

Having many i-particles, instead of just one, makes it profitable to use a somewhat different parallel code structure. If the i-particles reside on different processors, distributing the particles as in the individual time-step case could result in complex communication patterns, with consequential increase of code complexity. Therefore, we chose to let every PE have a local copy of all particle data. The force computation is done in parallel by making each PE compute force contributions only from its own set of j-particles, assigned to it during initialisation. A global reduction operation sums up partial forces, and distributes the result to all PEs. Then each PE integrates the orbits of all i-particles, and stores results in its own memory. Concerning the search for i-particles, each PE searches only among its j-particles, to determine a set of i-particles candidates. Then a global reduction operation is performed on the union of these sets, in order to determine the real i-particles, i.e. those having the smallest time. The resulting set is scattered to all PEs for the force computation. Since every PE owns a local copy of all particle data, only a set of labels identifying the i-particles is scattered, reducing the communication time.

2.3.3 The GRAPE code

The software library interface for the GRAPE hardware consists of a number of function calls, the most relevant for performance analysis being those which involve communications of particles data to and from the GRAPE. Such communication operations include sending j-particle data to GRAPE, sending i-particle data to GRAPE, and receiving results from GRAPE. A sketch of the program flow for an N-body code which uses GRAPE is given in fig. 2.4.

Parallelisation

The presence of the GRAPE boards introduces a certain degree of complexity with respect to code parallelisation. The GRAPE-hosts obviously play a special role within the PEs set. This asymmetry somehow breaks the SPMD paradigm that parallel MPI programs are expected to comply with. Besides the asymmetry in the code structure, also the data distribution among PEs is no more symmetric. The force computation using GRAPE is performed, similarly to the non-GRAPE case, by assigning an equal number of j-particles to each GRAPE. The GRAPE computes the partial force exerted by the j-particles assigned to it on the i-particle set, which is the same for all GRAPEs. After that, a global sum on the partial results, performed on the parallel host, will finally give the total force.

Since force computations and j-particle position extrapolations are done on the GRAPE, the only relevant work to execute in parallel by the PE set is the search for i-particle candidates, which is accomplished exactly as in the code described in section 2.3.2 above.

2.4 Code performance

We describe and analyse in this section the measurements that we carried out for the performance evaluation of the codes described in section 2.3. Our measurements are intended to explore the scalability of parallel N-body codes. We performed runs varying both the number of particles N and the number of processors PEs; we scaled N from 1024 to 16 384, and PEs from 1 to 24. NBODY1 does not need a large amount of run-time memory, just about 200 bytes per particle, but is heavily compute-bound (Hut, 1996). Our timings were carried out in order to show the relative computational requirements of the various code tasks, and how these change as a function of N and PEs. Reported values are averages of the values measured for each processor. These measurements showed a negligible deviation, which is thus not reported on the figures below.

Our runs were started having a Plummer model distribution (Plummer, 1915) as initial condition, in which density decreases outward as the fifth power of the distance from the cluster centre. The gravity force is modified by introducing a *softening parameter*, which is a constant term, having the dimension of a length, whose squared value is inserted in the denominator of the gravity force expression eq. (1.1) (see also caption of fig. 1.1). The softening parameter reduces the strength of the force in case of close encounters and thus

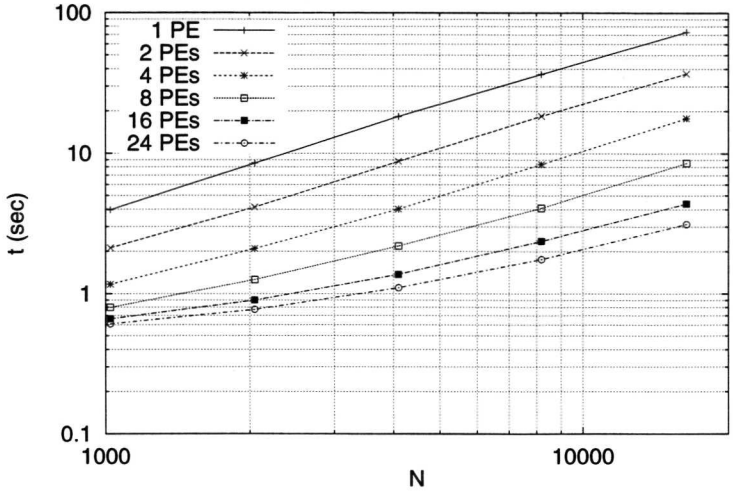


Figure 2.5: Global timings for the parallel individual time-step code, running for 1000 iterations.

prevents the formation of tightly-bound binaries. In this way very short time steps and correspondingly long simulation times are avoided. In our runs, this parameter was set equal to 0.004. As a reference, the mean inter-particle distance in the central core of the cluster, when $N = 16384$, is approximately equal to 0.037 in N-body units (Heggie & Mathieu, 1985).

2.4.1 Individual time step code

The essential tasks of this version of the code (hereafter called IND) are shown in the code flow sketched in fig. 2.2. As described in section 2.3.1, the parallel version of this code implements communications in the i-particle search task, then when the i-particle position is broadcast, and when partial forces are gathered by the PE that owns the i-particle. Fig. 2.5 shows the timings, and fig. 2.6 the performance of the parallel version of the IND code.

The metric we use to quantify the code performance is the parallel efficiency, defined as:

$$P_n = \frac{t_1}{n \cdot t_n}$$

where n is the number of PEs used, and t_n the execution time when using n PEs. The timings shown in the figures refer to 1000 iterations of the code. The t_n values depend about linearly on N, since the number of operations to compute the force on a given particle scales linearly with N, and in each run the same number of force computations is performed, i.e. 1000, independently of the total number of particles. An interesting super-linear speedup is visible in fig. 2.6, which can be explained with an optimised cache utilisation. The IND code, when the work-load is high ($N \geq 8192$), is highly compute-intense, as fig. 2.8 clearly shows.

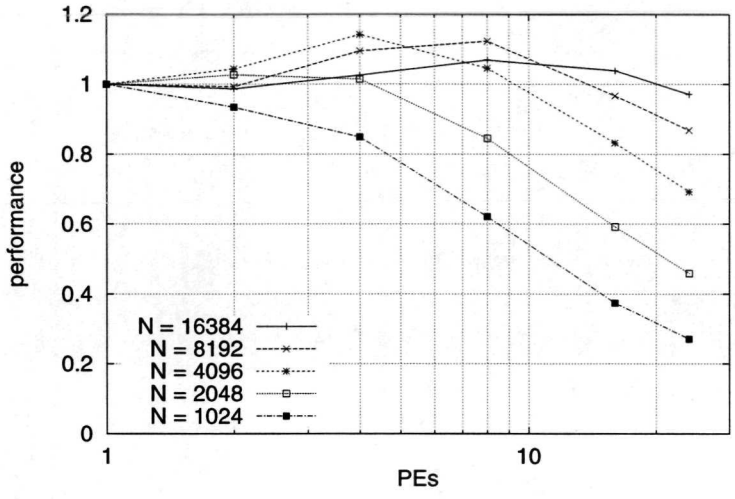


Figure 2.6: Performance of the parallel individual time-step code, running for 1000 iterations. One iteration consists of advancing a single particle per time step. A super-linear speedup effect, discussed in the main text, can be seen for intermediate values of the work-load per processor.

In this case, also when the number of processors is high, thus with relative small number of particles per processor, the communication overhead is still small. Since the number of particles per processor decreases as PEs increases, the number of cache misses decreases too, thus the cache is better exploited as PEs increases. This effect, combined with the limited importance of the communication overhead for the high workload cases, leads to the super-linear speedup visible in fig. 2.6.

Fig. 2.7 and 2.8 show the fractional time shares of each task, and how these shares change as the number of PEs changes. Fig. 2.7 shows the time shares for runs with $N = 1024$, and fig. 2.8 for runs with $N = 16\,384$. Fig. 2.7 clearly shows how the IND code suffers from a communication overhead when the computational work-load is light. On the other hand, as shown in fig. 2.8, the code performs quite satisfactorily when this ratio is high, thanks to the compute-intense characteristics of the N-body code, and the high performance communication network of our architecture.

2.4.2 Block Time-step Code

The basic tasks of this version of the code (BLOCK hereafter) are the same as the IND code. The only difference is that now the number of i-particles per iteration can be larger than one. As stated in section 2.3.2, this optimises the force computation procedure, also in view of the use of GRAPE, but, on the other hand, increases the communication traffic, since information about many more particles must be exchanged at each time step.

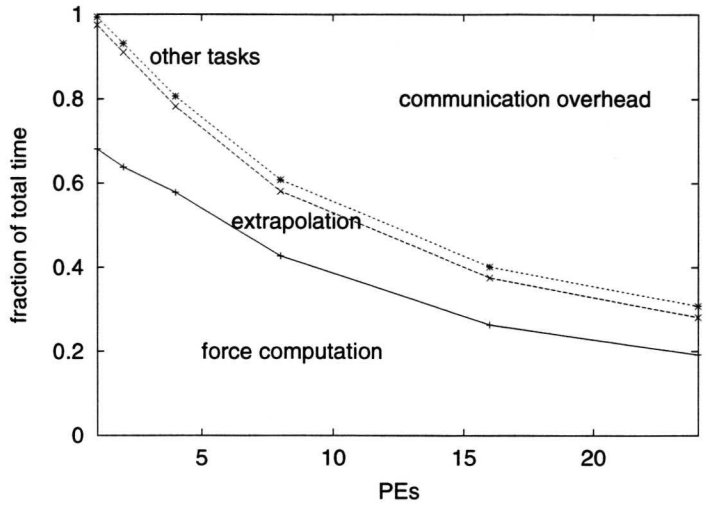


Figure 2.7: Execution time shares vs number of processors for the IND code. Runs with 1024 particles.

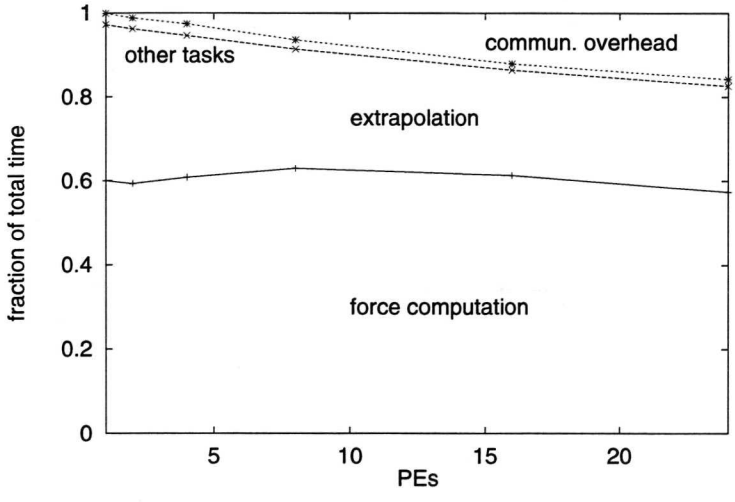


Figure 2.8: Execution time shares vs number of processors for the IND code. Runs with 16384 particles.

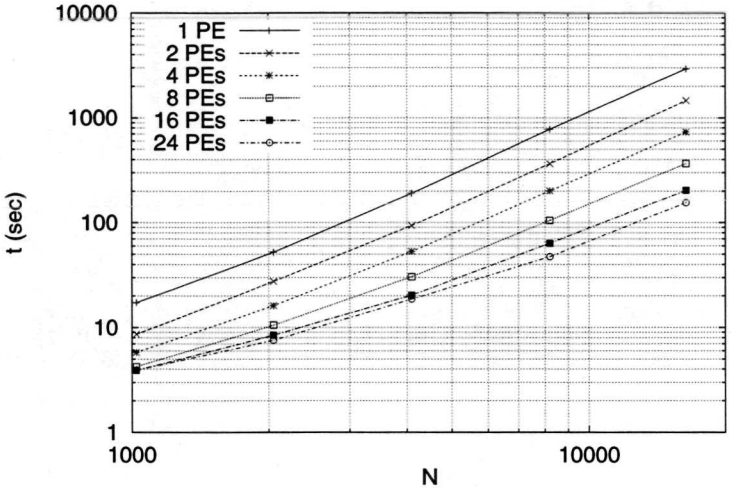


Figure 2.9: Global timings for the parallel block time-step code. In this case, at each time step, force is computed on many particles.

N	$\langle N_i \rangle$
1024	35.05
2048	43.91
4096	111.16
8192	207.52
16384	351.14

Table 2.2: Mean number of i-particles per iteration in the BLOCK code runs.

The effect of this is clearly shown in the figures presented here. Fig. 2.9 shows total timings, and fig. 2.10 shows performance of this code. In this case the execution time grows as a function of N^2 because the number of i-particles, i.e. the number of force computations, grows approximately linearly with N. Since the computational cost for the force on each particle also grows linearly with N, the resulting total cost is $\mathcal{O}(N^2)$. The mean number of force computations per iteration as a function of N is given in table 2.2. Fig. 2.10 shows how the performance gain of this code is less spectacular than the gain of the IND code, since communication overhead plays a larger role in the total execution time. This large overhead can be seen in fig. 2.11 and 2.12, that show how the execution time shares evolve as a function of PEs number. These figures show that for the BLOCK code, almost all the computational part of the execution time is spent in the force computation task. The j-particles extrapolation, that takes roughly 25% to 30% of the total time in the IND code (see figures 2.7 and 2.8), is reduced to less than one percent.

2.4.3 GRAPE Code

The code version which makes use of GRAPE boards will be called GRP hereafter. We present performance results of both the serial, and the parallel implementation. The communication overhead of the parallel version is composed of host-GRAPE communication and network communications. The parallel code runs have been done by using only the DAS

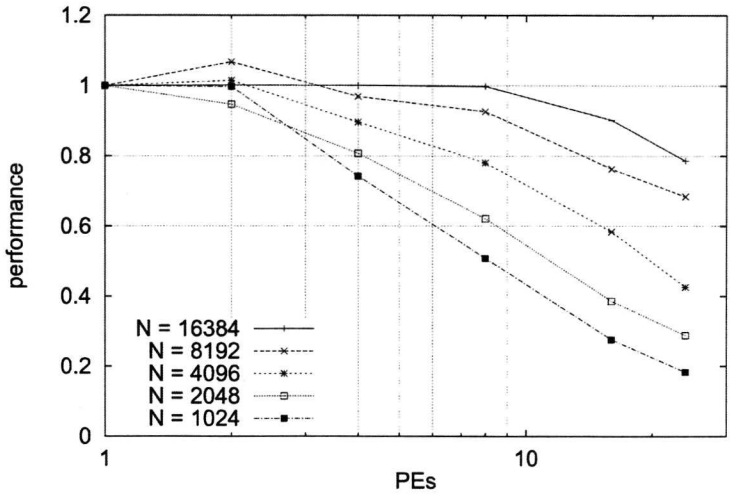


Figure 2.10: Performance of the parallel block time-step code.

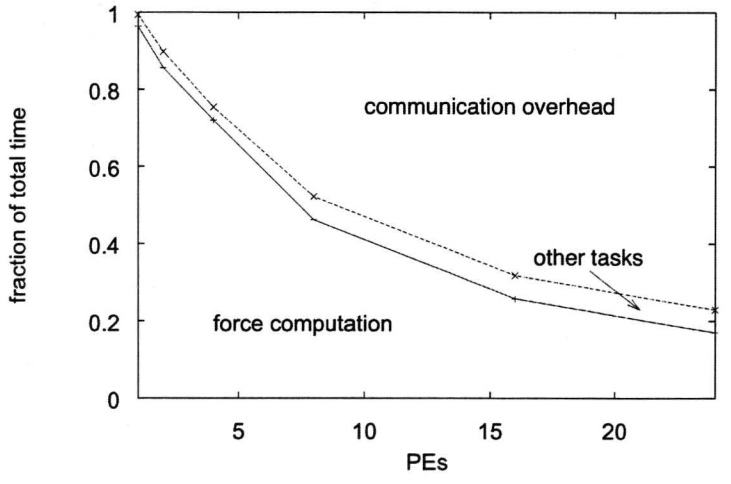


Figure 2.11: Execution time shares vs number of processors for the BLOCK code. Runs with 1024 particles.

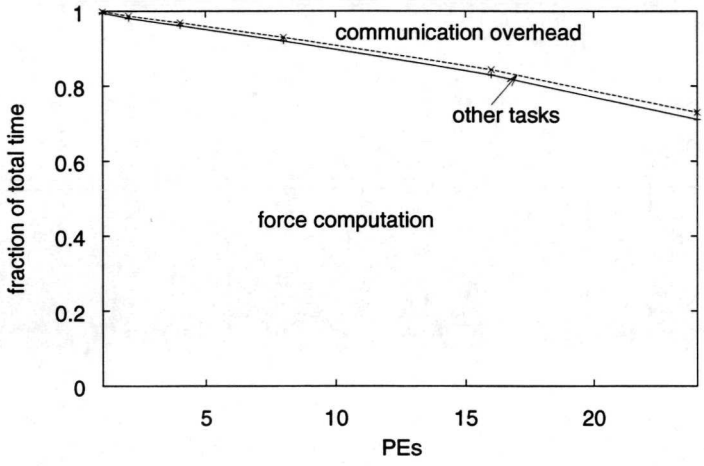


Figure 2.12: Execution time shares vs number of processors for the BLOCK code. Runs with 16 384 particles.

nodes connected to the GRAPE boards at our disposal, thus the maximum number of PEs in this case is 2.

It is clear from fig. 2.13 that the parallel performance is very poor. In that figure, GRAPE0 refers to the GRAPE with 62 pipelines, and GRAPE1 to the GRAPE with 94 pipelines. Fig. 2.13 also shows that runs on GRAPE1 are a bit faster, thanks to the larger number of pipelines available. The low parallel performance shown in fig. 2.13 can be explained by the low number of i-particles, especially for the low-N runs, that prevents the GRAPE boards to be fully exploited. Moreover, a large communication overhead dominates the GRP code, as fig. 2.14 for the GRAPE0 case, fig. 2.15 for the GRAPE1 case, and 2.16 for the parallel case show. These figures also show that the time share spent in GRAPE computations (i.e. force computations) is quite low, resulting in a low efficiency of this code in terms of GRAPE exploitation. One reason for that is of course the very high speed of the GRAPE. This device is by far faster in accomplishing its task than its host and the communication link between them.

Another effect that can be seen in the figures is the increase of the time share of the orbit integration task when N goes from 2048 to 4192. This can be explained by the increase of cache misses when this task is executed. The cache size of a node is 256 Kbytes, which makes it able to contain data for about 1000 particles (each particle carries about 200 bytes of data). The orbit integration task works on data which are located randomly on the memory, thus the chance of a cache miss when the cache does not contain the whole data set is relatively high. This effect produces the increase of the orbit integration time share between $N = 2048$ and $N = 4192$. Subsequently, the timings are more and more dominated by the increase of the GRAPE computation time share.

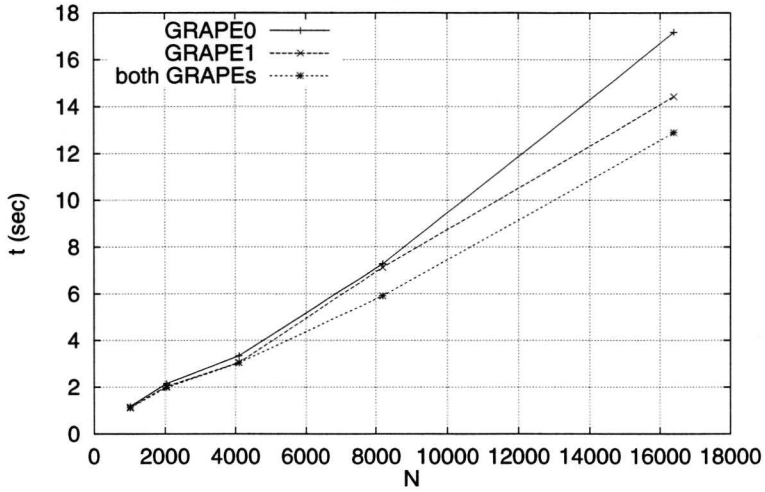


Figure 2.13: Execution time for 1000 iterations of the GRP code.

The figures clearly show that for our hardware configuration the capabilities of the GRAPE will only be fully utilised for problems involving over 40 000 particles per GRAPE. This number is, however, limited by the GRAPE on-board memory for j-particles, which is only slightly higher than 40 000.

Our measurements of the low level host-GRAPE communication routines show that a large amount of time spent in communication is due to software overhead in copy operations and format conversions. As an example, we show in fig. 2.17 measurements done on the j-particle send operation. Similar measurements (Kawai *et al.*, 1997), performed on a faster host, showed a higher communication speed, linearly dependent on the host processor clock speed. Nevertheless, even though the GRAPE boards are not exploited optimally, the execution times for the GRP code are by far shorter than those for the BLOCK code. The heaviest run on 2 GRAPEs is about one order of magnitude faster than the heaviest run of the BLOCK code on 24 PEs. Considering the total amount of computing power used in these two cases, i.e. the execution time times the number of processors used, shows that the BLOCK code needs about 140 times more computing time to perform the same amount of work as the GRP code. A global comparison of the throughput of all codes studied here is given in section 2.4.4 below.

2.4.4 Code Comparison

In order to evaluate the relative performance of the three versions of the N-body code studied in this chapter, a series of runs has been made, where both a 8192 particles system, and a 32 768 particles system were simulated for 7200 seconds. We compare the performance of

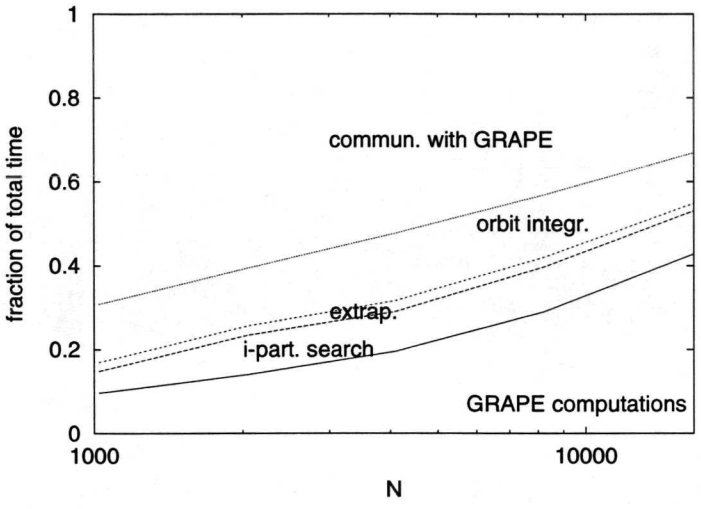


Figure 2.14: Execution time shares vs number of particles for the GRP code. Runs on GRAPE0 (62 pipelines).

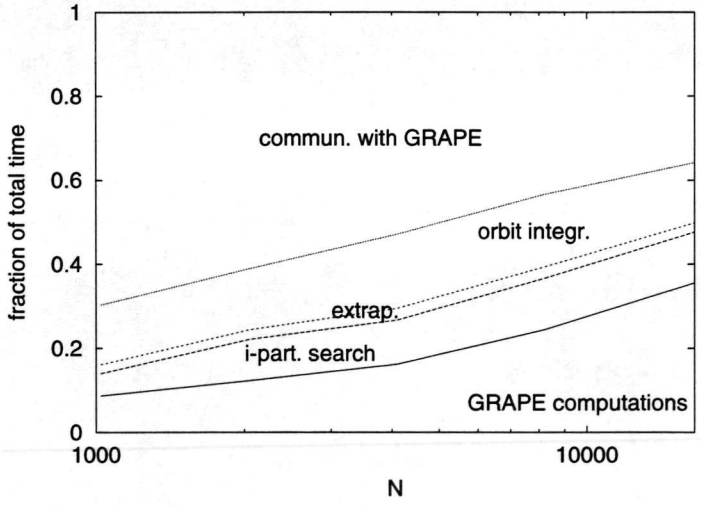


Figure 2.15: Execution time shares vs number of particles for the GRP code. Runs on GRAPE1 (94 pipelines).

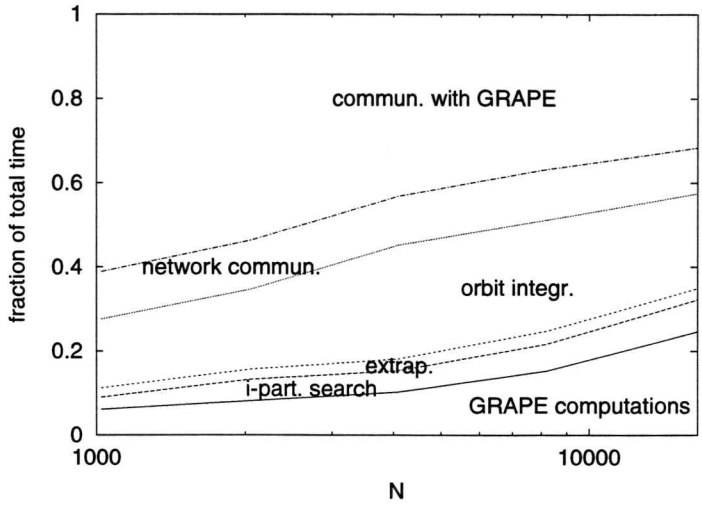


Figure 2.16: Execution time shares vs number of particles for the GRP code. Runs on both GRAPEs.

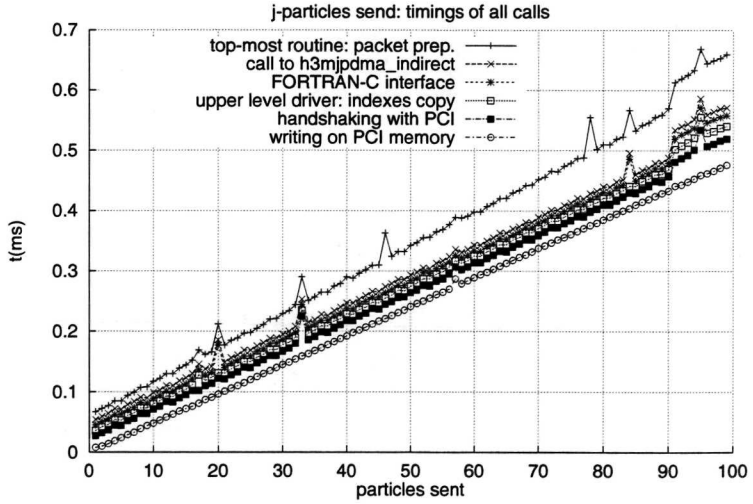


Figure 2.17: Software overhead in the j-particle send operation. The difference between the top-most timing (the cumulative task timing) and the timing immediately below is due to format conversion. The other differences are mainly due to copy operations.

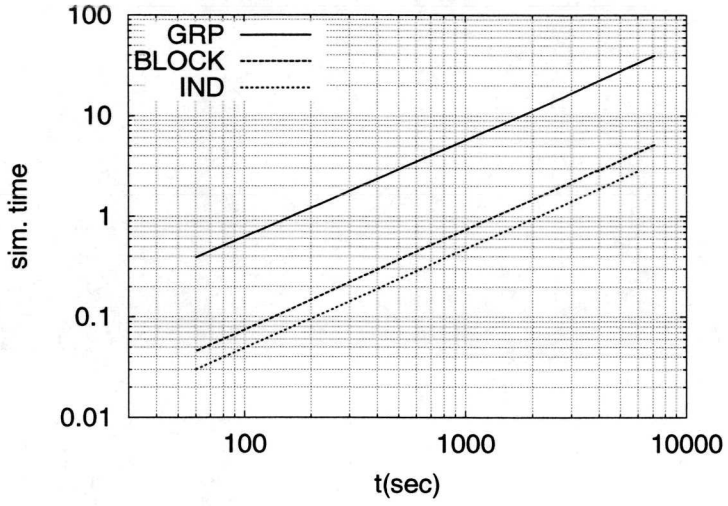


Figure 2.18: Performance comparison for the three versions of the N-body code. Runs with 8192 particles. The IND and BLOCK codes are run on 24 processors, the GRP code is run on two processors each connected to a GRAPE.

the GRP code, with respect to the other codes run on the general purpose host, against an increasing computational load. The fastest hardware configuration is used in each case, i.e. 24 PEs for the IND and BLOCK code runs, and 2 PEs (and hence 2 GRAPEs) for the GRP run. Fig. 2.18 and 2.19 show the evolution of the simulation time, as a function of the wallclock time. In this way, the performance of each code is specified in terms of how long one should wait before a simulation reaches a certain N-body time. Those figures show that the GRP code outperforms the other two codes by a factor 8, when the computational load is lighter, and by a factor 20, with a heavier computational load. In both cases, the BLOCK code is 1.5 times faster than the IND code, thanks to the optimisation of the j-particles extrapolation step. Fig. 2.19 shows an initial overlapping of these two codes performance curves, due to a start-up phase, which is not visible in fig. 2.18, because at the first timing event (after 60 s) this system is already stabilised.

Fig. 2.18 and 2.19 clearly show the large performance gain obtained with GRAPE. Using only two PEs, an order of magnitude better performance was attained compared to the BLOCK code on 24 PEs. Due to the reduction in the time needed for the force calculation, the communication overhead for the GRP code accounts for approximately 50% of the total execution time (see fig. 2.15 and 2.16). Hence an even larger relative gain may be expected for larger problems, as the relative weight of the communication overhead will decrease. The difference in performance between the two cases shown respectively in fig. 2.18 and 2.19 clearly illustrates this effect.

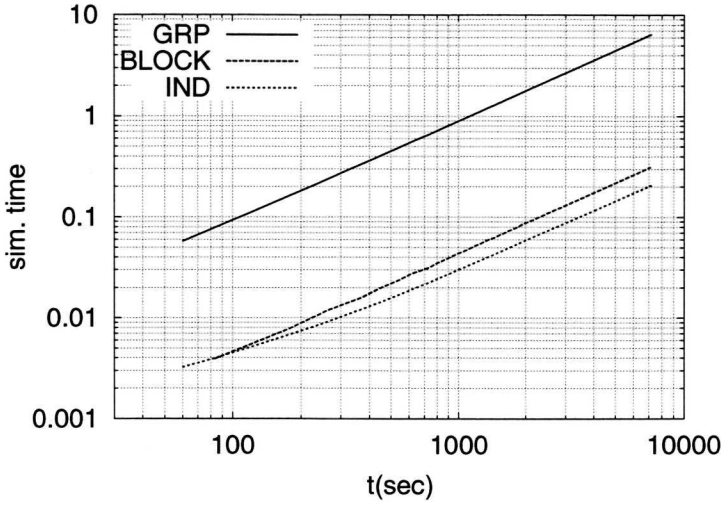


Figure 2.19: Same as fig. 2.18. Runs with 32 768 particles.

2.5 Discussion

Our performance analysis reveals a very good parallel performance of the BLOCK and especially the IND code. We also show that the use of GRAPE leads to a dramatic performance gain, even at a low efficiency in terms of GRAPE boards exploitation. Such low efficiency is mainly due to a very high communication overhead, even for the largest problem studied. This overhead can be greatly reduced by the use of a faster host, and by the development of an interface requiring fewer format conversions. The GRAPE-hosts in the system studied in this chapter have a 200 MHz clock speed. Nowadays standard clock speeds are up to one order of magnitude faster. The use of a state-of-the-art processor would reduce the host and communication times significantly. The low utilisation of GRAPE, shown in fig. 2.14, 2.15 and 2.16, suggests that the problem size has to be increased to attain a optimal SPD utilisation.

The measurements described in this chapter are the basis for the calibration and validation of our performance simulation model. In chapter 3 our model will be described, and used to simulate different classes of N-body codes, running on different hybrid architectures.

Chapter 3

Modelling and Simulation of Hybrid Architectures[†]

3.1 Introduction

In this chapter the performance model that we developed to simulate the behaviour of hybrid architectures is introduced. Hybrid architectures were presented in section 1.3.4 as systems where a high performance general purpose computer is coupled to one or more Special Purpose Devices (SPDs). They can be seen as a special case of computer systems described by the heterogeneous computing paradigm (Freund & Siegel, 1996; Palazzari *et al.*, 2000). In section 1.3.4 we also discussed why such a system can be the optimal choice for several fields of Computational Science. The relevance of the GRAPE in the field of Numerical Astrophysics has been discussed in section 1.3. Quantum Chromodynamics is another field that has benefited substantially from the use of SPDs. In a recent review on Computational Quantum Chromodynamics the state of the art for the use of dedicated computers in that field has been presented, with the Japanese CP-PACS computer (Aoki *et al.*, 1999), the

[†]This chapter is based on work published in:

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *A Simulator for Parallel Hybrid Computer Systems*, in R.L. Lagendijk; J.W.J. Heijnsdijk; A.D. Pimentel and M.H.F. Wilkinson, editors, Proceedings of the seventh annual conference of the Advanced School for Computing and Imaging, pp. 210-219. ASCI, 2001.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Prediction of N-body Simulations on a Hybrid Architecture*, Computer Physics Communications, **139**, 34–44, 2001.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *A Versatile Simulation Model for Hierarchical Treecodes*, in P.M.A. Sloot; C.J.K. Tan; J.J. Dongarra and A.G. Hoekstra, editors, Proceedings of the ICCS2002 Conference, LNCS vol. 2329, pp. 176–185. Springer Verlag, 2002.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Modelling of Hierarchical N-body Codes Running on Hybrid Architectures*, in E.F. Deprettere; A.S.Z. Belloum; J.W.J. Heijnsdijk and F. van der Stappen, editors, Proceedings of the eighth annual conference of the Advanced School for Computing and Imaging, pp. 211–218. ASCI, 2002.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Modelling of Distributed Hybrid Architectures*, IEEE Transactions on Parallel and Distributed Systems, in press, 2003.

QCDSP machine built in the USA (Mawhinney, 1999), and the APE system developed in Europe (Tripiccione, 1999).

Configuring a hybrid system and finding the optimal mapping of the application tasks onto the hybrid machine often is not straightforward. Performance modelling, which we discussed in section 1.6, provides a tool to tackle and solve these problems. We developed a performance model to simulate a hybrid architecture consisting of a parallel multiprocessor where some nodes are the host of a GRAPE board. GRAPE, introduced in section 1.3, is a very high performance SPD used in Computational Astrophysics.

We present here the general modelling framework, and the methodological approach that we used to build our model. Based on this modelling background, and on the experimental data presented in chapter 2, we developed the performance models described in this chapter. We present here the details of the implementation of both the model used for the simulation of the direct code, introduced in section 1.4.1 and discussed in detail in section 2.2.2, and the model for the simulation of the treecode, introduced in section 1.4.2. We validate the accuracy and versatility of our models by simulating existing configurations, and use them to forecast the performance of other architectures, in order to assess the optimal hardware-software configuration.

Forecasting and analysing the performance of a hybrid architecture is not trivial. Performance modelling can provide a solution to this problem. The range of applications of performance modelling in Computer Science is vast. Recently, in a review of performance modelling research, applications were presented spanning the range from scheduling in global computing systems (Aida *et al.*, 2000) to modelling of large-scale scientific applications (Adve & Sakellariou, 2000), based on both the analytical approach (e.g., Gunther, 2000; Hoisie *et al.*, 2000) and on simulation (e.g., Kurc *et al.*, 2000).

Analytic models (see, e.g., Cremonesi & Gennaro, 2002) easily become intractable due to the complexity of the simulated system, and usually show a limited flexibility. Simulation models (Bagrodia *et al.*, 1998; Adve *et al.*, 2000) allow for the study of very complex systems. Their high degree of versatility makes it possible to estimate the performance of hardware or software architectures during the various phases of their development (see, e.g., Pimentel *et al.*, 2001).

We have built a performance model, based on functional task modelling (Dikaiakos *et al.*, 1996). Our model simulates the behaviour of a parallel multiprocessor, where specific nodes can act as the host of an SPD. This helps us to understand the interactions between the SPD, the host, and the application that is run on the hybrid system. Our aim is to have the possibility to adapt and modify the hardware model, in order to find the configuration that gives the best performance, and to simulate a different software application just by changing the higher level software specifications. Hence we developed a model able to make predictions of the performance of the system for a given algorithm, and to tell us how hardware and software can be adapted to one another to obtain the best performance.

The hybrid system that we used to validate our model consists of a local cluster of the DAS parallel computer (Bal *et al.*, 2000), where two nodes are the host of a GRAPE board. A direct summation N-body code (Aarseth, 1999) that, as described in chapter 2,

we parallelised and adapted for use with the GRAPE, was executed on this system. We validated our performance model on this architecture, and used the model to make predictions on the system behaviour, when both hardware and software modifications are introduced. Furthermore, we also studied the behaviour of a treecode (Barnes & Hut, 1986) on such a system.

3.2 Design considerations

3.2.1 Requirements

Hybrid Architectures can be complex to design, and expensive to realise. Performance modelling is an effective tool to estimate their performance rapidly and inexpensively. We aimed to build a versatile model, able to simulate different applications running on different computer architectures. Therefore, we have structured our model so as to separate the modelling of the hardware from the modelling of the algorithm. This allows us to modify the model of the application, leaving intact the underlying model of the machine, and vice versa.

Scope of our model

Generally, performance models are designed to simulate an application or a hardware architecture in great detail, and need powerful simulation environments, such as POEMS (Adve *et al.*, 2000), a comprehensive environment for the study of complex computer systems, or Artemis (Pimentel *et al.*, 2001), specifically developed for embedded systems design and analysis. In our case, we do not aim at simulating our application down to the single instruction level, or our machine at the single electronic component level. We focus on the interaction of the SPD with the parallel host, and the interplay of those two components with the application executed on them. We use an iterative refinement approach, starting coarse, and, if necessary, refining those modules that produce unacceptable errors. For this purpose, we found it sufficient to model the system components at a functional level (Dikaiakos *et al.*, 1996). This approach involves much less model complexity, still giving us sufficiently accurate results.

Level of granularity

The level of granularity of our model is dictated by the accuracy that we want to reach in our simulations, taking into account that the aim of our performance analysis research is the optimal performance of the software application, typically achieved by balancing the hardware components' workload for a given set of software application tasks. As shown below, we get a satisfactory accuracy with a rather "coarse grained" functional model. The basic unit of our abstract algorithm is the task, defined as a code block which encompasses a set of instructions performing a specific operation. This operation is characterised by having a non negligible execution time, accessing a set of resources which is constant in time, and

depending on a limited number of application parameters. Similarly, the model granularity for the architectural components has been set at the level where the atomic units are the major computing elements, such as the SPD and the host node.

Model structure

The computational environment that we model is specified, at the more abstract level, by a number of formal entities. These are the algorithm, the hybrid machine, and the mapping interface. In the specific case described here, the algorithm computes the numerical solution of the gravitational N-body problem. The algorithm model generates simulation parameters, and activates basic operations. The different operations of this code have different demands for computational power, the force computation being by far the most demanding task. The design of the hybrid architecture on which this algorithm is executed matches these requirements, by including a model of a specialised hardware for the gravitational force evaluation. A sequence of tasks describes the behaviour of each component, and the concurrent access to machine components by a set of application tasks is treated as a critical section.

3.2.2 Functional model and implementation environment

Our functional model approach has been presented in section 1.6, where we described how we identify the main constituents of the modelling environment. We make a model of the software application in the *application model*, where we specify the time spent on each task $\mathcal{T}_i$ as a function of the application parameters π_i. Similarly, in the *machine model*, the characteristics of the hardware resources $\mathcal{R}_j$ depend on the machine parameters μ_i. The *mapping interface* maps each $\mathcal{T}_i$ of the application model on the appropriate $\mathcal{R}_j$ of the machine model. The resulting *simulation model* returns the simulated execution time, which depends on both the π_i and μ_i. In this way, we can study the performance of existing systems, and forecast the performance of the systems under design.

The simulation language used to implement our model is PAMELA (PerformAnce Modeling LAnguage) (van Gemund, 1993, 2003), developed by Arjan van Gemund at the Delft University of Technology, aimed at either simulation or analytic performance analysis. PAMELA is a C-style procedure-oriented simulation language where a number of operators model the basic features of a set of concurrent processes. In a *procedure-oriented* language, concurrent process interaction takes place via shared variables, in contrast to *message-oriented* languages, which describe communications in terms of explicit messages between interacting processes.

The execution time of a process is modelled by the `delay` statement; the sequential execution of processes is implemented by the `seq` (prefix) or `;` (infix) construct. Parallelism is specified by means of the `par` (prefix) and `||` (infix) constructs, which are implemented in a fork/join fashion (i.e. with implicit synchronisation). Explicit synchronisation between a couple of processes is implemented with the `wait` and `signal` operators, while mutual exclusion is realised with the `P` and `V` semaphore statements, which implement Dijkstra's classical solution to the resource contention problem (see, e.g., Tanenbaum, 2001, § 2.3.5).

PAMELA models the execution of processes in terms of the Discrete Event Simulation paradigm, as the use of the `delay` primitive suggests. A model can be *material-oriented*, when the execution flow of the process is specified in terms of the various system resources that the process will access, or *machine-oriented*, where the emphasis is on the resource, with a specification of the series of operations that each resource should accomplish. PAMELA is more suited for the first procedural approach, although machine-oriented models can also be built within this framework.

In order to show how PAMELA is typically used to describe a parallel system, we give as an example the model of a client-server system, where C concurrent clients execute N iterations each. An iteration consists of local processing with duration τ_l, followed by a request to access the server s which, once accessed, is used for a time τ_s. Such system is modelled by:

```
par (p = 1 ... C)
    seq (i = 1 ... N)
        { delay(τ_l) ;  P(s) ; delay(τ_s) ; V(s)  }
```

where line breaks and indentations are used for the sake of clarity, and have no syntactic meaning. A detailed overview on PAMELA is given in van Gemund (1993).

3.3 Model implementation

In this section we describe how our performance model reproduces the tasks of the codes under study. The direct code tasks have been described in section 2.2.2. Here we specify the dependence of the execution time of each task on the application parameters, like N_i, the number of i-particles, and N_j, the number of j-particles.[1]

3.3.1 Direct code

Our application model for the direct code is modelled as a sequence of tasks, as sketched in fig. 3.1. Each computation task is implemented by a `delay` statement (see section 3.2.2), possibly followed by a support function that sets the value of time dependent parameters, as N_i and N_j. As described in section 2.3, there are communication operations at the end of the i-particle search and force compute tasks, and both are global all-to-all operations. They are implemented in the model by means of a synchronisation operation, followed by a `delay` statement. The delays in the model of each task depend on the model parameters according to the formulae reported in table 3.1. These expressions have been obtained by analysing the data presented in chapter 2, and inferring the dependence of the execution time of each task on the model parameters. In the following section we describe the subset $\mathcal{P}_i$ of application parameters that affect the performance of each task $\mathcal{T}_i$.

[1]We recall that i-particles are the particles *on* which force is computed, whereas j-particles are the particles *from* which force is computed.

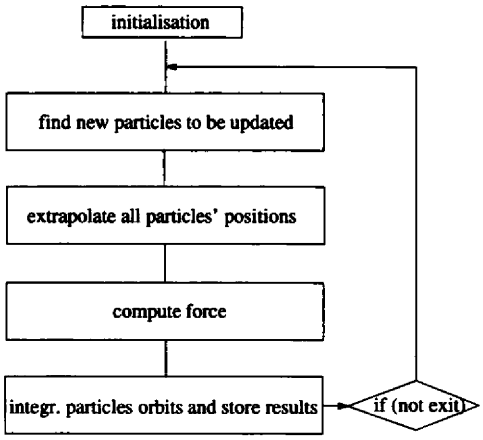


Figure 3.1: Basic sketch of the direct N-body code tasks.

tasks	**modelling formulae**
i-particle local search	$\mu_P \cdot \pi_{src} \cdot N/P$
i-particle global communication	$\mu_L \cdot \pi_{igl} + \mu_L \cdot P + (\mu_L \cdot \pi_{inl} + \mu_B \cdot \pi_{ib} \cdot \log(P)) \cdot N_i$
extrapolation on host (non-GRAPE case)	$\mu_P \cdot \pi_{xtr} \cdot N/P$
extrapolation on host (GRAPE case)	$\mu_P \cdot \pi_{xtr} \cdot N_i$
local force (non-GRAPE case)	$\mu_P \cdot \pi_{frc} \cdot N_i \cdot N/P$
local force (GRAPE case)	$(\mu_S + \mu_G \cdot N/G) \cdot \left\lceil \frac{N_i}{n_{pipes}} \right\rceil$
j-particle send to GRAPE	$\mu_P \cdot \pi_{prep} \cdot \left\lceil \frac{N_j}{90} \right\rceil + \mu_C \cdot \pi_{jpart} \cdot N_j$
i-particle send to GRAPE	$\mu_C \cdot \pi_{ipart} \cdot \left\lceil \frac{N_i}{n_{pipes}} \right\rceil \cdot n_{pipes}$
receive results from GRAPE	$\mu_C \cdot \pi_{res} \cdot \left\lceil \frac{N_i}{n_{pipes}} \right\rceil \cdot p_{max}$
force global communication	$(\mu_L \cdot \pi_{fl} + \mu_B \cdot \pi_{fb} \cdot \log(P)) \cdot N_i$
orbit integration	$\mu_P \cdot \pi_{orb} \cdot N_i$

Table 3.1: Synopsis of the application tasks, and the modelling formulae for their time dependence on the model parameters, whose values are given in table 3.2. Here, G is the total number of GRAPE boards, n_{pipes} is the number of pipelines in a GRAPE board, p_{max} is the maximal number of pipelines in a GRAPE board; for the GRAPE-4 $p_{max} = 96$ (see section 1.3 for details). The other variables are defined in the text.

Application model

For an N-body code, the most important parameter is obviously N, the total number of particles, which is a measure of the problem size. Moreover, the dynamical parameter that affects the performance of each task in a block time step code is N_i, the number of particles for which force is going to be computed. We observed a highly oscillatory behaviour for this parameter, shown in fig. 3.2. This oscillation of N_i between high and low values can be due to a small number of binary stars, which have a strong mutual interaction, requiring a small time step, or to close encounters between pairs of stars. The high occurence of low values of N_i between iteration #60 and iteration #300, implying that one or two particles evolve with a low time step, is an indication of the presence of a binary system in that simulation. The number of particles having a larger time step is also larger; when they are selected as i-particles, the value of N_i becomes much higher. We give the value of N_i at each iteration, obtained from the trace of real runs, as an input to our simulator.

i-particle search. The task of finding the N_i particles is modelled as a linear function of N, since the search is done over a set of candidates, whose number is a nearly constant fraction of N. In the parallel case, each processor searches a local list of candidates, which is a subset of the local particle set. The actual i-particles are chosen after this local search is completed, again by selecting from the candidates those particles with the smallest time value.

This global search uses a collective communication. The measured communication time shows both a linear dependence on the number of processors P, and on N_i. The N_i scaling factor is modulated by a term proportional to $\log P$. Based on our measurements, we used the fitting formula given in fig. 3.3 to model the global search task.

Fig. 3.3 shows the dependence of this task on N_i, for three different representative sets of values for N and P. A data point in this graph is the average value of the timings on each processor at a given iteration of the code. Occasionally, values much higher than the average have been measured, as shown in the figure, arguably due to external data traffic in the network. The fitting formula is not affected by these spurious values.

Extrapolation. The extrapolation phase, in the non-GRAPE case, consists of a fixed number of operations done on every particle in the system. Each processor extrapolates only its own j-particle positions, thus the extrapolation time shows a linear dependence on N/P, i.e. the workload per processor. A sketch of the dependence of the execution time for this task on N/P is given in fig. 3.4. Each point here is the average value over the entire run, for a given pair (N, P). This figure shows a jump in the dependence of t on N/P, due to a cache effect.[2] We chose to model only the out-of-cache behaviour, because we are more interested in situations characterised by a large workload.

[2]The cache size of our system is 256 Kbytes per processor, and each particle carries about 200 bytes of data. Then a workload per processor larger than about 1000 particles will cause the problem to run out of cache.

machine parameters		
μ_P	(μs to perform a processor cycle)	1/200
μ_L	(network latency in μs)	40
μ_B	(μs to transmit a byte over the network)	1/150
μ_S	(μs to startup the GRAPE pipeline)	75.6
μ_G	(μs for the GRAPE to compute a force interaction)	0.19
μ_C	(μs to transmit a byte on the GRAPE-host channel)	1/133

application parameters								
computations			network communications			host-GRAPE communications		
π_{src}	(i-particle search)	54	π_{igl}	(i-particle commum. startup)	3.5	π_{jpart}	(j-part. send to GRAPE)	80
π_{xtr}	(part. pos. extrapolation)	260	π_{inl}	(i-particle transmission)	0.0025	π_{ipart}	(i-part. send to GRAPE)	40
π_{frc}	(force computation)	260	π_{fl}	(force data transmission)	0.5	π_{rec}	(receive results from GRAPE)	130
π_{orb}	(orbit integration)	420	π_{ib}	(i-particle broadcast)	22.5			
π_{prep}	(format conversions in packet preparation)	480	π_{fb}	(force broadcast)	105			

Table 3.2: Values of the performance parameters appearing in the modelling formulae in table 3.1.

When GRAPE is used, the application code has to perform the extrapolation only for the i-particles. The GRAPE contains an extrapolation pipeline for the j-particles, but it does not extrapolate the i-particle positions; the host must perform this operation. Hence, in this case, the extrapolation task is modelled as a linear function of N_i, and is mapped on the host. Since every host must extrapolate all the i-particle positions, there is no dependence

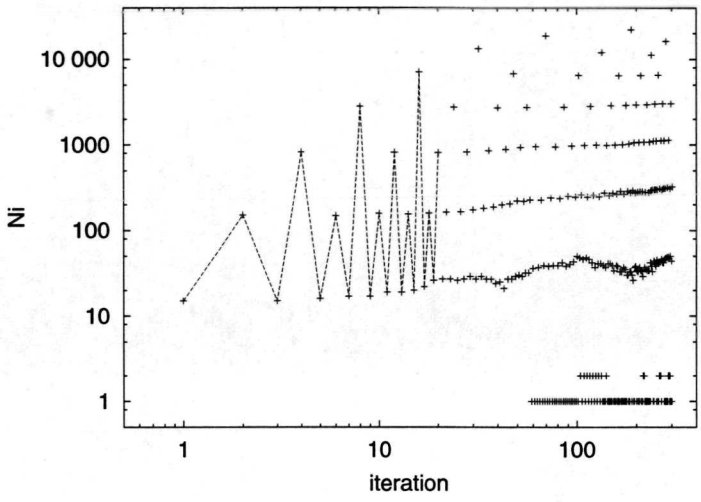


Figure 3.2: Time evolution of N_i for a simulation with $N = 32\,768$. The first 20 values are connected by a line, to make the oscillation clear. The unity (or twice the unity) value that can be seen on the bottom-right side is a hint of the presence of a binary star.

on P. The GRAPE performs the j-particle extrapolation simultaneously with the force computation. When the GRAPE computes the force exerted by a certain particle j_1, the j_1-data fetched from the GRAPE memory are passed through to the extrapolation pipeline. The pipeline outputs the extrapolated position of j_1, which is input to the force pipeline. The timing of the force computation task also includes the extrapolation. This is why there is no separate modelling for the j-particle extrapolation done on the GRAPE in table 3.1.

Force computation. The force computation task in the non-GRAPE case scales linearly with $N_i \cdot N/P$. Fig. 3.5 shows this dependence for a representative set of runs. Also in this case a data point refers to a single iteration, and is the average value of the timings for all the processors.

When forces are computed on the GRAPE, it does this task on N_p particles at the same time, using its array of pipelines. Then the same amount of time is spent to compute forces, for a number of i particles ranging from 1 to N_p. This time scales linearly with N/G (see fig. 3.8) where G is the number of GRAPEs available, since the force computation consists in an iteration on the N/G particles constituting the particle subset assigned to a GRAPE. The operation of receiving the result data from GRAPE is similar to the i-particle send, showing the same step behaviour.

The force computation task performed on the GRAPE shows a rather complicated structure. A number of communication procedures between the GRAPE board and the host must be performed, besides the actual force computation task. Figures 3.6, 3.7, and 3.8

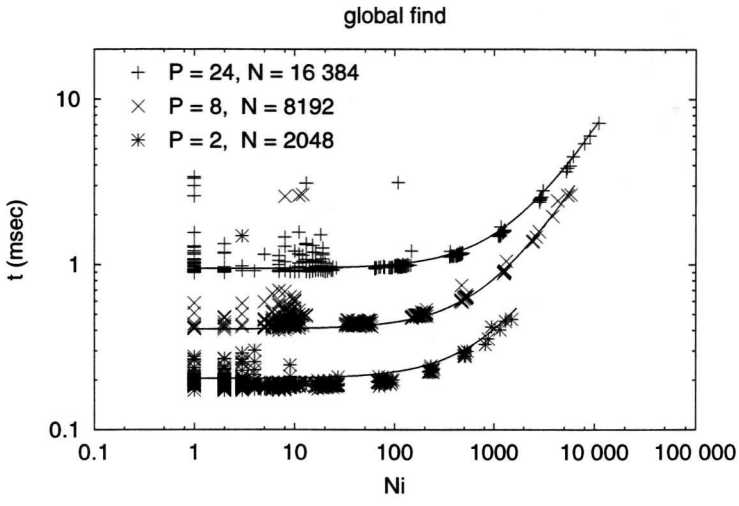


Figure 3.3: Timings averages of the global communication operation associated with the i-particles search. The formula we obtain by fitting the measurement values, which is used in the simulation model, and reproduced here as a continuous curve, is: $t = 0.14 + 0.038 \cdot P + (0.97 \cdot 10^{-4} + 0.15 \cdot 10^{-3} \cdot \log(P)) \cdot N_i$ (see parameterised expression in table 3.1).

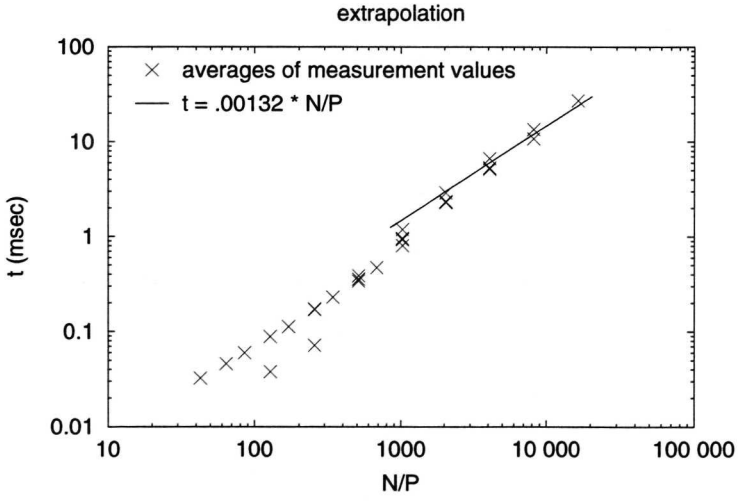


Figure 3.4: Timings of the extrapolation task for the non-GRAPE case, and fitting formula used into the model. A cache effect is clearly visible at $N/P = 1024$. We are interested in situations with high workload, thus we fit only the out-of-cache subset.

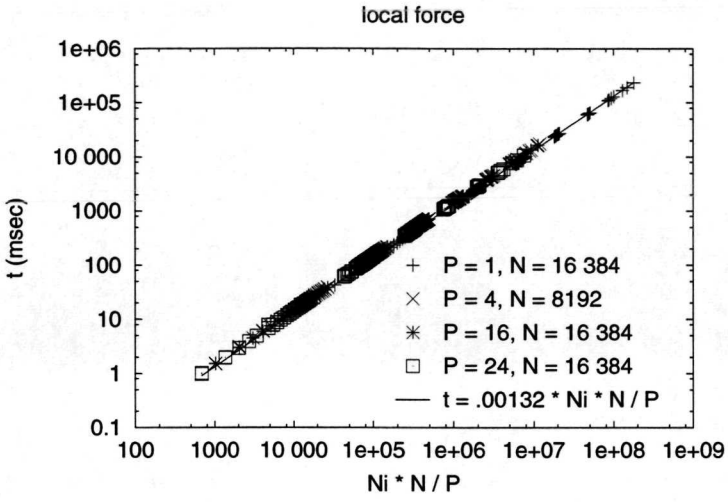


Figure 3.5: Timings and fitting formula for the local force computation in the non-GRAPE case, for a set of representative runs.

illustrate these tasks. Before GRAPE computes forces, the host sends it the j-particle positions that have changed in the last iterations. Since GRAPE stores the j-particle data in its internal memory, only the updated j-particle data need to be sent to it. Moreover, also the time-advanced position of the i-particles need to be sent in the same packet as the j-particles. In this way GRAPE avoids computing the self-interaction for the i-particles. The actual delivery of data is done in packets of up to 90 particles, and shows a linear dependence on the amount of data sent, plus a fixed latency time for each actual send operation. The j-particles send step is then a function of N_i and N_j. Fig. 3.6 shows the measured performance of this operation as a function of the data sent.

Another send operation is performed to send the i-particles to GRAPE. The actual data delivery is done in packets of N_p particles, where N_p is the number of active pipelines on the GRAPE board (up to 96). The time dependence of this operation with respect to N_i, the number of particles sent, is then a simple step function. Fig. 3.7 shows this dependence, for $N_p = 62$.

Besides the local force computation, a global communication is also needed for the parallel GRAPE code, as the total force computation requires a global sum. The execution of this operation does not differ between the GRAPE and non-GRAPE codes. Measurements of this operation from real runs show a communication time linear in $N_i \cdot \log P$, as shown in table 3.1.

***i*-particle update.** The final operation, i.e. orbit integration, updating and storing of the particles physical quantities, is a linear function of N_i, with no dependency on P, since every

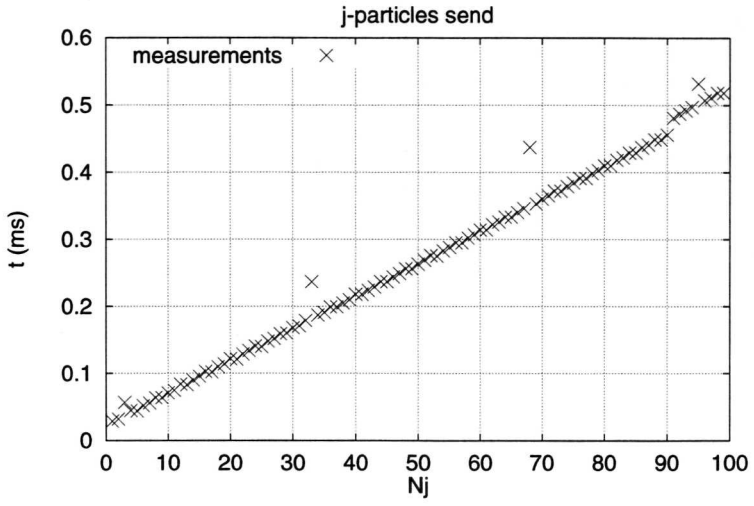


Figure 3.6: GRAPE related tasks. Timings of the j-particle send task as a function of the workload are shown. Measurements of the communication tasks show some occasional spike due to external processes, e.g. operating system function calls. A GRAPE with 62 pipelines is used for these measurements.

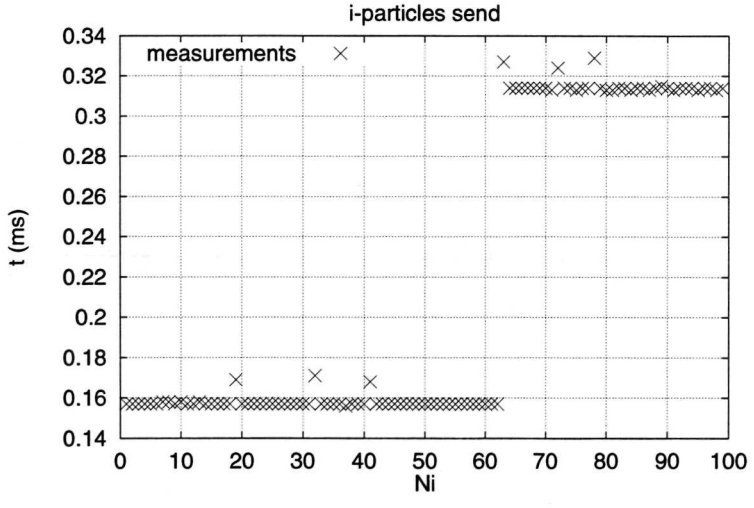


Figure 3.7: Same as fig. 3.6; here we show timings of the i-particle send task.

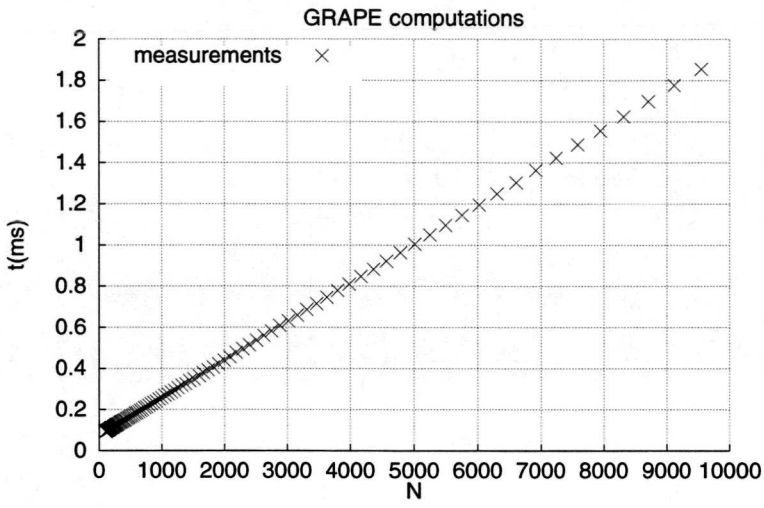


Figure 3.8: Same as fig. 3.6; here we show timings of the force computation task. The pipeline startup latency $\mu_S \simeq 75\mu$s is clearly visible.

processor performs this task for all the i-particles.

3.3.2 Treecode

The treecode, introduced in section 1.4.2, is widely used in Computational Astrophysics for the simulation of systems that do not require high computational accuracy. By trading lower accuracy with higher speed, the treecode is able to reduce the computational complexity of the N-body problem from the $\mathcal{O}(N^2)$ scaling of the direct code to $\mathcal{O}(N \log N)$.

The treecode computes force on a given particle i by grouping particles in larger and larger cells as their distance from i increases, force contributions from such cells being truncated multipole expansions. A simple pseudo-code sketching the basic tasks of the treecode is given in fig. 3.9.

The first task of a treecode iteration is to build a tree structure by hierarchically connecting each cell to the "child" cells that the cell encompasses (see section 1.4.2 for details). Then force is computed for each i-particle by traversing the tree, and looking for cells that satisfy an appropriate acceptability criterion (see section 1.4.2 and chapter 4 for details on acceptability criteria).

The original treecode algorithm has been modified in several ways to improve its performance. An optimisation of the tree traversal phase has been realised by grouping particles according to their spatial proximity (Barnes, 1990). Then a single traversal for each group is performed, whereas the original algorithm performs a tree traversal for each particle. This drastically reduces the number of tree traversals, and allows for concurrent force computa-

```
t = 0
while (t < t_end)
  build tree
  for each i-particle
    traverse tree to compute forces
  integrate orbits
  t = t + Δt
```

Figure 3.9: Pseudocode sketching the basic tasks in a treecode.

tion on vector machines. This optimisation is also suited for the use of the treecode with GRAPE, because each pipeline of the array contained in a GRAPE board can compute force on a different particle simultaneously. The drawback of this technique is an increase in memory use. In fact, for each particle group, an interaction list containing the information concerning all the cells interacting with the group must be written and stored in memory.

The use of interaction lists is also useful for parallelisation on distributed systems, as in the parallel treecode (Warren & Salmon, 1993, 1995). The possibility of decoupling tree traversal and force computation through interaction list compilation, allows for the implementation of latency hiding algorithms for the retrieval of cell information stored in a remote processor memory (Warren & Salmon, 1995; Salmon & Warren, 1997). We will refer to this version of the parallel treecode as HOT, the acronym of Hashed Oct-Tree, as the code was called by Salmon and Warren.

Another modification, introduced in the code GADGET (Springel, Yoshida, & White, 2001), consists in implementing the individual time step scheme, originally introduced in the direct N-body code, as described earlier in sections 1.4.1 and 2.3.1. In this manner, each particle is assigned an individual time step, and at each iteration only those particles having an update time below a certain time are selected for force evaluation (Springel *et al.*, 2001), so that force is computed only on a small fraction of the N particles. In this code, a different approach for remote interactions computation is also implemented: data of the selected particles are sent to the remote processors, interactions are computed remotely, and results are received back. A further modification consists in rebuilding the local tree less frequently than at every iteration. This version will be referred as GDT, which is a short for GADGET. In fig. 3.10 we give a pseudo-code representation of the generic algorithm that our model simulates.

Application model

Many different versions of the treecode have been proposed, implementing different tools and techniques. A recent report on this is given in (Springel *et al.*, 2001). Our performance model is designed to reproduce the behaviour of state-of-the-art parallel treecodes, running on distributed architectures, and able to make use of dedicated hardware. In this section, we describe each task of our application model, together with their modelling expressions.

```
t = 0
while (t < t_end)
  if code is GDT
    if it is time to rebuild tree
      build local tree
  else if code is HOT
    build local tree
    exchange data to build global tree
  if code is GDT
    for each selected particle
      traverse local tree to compute local forces
    send particles to remote nodes
    receive particles from remote nodes
    compute force on remote particles
    send forces to remote nodes
    receive forces from remote nodes
  else if code is HOT
    for each group
      build interaction list
      (communication needed for remote data retrieval)
    for each group
      for each particle in group
        compute forces
  integrate orbits
  t = t + Δt
```

Figure 3.10: Pseudocode sketching the generic parallel treecode tasks. HOT and GDT are the two versions of the treecode modelled in this work. Tasks involving communication are highlighted using a grey background.

Table 3.3 shows a synopsis of the modelling expressions, given as functions of the appropriate application parameters.

Tree building. The tree building task consists of two operations: particle insertion into the tree structure, and computation of multipole terms for each cell of the tree. Both operations, as long as local trees are concerned, do not require communication among processors. The particle insertion operation scales as $n \log(n)$, where n is the number of particles per processor. The multipole terms computation depends linearly on the number of cells per processor n_c, which is set equal to $0.1\,n$ divided by the number of particles per leaf cell. Each computing node processes all the particles independently to build a local tree. When

task	parameter definition	modelling formula
build local tree	n: number of particles per processor n_c: number of cells per processor n_{mp}: operations per cell to compute multipoles	$n \cdot \log(\pi_{bt} \cdot n) +$ $\pi_{mp} \cdot n_c \cdot n_{mp}$
data exchange for HOT global tree	P: number of processors	$\pi_{gt} \cdot P$
tree traversal	m: fraction of particles selected for force computation (= 1 if code is HOT) n_g: number of groups per processor j_{loc}: number of local force sources per group θ: opening angle (accuracy parameter for the force computation) K_j: scaling coefficient for force sources	$\pi_{tt} \cdot m \cdot n_g \cdot j_{loc}$ $j_{loc} = K_j \cdot \theta^{-3} \cdot \log(\theta^3 \cdot n)$
data exchange for HOT global lists	j_{rmt}: number of sources per group from remote processors	$\pi_{gl} \cdot j_{rmt} \cdot n_g$ $j_{rmt} = K_j \cdot \theta^{-3} \cdot \log P$
compute forces	j: total number of force sources per group N: total number of particles	$\pi_{cf} \cdot m \cdot n \cdot j$ $j = K_j \cdot \theta^{-3} \cdot \log(\theta^3 \cdot N)$
data exchange for GDT remote forces	*for parameter definition see above*	$\pi_{rf} \cdot m \cdot N \cdot \log P$
remote i-particle data sent to GRAPE hosts	*for parameter definition see above*	$\pi_{gr} \cdot (m \cdot n + j \cdot n_g)$
force data sent back by GRAPE hosts	*for parameter definition see above*	$\pi_{gs} \cdot m \cdot n$
integrate orbits	*for parameter definition see above*	$\pi_{or} \cdot m \cdot n$

Table 3.3: Synopsis of the modelling expressions for each task of the application model. All π terms are constant factors depending on the operations per particle performed, or the bytes per particle transmitted. Communication task expressions are highlighted using **a grey background**. Parameter values are given in table 3.4.

the tree building task is accomplished, the local tree contains all the particles which are located within the geometrical domain assigned to the processor. GDT uses only local trees for the force computation task, hence it does not execute other operations to complete the tree building task after the local tree building. Conversely, the HOT code exchanges information among the processors after the local tree building, so that each processor is able to build a global tree. In this way, no communication will be necessary during the force computation task. The data exchanged to build the global tree, the so-called local essential tree, are assumed to be equal for each processor, so that this operation is assumed to scale linearly with the number of processors P.

Tree traversal. This task is performed by HOT before the force computation, while GDT performs, for each i-particle, tree traversal and force computation as the same task. Namely, HOT first traverses the tree in order to build an interaction list for each group of nearby particles (an input parameter states how many particles make up a group), then uses the list to compute forces on each particle of the group (see also section 3.3.2 above). GDT instead, for each particle selected for force computation, traverses the tree and computes force simultaneously. The local tree traversal has to be done once for each particle group (GDT does not use groups, so in this case the number of groups is equal to the number of particles). It depends linearly on the number of local force sources. An expression for the total number of force sources j was found by Makino (Makino, 1991b), who gives $j \propto \theta^{-3} \log(\theta^3 N)$, where θ is the opening parameter (see equation 1.3 in section 1.4.2). The number of local force sources is then $j_{loc} \propto \theta^{-3} \log(\theta^3 N/P)$, and the number of remote force sources is $j_{rmt} = j - j_{loc} \propto \theta^{-3} \log P$. GDT performs this operation only for a fraction m of the particles per processor n. In the case of HOT, we simply set $m = 1$.

The HOT code completes this task with a communication operation, where information concerning remote force sources is received by each processor. This operation depends linearly on the number of groups, and on the number of remote force sources j_{rmt}.

Force computation. The cost of the force computation task on each processor is proportional to the number of i-particles per processor $m \cdot n$, times the number of force sources j. For the HOT code, this task does not require communication, since all information about remote force sources has been exchanged in the tree traversal task. In the GDT case, local i-particles are sent to the remote processors, then remote partial forces are retrieved to finally obtain the total force on each i-particle. This operations are global communication operations, and are assumed to depend on $\log P$, consistently with the modelling formulae of the global communication tasks of the direct code model in section 3.3.1 (see also table 3.1). This task also depends linearly on the total amount of data exchanged, i.e. on the total number of i-particles $m \cdot N$.

If GRAPEs are used, the force computation task is performed only by the GRAPE hosts. As a consequence, the communication operation executed in the HOT case during the tree build task in order to build the global tree, is executed only by the GRAPE hosts. As far as the actual force computation is concerned, the GRAPE hosts first compute force on their local particles, then receive remote i-particles and corresponding interaction lists from

	HOT on Delta	GDT on T3E	seq. tree on GRAPE-5
machine parameters			
μ_P $[\frac{\text{ns}}{\text{flops}}]$	12.5	1	1.67
μ_N $[\frac{\mu\text{s}}{\text{bytes}}]$	0.05	0.002	n/a
code dependent application parameters			
m	1	0.02	1
n_g	n	n	$n \cdot 0.0005$

code independent application parameters		
K_j	(force sources in tree traversal)	100
π_{bt}	(build local tree)	0.15
π_{mp}	(compute multipoles in local tree)	10
π_{gt}	(HOT global tree)	3200
π_{tt}	(tree traversal)	0.4
π_{gl}	(HOT global lists)	32
π_{cf}	(compute forces)	0.15
π_{rf}	(GDT remote forces)	48
π_{gr}	(remote i-part. sent to GRAPE hosts)	32
π_{gs}	(forces sent back by GRAPE hosts)	24
π_{or}	(integrate orbits)	0.3

Table 3.4: Values of the performance parameters appearing in the modelling formulae in table 3.3

the "un-graped" processors, compute force on remote i-particles, and finally send back forces to the remote processors.

The actual force computation operation on the GRAPE is modelled using the same expression as in table 3.1, where the number of i-particles is in the present case put equal to the number of particles per group n/n_g, and the number of j-particles (N/GRAPEs in the formula in table 3.1) now is equal to the total number of force sources per group j.

The cost of the communication operation to send remote particle data to the GRAPE

host is proportional to the amount of bytes which are sent, which is proportional to the number of i-particles sent, plus the length of the correspondent interaction lists. Hence it is modelled as a linear function of the number of i-particles $m \cdot n$ plus the number of force sources for all groups $j \cdot n_g$. The cost of sending back the forces is proportional to the number of i-particles $m \cdot n$.

Orbit integration. This task consists in the updating of the i-particles positions, and does not require communication. It is modelled as a linear function of the i-particles per processor $m \cdot n$.

Computer architecture

The parallel system simulated in our machine model is a generic distributed multicomputer, where given nodes can be connected to one or more SPDs. When SPDs are present, the appropriate task is executed on them. The application model needs no modification in this case. According to an input parameter which tells whether SPDs are present, the mapping interface chooses the routine that maps the task to the SPD, or to the general purpose processor. Since we are interested in SPDs dedicated to the gravity force computation, the machine model of the SPD reproduces the GRAPE activity, and its communication with the host. The modelling of the fairly complicated data exchange machinery between GRAPE and its host is discussed in section 3.3.1.

The hardware characteristics of the simulated multicomputer are parameterised by two constants, μ_P and μ_N, where μ_P accounts for the processor speed, in nanoseconds per floating point operations, and μ_N accounts for the network speed, its value being the transfer rate in μs/B. In the execution model, each computation-related function will be multiplied by μ_P, and each communication-related function (those highlighted with a gray background in table 3.3) will be multiplied by μ_N. Parameter values for the simulations presented in section 3.4.3 below are given in table 3.4.

3.4 Simulations

In section 3.3 the modelling of the various application tasks of the N-body codes that we study have been described. In this section, we show how our models reproduce the real system, and simulate possible modifications. The models consist in a sequence of tasks, as described in section 3.3, each one specifying, by means of appropriate `delay` operations, how much wall-clock time is spent to perform them. The access to GRAPE is controlled by a semaphore. Model results are compared with data obtained from the performance analysis study of our system reported in section 2.4.

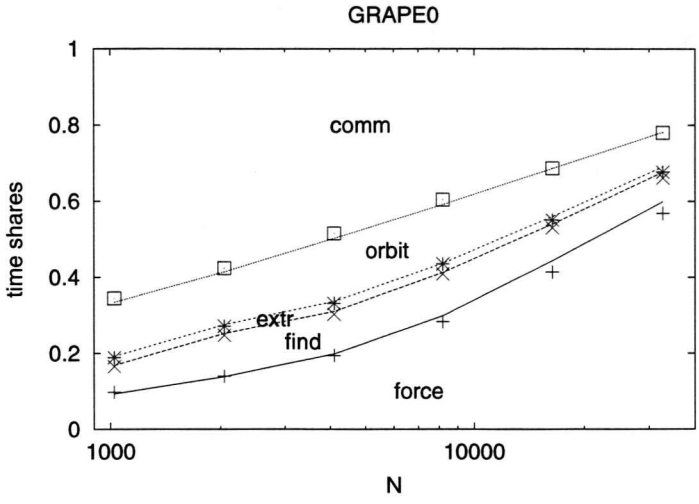


Figure 3.11: Model validation. Points indicate the data obtained from actual timings, lines are simulation results. GRAPE0 refers to the system where the GRAPE has 62 pipelines.

3.4.1 Serial direct code

Validation

The reliability of our model has been checked by making a comparison between the simulation results, and the actual measurements of a set of runs of NBODY1 on a system consisting of a GRAPE connected to its host, which is a node of the DAS, as described in section 2.2.1. Two separate series of runs have been performed, on each GRAPE at our disposal. The GRAPE board with 62 pipelines is labelled GRAPE0; the other board, with 94 pipelines, GRAPE1. Each run consists of 300 iterations, with N ranging from 1024 to 32 768. The initial condition is a Plummer model (Plummer, 1915), i.e. a star distribution with density decreasing as the fifth power of the distance from the cluster centre (see, e.g., Spitzer, 1987, p. 13). In section 3.3.1 we compared measurements with simulation for single tasks. Now, in fig. 3.11 and 3.12, we present a global comparison, where we show how each task scales with N. In these figures we plot the time share spent by the application in accomplishing each task. These figures show that our model produces results in good agreement with the real measurements. These measurements are presented in section 2.4.3.

It can easily be seen how the system performance is strongly penalised by communication overhead, unless the workload is high (i.e. $N \geq 16384$). Even in such cases, GRAPE is not fully exploited yet, due to the large time-share taken by host computations. It is clear from this that a faster host and an improved communication interface are needed to achieve an optimal GRAPE utilisation. A comparison between fig. 3.11 and 3.12 shows that the time share for the force computation is smaller for GRAPE1. This is due to the higher number of

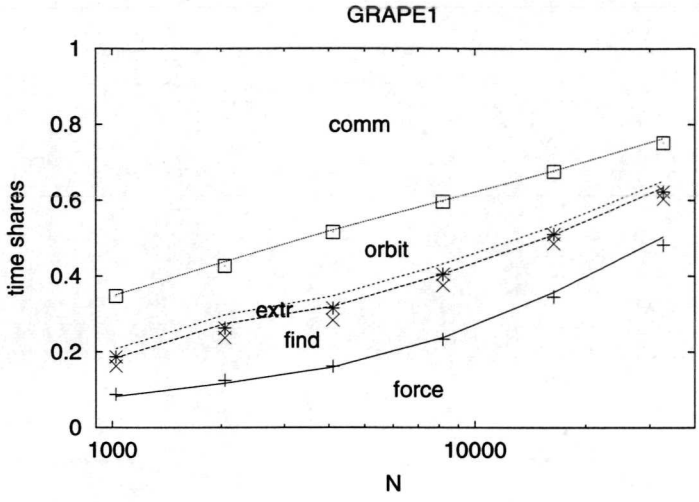


Figure 3.12: Model validation. Same as fig. 3.11. GRAPE1 refers to the GRAPE where the number of pipelines is 94.

pipelines in this GRAPE board, which makes the force computation faster. See also fig. 2.14 and 2.15 on page 45, where the same performance data for the real system are presented.

To test the versatility of our model, we also validated it with respect to a system configuration without GRAPE. In this case the mapping interface of the model, instead of selecting the procedure where the use of GRAPE is modelled, maps the force computation task on a different procedure, where the force computation task is modelled as a linear function of $N_i \cdot N$. The user selects whether the force computation will be modelled as a host related task, or as a task involving the use of GRAPE, simply by changing an input parameter. Fig. 3.13 shows the task time shares, while fig. 3.14 shows the total execution time for the application that does not make use of GRAPE. Measurements data and simulation results are compared.

Fig. 3.13 shows that the force computation task dominates the system activity. From a comparison with fig. 3.11, where the force computation share is remarkably smaller, it becomes clear how effective is GRAPE in optimising this task. In fig. 3.14 the execution time for the code that uses GRAPE1 is also plotted to show how large is the speedup achieved thanks to the GRAPE.

Predictions

The above discussion highlights the need for a faster host and communication interface. Our model has been used to forecast the benefit obtainable by operating such improvements.

We modified our model to simulate a host twice as fast as our present host, and with

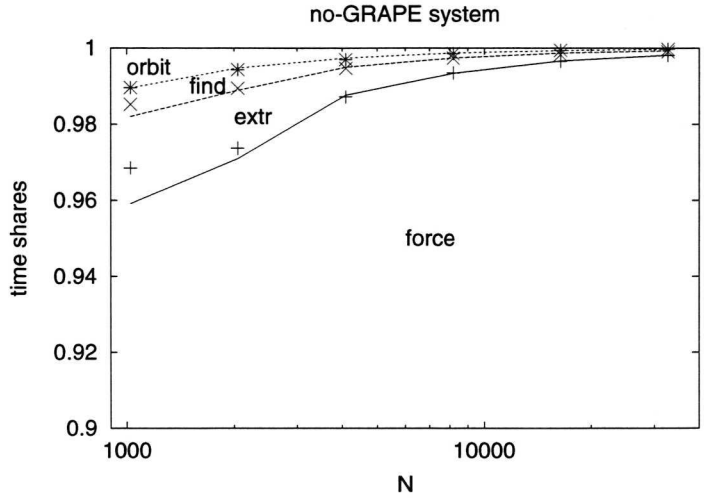


Figure 3.13: Model validation for a system not using GRAPE. Since the force computation time share for this code is by far larger than the other task shares, the y-axis scale has been changed to make data more readable.

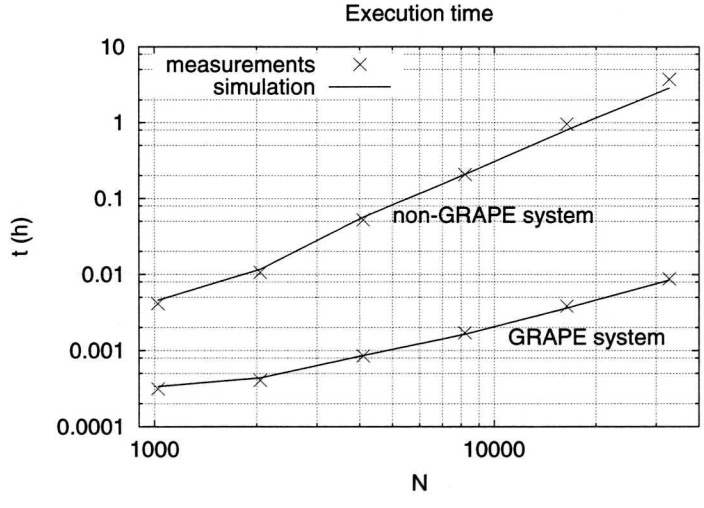


Figure 3.14: Model validation for a system not using GRAPE. Comparison of the total execution times of the GRAPE system and the non-GRAPE system.

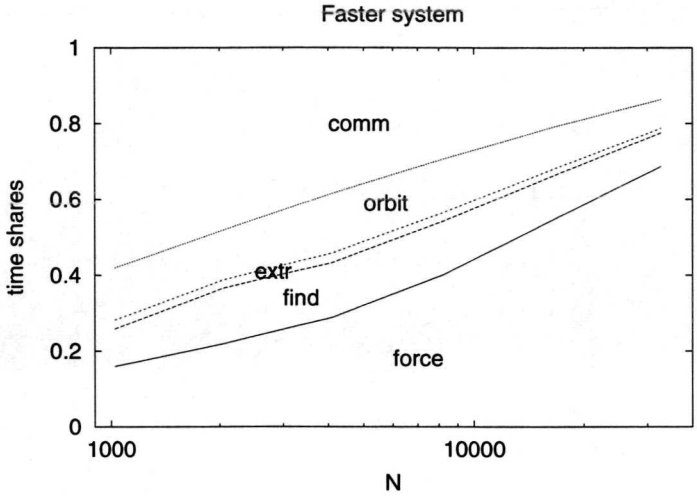


Figure 3.15: Simulated time shares for a system with a faster host and communication interface, but with the same GRAPE board as GRAPE1.

communication performance two times faster as well.[3] In this manner we try to reproduce the system measured in Kawai *et al.* (1997), consisting of a DEC workstation 500MHz, using a GRAPE board with 94 pipelines of the GRAPE-4 cluster. The time-shares, and a comparison of the estimated performance gain of the simulated system with respect to the system described in the previous section are given respectively in fig. 3.15 and 3.16.

These figures show that the GRAPE board is used more efficiently now, and the overall system performance benefits of this. Nevertheless, it appears that when the workload is high, this performance gain decreases. This is predictable, since in this case the relevance of the host and the communication interface is not as large as with a lighter workload. The estimate in Kawai *et al.* (1997) is in agreement with ours, it only attributes a larger time share to the host computation tasks at small values of N. This discrepancy can be explained considering that they model $N_i = 1.6 \cdot N^{1/2}$, with no oscillation. In this way, when $N < 3600$, N_i is always smaller than 96, i.e. the maximal number of pipelines in a GRAPE board. Now, since the force computation and communication step is always done in a single iteration, the relevance of GRAPE and the communication interface is reduced. Conversely, if the value of N_i can oscillate and assume values greater than the number of pipelines in a GRAPE board, as it happens in our model, two or more iterations are necessary, increasing the relative load of the communication and GRAPE computation tasks.

Another use of our model is the estimation of the performance gain that can be reached by improving the communication operations. As mentioned above, the i-particle send and

[3]More precisely, the performance is set to be two times faster for the send operations, and five times faster for the receive operations, in order to reproduce in any aspect the performance figures given in Kawai *et al.* (1997).

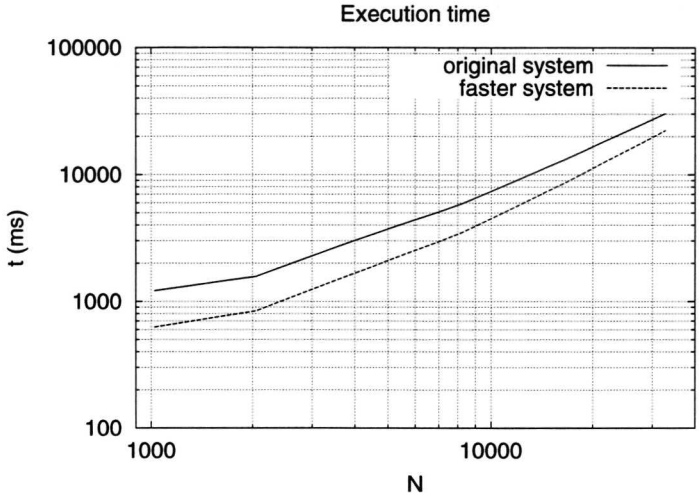


Figure 3.16: Simulated performance gain for a system with a faster host and communication interface, but with the same GRAPE board as GRAPE1.

the result retrieval are performed in a buffered fashion, where data are sent for a number of particles always equal to N_p, also when N_i is less than N_p. For the result retrieval, the situation is even worse; in that case, the operation is performed always for 96 particles, i.e. the maximal number of GRAPE pipelines.

One may then wonder how much the system performance could benefit, if this communication protocol is improved. We simulated a situation where the send and receive operations are accomplished by transmitting a variable size packet of up to N_p particles per time, in a fashion similar to the j-particle send. However, an extra amount is added to the packets to represent the defective pipelines in the GRAPE board. Every GRAPE board includes 96 pipelines, but some of those can be defective. We assume the worst-case situation, in which the $N_d = 96 - N_p$ defective pipelines are the first ones to be accessed. Assuming that data regarding $N_i = \tilde{N}$ particles have to be transmitted, the operation that we model consists in writing to (or reading from) the first $\tilde{N}$ non-defective pipelines. Then our packet contains $N_d + \tilde{N}$ items.

The results of this simulation are shown in fig. 3.17 and 3.18. The simulated board has 94 pipelines working, and 2 defective. It can be seen how little influence the discussed modification has on the system performance. In this case, the model forecast discourages the enterprise of implementing such a modification in the real system. This example shows how performance modelling can be useful in evaluating whether a new project is promising, or, as in this case, it is likely to be unsuccessful.

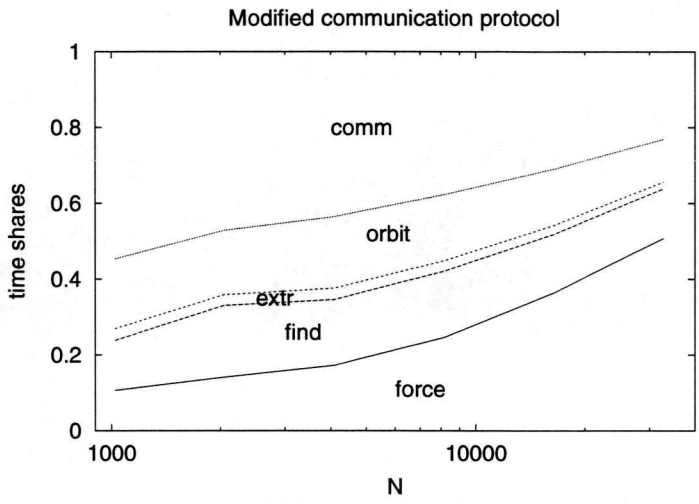


Figure 3.17: Simulated time shares for a system with modified communication operations. The GRAPE board is the same as GRAPE1.

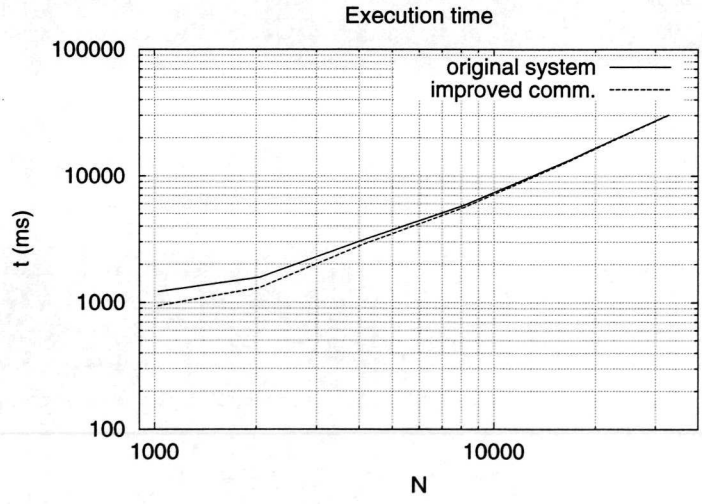


Figure 3.18: Simulated performance gain for a system with modified communication operations. The GRAPE board is the same as GRAPE1.

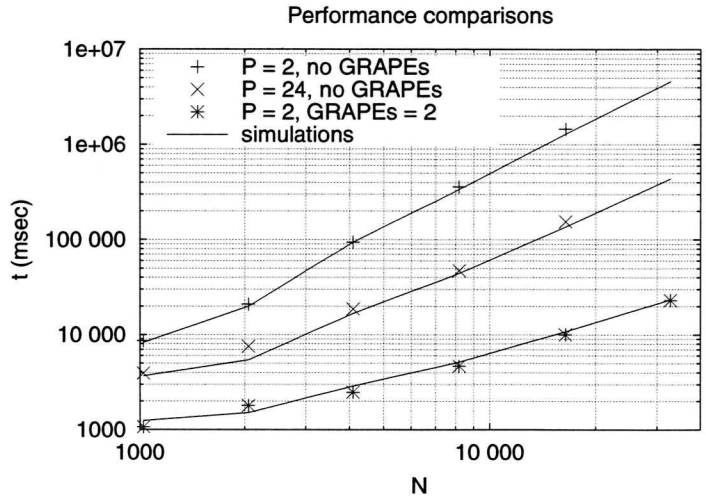


Figure 3.19: Model validation: comparisons of the overall execution times (real and simulated).

3.4.2 Parallel direct code simulations

Validation

In order to check the validity and the versatility of our model, we compare our simulation results with the performance analysis data presented in section 2.4. In the first case, we consider a situation with 2 processors, each one connected to a GRAPE; in the second case, we scale up to 24 processors. This corresponds to the architecture at our disposal, described in section 2.2.1. Each run, either real or simulated, consists of 300 iterations, with N ranging from 1024 to 16 384 for the non-GRAPE case, and to 32 768 for the GRAPE case. As initial condition in the real runs we used, as in the serial case, a Plummer model (Plummer, 1915).

Timing results for the overall execution time are presented in fig. 3.19. We show results for the non-GRAPE case, with two different values of P, and for the GRAPE case with the two GRAPEs each attached to its own host. The ability of our model to fit the measurement values can be readily inferred from this figure. The worst case error amounts to $\simeq 40\%$ for the non-GRAPE case with $P = 24$, $N = 2048$ (a case that lies well outside the parameter range that we are really interested in), whereas the average error is $\simeq 10\%$.

Besides the overall timing, we also show the fraction of time spent to perform each task. Fig. 3.20 and 3.21 are for the non-GRAPE case, and fig. 3.22 is for the GRAPE case. It can be seen in figures 3.20 and 3.21 how the different application tasks scale with N, i.e. with the total workload. For the case with $P = 24$, shown in fig. 3.21, a large communication overhead at low values of N is visible.

The large share of execution time taken by the force computation task is a clear evidence

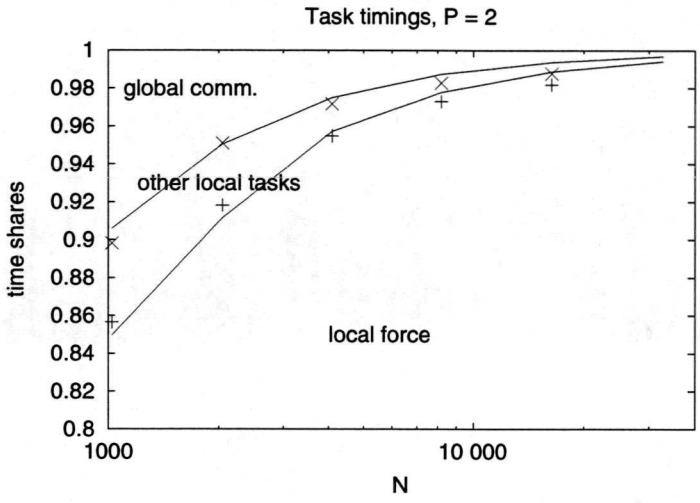


Figure 3.20: Model validation for the parallel non-GRAPE system. For each task, the ratio t_{task}/t_{tot} is shown, as a function of N. Here a system with two processors is shown. Points are measurement data from test runs, lines are simulation results. The scale on the y-axis does not start from 0 (cf. fig. 3.13 for the serial case).

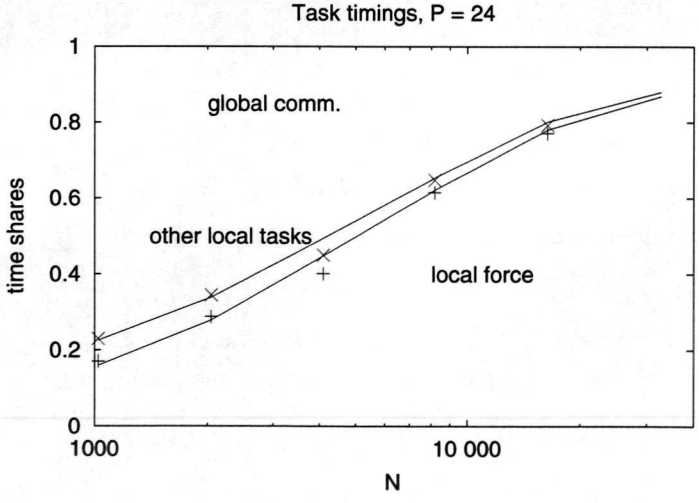


Figure 3.21: Model validation for the non-GRAPE system. Same as fig. 3.20. In this case a system with 24 processors is shown.

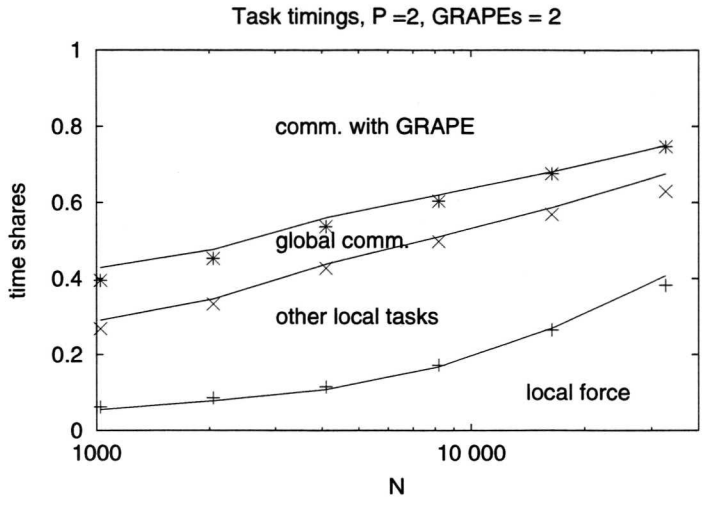


Figure 3.22: Model validation: time shares of the GRAPE system. See also fig. 2.16 on page 46 for the performance measurements of the real system.

of the need for a tool to accomplish this task faster. Fig. 3.22 shows how effectively GRAPE solves this problem. The relative importance of the force computation task has been drastically reduced by using GRAPE, even though at the cost of a large communication overhead with the SPD.

System comparison. Fig. 3.19 shows that the GRAPE system is two orders of magnitude faster than the non-GRAPE system having the same number of processors, while with respect to the most powerful non-GRAPE configuration available, the one with 24 processors, the performance gain is still about one order of magnitude. Our model can reproduce the behaviour of both systems quite satisfactorily. In the following, we use our model to predict how this behaviour changes as a consequence of system modifications.

Predictions

In this section we present some examples of the use of our model in order to predict the performance of systems where either hardware or software modifications have been carried out. Performance estimation and algorithmic design are the main fields of application that our simulation model is designed to serve.

Clustered GRAPEs vs distributed GRAPEs. A fundamental question that we want to answer is whether it is more efficient to connect several GRAPEs to the same host node, or to have a network with several nodes, each one being the host of a GRAPE. The first

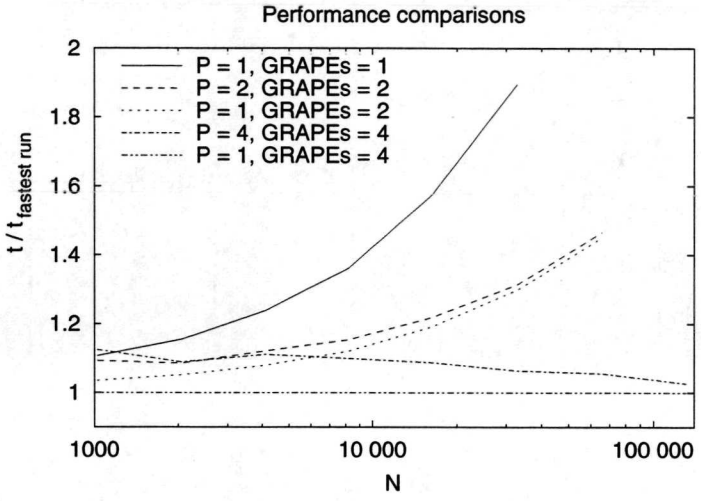


Figure 3.23: Simulation of different host-GRAPEs configurations. The execution time for each simulated configurations is divided by the timings of the fastest run, which is in all cases the one with $P = 1$ and GRAPEs = 4. The actual time range can be inferred from fig. 3.19.

configuration, which also reflects the original system architecture for this device,[4] does not exploit a multiprocessor host in order to perform the particle search in parallel but, on the other hand, does not incur any overhead cost for the two global communications required by the parallel code. The configurations that we simulated contain 1, 2 or 4 GRAPEs, either connected to one single host, or distributed one GRAPE per host. The result of our simulation is shown in fig. 3.23. The total number of particles is increased up to 32 768 particles per GRAPE board. It can be seen that the performances are almost equal for all cases having the same number of GRAPEs. The gain for the multiprocessor configuration to distribute the local search, is roughly of the same amount as the loss due to communication overhead. It can be inferred that both hardware configurations analysed here, i.e. localised SPDs versus distributed SPDs, perform about equally well, the single host configuration performing slightly better.

A more realistic N-body code. The code modelled in the preceding sections is a basic N-body code. State-of-the-art astrophysical codes will contain additional functionality, e.g. to model close encounters to binary stars and the evolution of stars. In state-of-the-art direct N-body codes, a binary star is treated as a single entity. When a third star approaches the binary, the motion of the two components of the binary, plus the encountering star, is resolved analytically, by means of a rather complex procedure (Funato *et al.*, 1996). This additional functionality must be provided by the host, leading to an additional workload. In this case,

[4] I.e. the GRAPE-4 system at the University of Tokyo, consisting of 36 GRAPE boards connected, in a hierarchical fashion, to a single workstation (see section 1.3 and fig. 1.5).

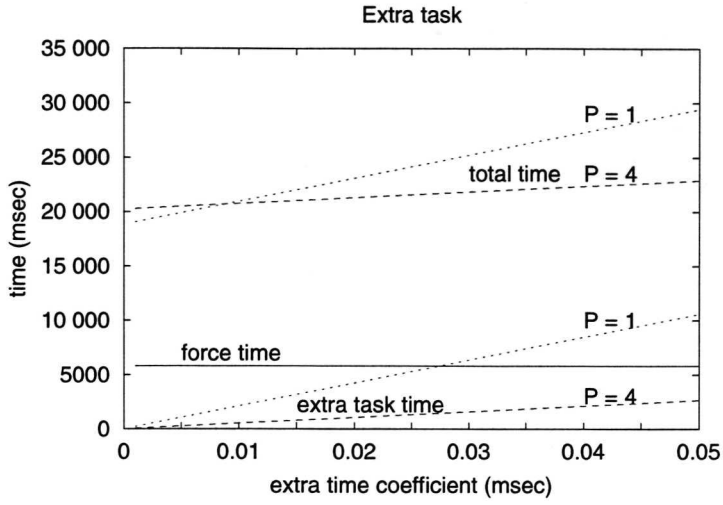


Figure 3.24: System performance comparison when an extra task, modelled as $t = \eta \cdot N_i/P$, is added, where η is an experimental factor of proportionality. In this case is $N = 32\,768$.

it is interesting to see when the multiple host configuration begins to outperform the single host configuration. This extra task is assumed to be linearly dependent on N_i through a coefficient η, and perfectly parallelisable. This last condition is likely to hold in reality as long as P remains reasonably small.

We simulated the case with $N = 32\,768$, and compared the "clustered" case where one processor hosts four GRAPEs to the "distributed" case where four processors host one GRAPE each. From the previous section, the clustered configuration is faster when the algorithm without the extra task is used. The results are shown in fig. 3.24. We can see that the configuration with $P = 4$ begins to perform better at $\eta \simeq 0.01$, when the time spent in the extra task is still negligible compared to that spent in the force task (at least for the distributed case). In order to compare η with the parameters reported in table 3.2, we have to divide it by the processor time cycle, which is $0.5 \cdot 10^{-5}$ ms. This results in $\tau_\eta \simeq 2000$, i.e. about ten times the value of the extrapolation or local force task parameter in table 3.2. The amount of computations for the close encounters procedure mentioned above, is of this order of magnitude. This example shows that the multiple host configuration is more appealing because of its better performance potential.

Distributed GRAPEs load balancing. A problem in the distributed GRAPEs configuration is the load balancing of the force computation task. When the number of pipelines per board is not the same for all the boards, the boards with the highest number of pipelines are faster in performing the force computation, because they handle more i-particles per unit time. In our case, the idle time is up to 10% of the total averaged time of the two

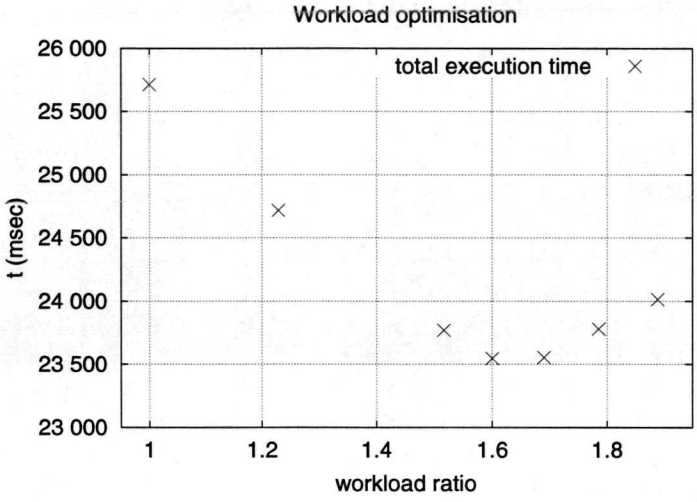


Figure 3.25: Load balance optimisation. Workload imbalance for a configuration with two GRAPEs on different hosts.

GRAPE boards, which means that the fastest board is idle for about 20% of the time. For the sake of readability, the idle time was not shown explicitly in fig. 3.22, but was included in the *communication with GRAPE* task. We used our performance model to find the optimal partitioning of particles between the two GRAPEs in the distributed configuration, where each GRAPE is connected to its own host.

Fig. 3.25 shows the result of this study. The configuration analysed here includes a GRAPE with 62 pipelines, called G0, and a GRAPE with 94 pipelines, G1. The total number of particles is 32 768. The figure shows how the total execution time changes, as the number of j-particles on G1 is increased in order to better exploit its higher computational potential. The optimal distribution, i.e. the one with the minimal idle time, is not reached when the ratio r between the j-particles on the two GRAPEs is equal to $94/62 \simeq 1.52$, i.e. the ratio between the pipelines, but at a slightly higher value of r. Unbalance between the two GRAPEs is due to both computation and communication. When $r = 94/62$, only computation is balanced. The host-GRAPE communication time is faster for G1, because the time cost of the *receive result* operation is inversely proportional to the number of pipelines (see table 3.1). Then, at $r = 94/62$, G1 is still faster than G0, because of its better communication performance. A further slight overload of G1, such that $r \simeq 1.6$, balances the overall execution time of the two devices.

This somehow unexpected result, produced by the complex dependence of the computation and communication tasks on N_i and N_j, illustrates how a detailed simulation is useful to analyse the behaviour of a hybrid architecture.

	HOT on Delta		**GDT on T3E**		**seq. tree on GRAPE-5**	
	real	simulated	real	simulated	real	simulated
build tree	112	144	0.175	0.145	3.00	4.07
traverse tree	528	955	0.394	0.468	2.00	2.03
compute force	864	1433			6.90	9.57
host-GRAPE commun.	-	-	-	-	3.11	4.99

Table 3.5: Comparison of the timings breakdown between the real measurements and the simulation risults. Timings are in seconds for a single code iteration. The **HOT on Delta** case refers to a 8.8 million particle run on 32 i860 40 MHz processors; **GDT on T3E** refers to a 0.5 million particle run on 16 Alpha 300 MHz processors, and **seq. tree on GRAPE-5** to a one million particle run on a 500 MHz Alpha processor connected to a GRAPE-5 board.

3.4.3 Parallel treecode

Model validation

We present here the result of running our simulation model of the treecode, described in section 3.3.2. We use this model with parameter values such that performance measurements reported in the literature are reproduced. We show for each case the scaling with the total particle number N of each task of the code, compared with the corresponding real system timings, as reported by the measurements authors. Finally we present a plot comparing the total compute time for a code iteration of each configuration. We had to deal with the fact that in most cases data were available only for one measurement run. Therefore a conclusion on the ability of our performance model to reproduce the scaling behaviour of the simulated system can only be incomplete from these data. The partial information that we obtain from this work is nevertheless fundamental to provide us the main guidelines for the realisation of a parallel environment for the simulation of N-body systems, as reported in the section on model forecasts. Once this environment will be realised, we will be able to validate our performance model thoroughly, having a system of our own to carry out performance measurements. In table 3.5 we show the comparison between the timings breakdown of the real measurements and our simulation results. In fig. 3.27 the compare the global timings of the various cases.

HOT on Touchstone Delta. The Touchstone Delta was a one-of-a-kind machine installed at Caltech in the early nineties. It consisted of 512 i860 computing nodes running at 40 MHz, and connected by a 20 MB/s network. The performance measurements reported in (Warren & Salmon, 1993) are based on a run using the whole 512 nodes system, and consist in a timing breakdown of a code iteration taken during the early stage of evolution of a cosmological simulation, when the particle distribution is close to uniform. The total number of particles is $N = 8.8 \cdot 10^6$. Implementation limitations prevented our performance model to simulate 512 concurrent processes, so that we limited our simulation to 32 processes, and scaled

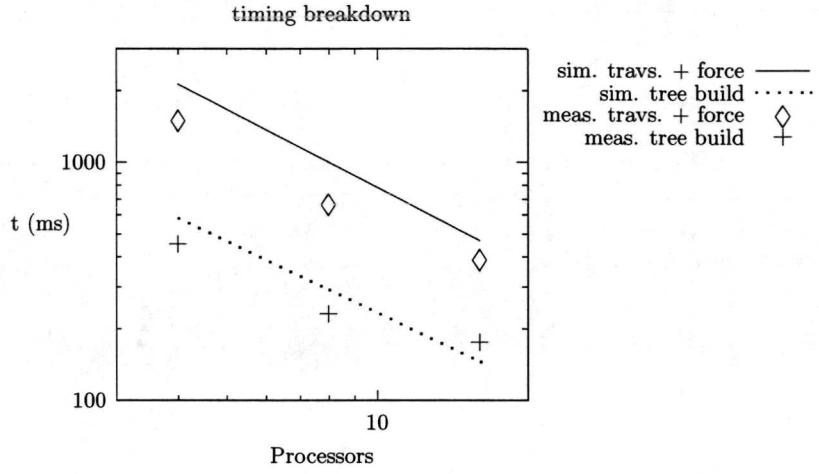


Figure 3.26: Timings of the GDT tasks. The real system timings are also reported. The hardware architecture is a Cray T3E. Performance scaling with the number of processors, with $N = 500\,000$.

down 16-fold the measured compute time reported in (Warren & Salmon, 1993). Since the communication overhead for that run was just $\simeq 6\%$, we assumed a linear scalability of the code. The timings breakdown of our simulation is presented in table 3.5. The real system measurements are reported for comparison.

The table shows that the force computation task and the tree traversal are the most expensive tasks. The relative computational weight of each task is qualitatively well reproduced by our model. Quantitatively, a large discrepancy between our model and the real system timings originates from an over-estimation of the tree traversal and the force computation tasks, which also results in over-estimating the total time, as shown in fig. 3.27. We must conclude that in this case our model is not sufficiently well matched.

GDT on T3E. This case reproduces the configuration described in (Springel *et al.*, 2001), where the GADGET code is run on the T3E hosted at the supercomputing centre in Garching, Germany. Each computing node has a frequency of 300 MHz, and the communication network has a throughput of 500 MB/s. Three cases are reported in (Springel *et al.*, 2001), each running the same cosmological simulation, where a system of 500 000 particles is evolved for 3350 time steps. The difference among the three cases is in the number of processors used. Since in this case measurements from three different hardware configuration are reported, we could compare our model results with a larger set of timing values. As reported in (Springel *et al.*, 2001), we assumed that only 5% of the particles are selected on average at each time step for force computation. Similarly, we assumed that the local tree is rebuilt each 10 time steps. Timing breakdowns are shown in fig. 3.26 and in table 3.5.

Fig. 3.26 shows the performance gain as the number of processors increases. The trend

in the measurements suggests a saturation in the attained performance, arguably due to an increasing load imbalance. This trend is not visible in our model results, because load imbalance is not modelled. Data in table 3.5 show that, with respect to the HOT case, now the tree build task is more expensive, despite the fact that it is performed only every ten iterations. This is to be expected, since the tree build task is performed for all particles, while the tree traversal and force computation tasks are performed only for a small fraction (5%) of the selected particles. Also in this case our model results match the real system timings. Springel *et al.* (2001) did not provide separate values for the tree traversal and the force computation tasks, so that only the aggregate value can be reported on the plot.

Sequential treecode on GRAPE-5. Here we simulate the configuration described in (Kawai *et al.*, 2000). In that case, a modified treecode is used to simulate a system containing one million particles, and groups of $\simeq 2000$ particles share the same interaction list. This code is run on a Compaq workstation with a 500 MHz Alpha 21264 processor, connected to a GRAPE-5 board containing 96 virtual pipelines,[5] each one able to compute a force interaction in 75 ns. Estimating a force interaction as 30 flops, the aggregate performance of a GRAPE-5 board is 38.4 Gflop/s. Table 3.5 shows the results of our simulation model, compared with the real system timings, as reported in Kawai *et al.* (2000).

In this case, the force computation task is performed by the GRAPE. An important fraction of the total timing is taken by the communication between the host and the GRAPE. The decrease of importance of the tree traversal task, due to the particle grouping technique, is clearly observable.

Cases comparison. We compare here the three cases presented above. We show in fig. 3.27 a plot of the time taken by a code iteration versus N, as obtained from our simulation model, compared with the real system measurements. The value for the HOT code on the Touchstone Delta is 16 times greater than the value reported in Warren & Salmon (1993), in order to scale their 512 processor run to our 32 processor simulation. Conversely, scaling our simulation data for 32 processors to 512 processors, would have resulted in simulation values overlapping the values for the GRAPE case.

The simulation values match within approximately a factor 2 the real system measurements. In the next section we present results of a performance simulation, where our model is used to forecast the behaviour of other configurations.

Model forecasts

In this section we explore the possibility of using a hybrid architecture consisting of a distributed general purpose system, where single nodes host zero or more GRAPE boards. We span the two-dimensional parameter space defined by the two quantities P, the number

[5] A GRAPE-5 board contains in fact 16 physical pipelines, each one running at 80 MHz, which is 6 times the speed of the board bus. The board "sees" 16*6 = 96 logical pipelines, running at 80/6 MHz. Appropriate hardwiring manages the data exchange between the pipelines and the board.

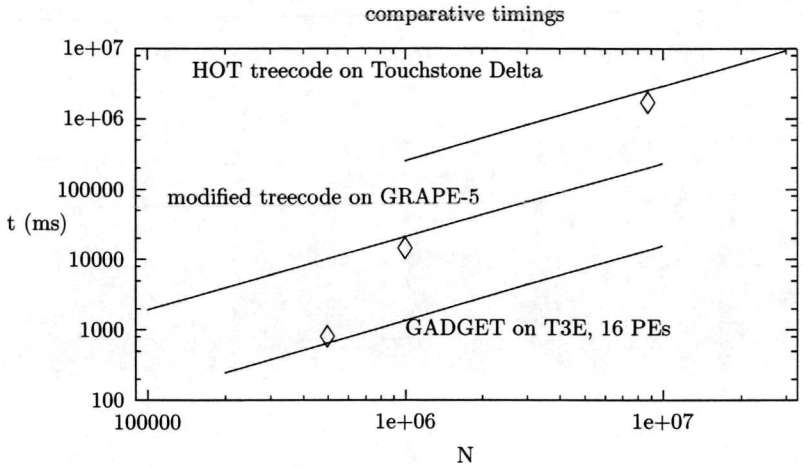


Figure 3.27: Predictions of a code iteration for the three simulated configurations. Note that GADGET moves only 5% of all particles on average in a time step, and the HOT case refers to a system with 32 processors, instead of the 512 of the original system.

of nodes, and G, the number of GRAPEs. We assign to those quantities values as follows: $P \in \{1, 2, 4, 8, 12, 16, 20, 24\}$, $G \in \{0, 1, 2, 4, 8, 12, 16\}$. We simulate the same software configuration as described in the previous section with respect to the case related to the sequential treecode on GRAPE-5. The SPD we simulate in this case is the GRAPE-4 (Makino *et al.*, 1997), whose performance per board is 30 Gflop/s, comparable to GRAPE-5's. It provides a higher accuracy with respect to GRAPE-5, and is used in fields as Globular Cluster dynamics on Planetesimal evolution (Hut & Makino, 1999), where high computing precision is required. The general purpose nodes are assumed to perform a floating point operation in 2 ns, and the communication network is assumed to have a 100 MB/s throughput.

When a node of the distributed system is a GRAPE host, forces on its local particles can be computed on the GRAPE that it hosts. Forces on particles residing on nodes that do not host GRAPEs can be computed on remote GRAPEs, provided that both particle positions and particle interaction lists be sent to the appropriate GRAPE host. This implies a very large communication traffic. With our simulation we try to evaluate the effect of this communication overhead.

Fig. 3.28 shows our results. It is clear that, as long as all nodes are connected to one or more GRAPEs, a significant performance gain is obtained. For comparison, we also provide timings of a system without GRAPEs. When not all nodes are GRAPE hosts, the very large communication overhead due to sending particle and interaction list data is disruptive for performance. This result suggests that the communication task needs a very careful analysis, in order to design an efficient parallel treecode for hybrid architectures. Here we assumed that an "un-graped" node sends all its data to a single "graped" node. We

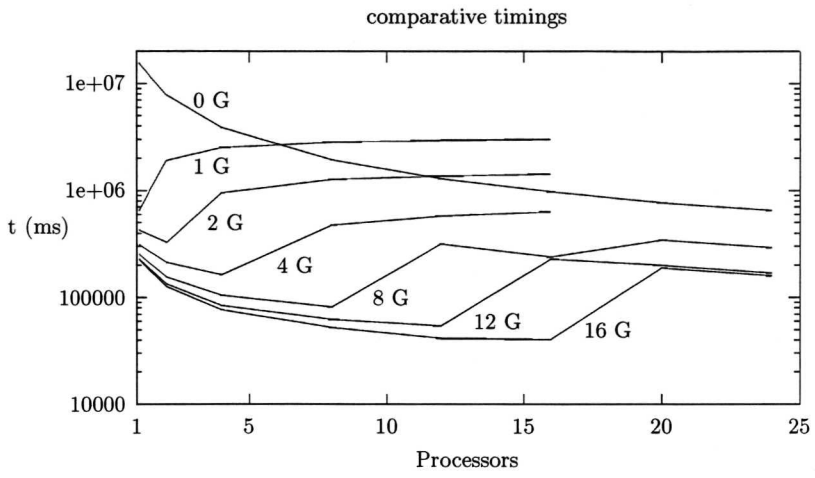


Figure 3.28: Timings for a system with P processors and G GRAPEs. Comparison with a system without GRAPEs (marked as 0 G) is also provided. The software configuration is the same as in the treecode on GRAPE-5 described above. The total number of particles is $N = 1\,000\,000$.

discuss this point further in the next section. The plot in fig. 3.28 also features an oscillatory behaviour, particularly evident in the case with 8 GRAPEs. The local minima (i.e. better performances) correspond to configurations where P is an exact multiple of G. In this case the computational load on the GRAPEs is perfectly balanced, whereas in the other cases some GRAPE bears a higher computational load from remote data.

3.4.4 Direct code vs treecode

A main goal of our research is to develop a distributed hybrid architecture optimised for the treecode. The treecode does not compute all particle-particle interactions directly. Instead, it computes partial forces on a given i-particle from a truncated multipole expansion of groups of particles, see section 1.4.2. The force interaction from a group is computed if the group is far enough, according to a Multipole Acceptability Criterion (MAC). Groups become larger and larger as their distance from the i-particle increases. This technique allows a decrease in the computing time of the force evaluation to $\mathcal{O}(N \log N)$, at the cost of a reduced accuracy, due to the truncated multipole expansion. Moreover, this asymptotic performance is reached for large values of N. Because of this, the treecode is widely used to simulate systems like clusters of galaxies, or large scale structures, where high accuracy is not needed, and N is large.

For a sufficiently large problem a treecode can outperform a direct code also for the simulation of systems that require high accuracy. In order to increase the treecode accuracy, we can tune two parameters: the highest term of the multipole expansion, and the MAC

parameter that decides if a group is far enough to compute the interaction. As already discussed in section 1.4.2, the most widely used MAC (see, e.g., Barnes & Hut, 1986) states that a multipole expansion is accepted if

$$\frac{l}{d} < \theta \tag{3.1}$$

where l is the size of the cell containing the group, d is the distance of the i-particle from the cell, and θ is the MAC parameter. For the low accuracy computations that usually involve the treecode, is $\theta \lesssim 1$. A more accurate code will run more slowly.

In order to have a higher accuracy code, θ has to be smaller. A realistic choice for a multipole expansions up to the quadrupole term, is $\theta = 0.2$. For a comparable accuracy with a multipole expansion up to the octupole term, we have to set $\theta = 0.5$ (McMillan & Aarseth, 1993). There is a trade-off between the two choices. A smaller θ implies a much larger amount of interactions to compute; it has been shown that the number of interactions scales as θ^{-3} (Makino, 1991b). On the other hand, a multipole expansion up to the octupole term implies a larger number of computations to obtain the multipole terms, and a larger number of computation to evaluate the force contributions from the octupoles. With our model, we can simulate the two cases, and obtain an indication of the most effective choice. In our simulations, we assume that also the force contributions from the multipoles can be computed on the GRAPE, by means of a pseudo-particle transformation (see chapter 4). In this case, multipole expansions are converted to pseudo-particle distributions that produce the same force. In this way, GRAPE can also compute force contributions from the multipole terms.

Moreover, we compare the performance of two different parallel treecodes, i.e. the HOT and GDT codes described in section 3.3.2. The main difference between the two codes is that in HOT, i.e. the parallel treecode originally developed in Warren & Salmon (1995), each processor computes forces only on the local i-particles. Information about remote particle groups, the so-called local essential tree, is obtained before the force computation starts.

Conversely, in the GADGET code (Springel *et al.*, 2001) (referred here as GDT), processors do not exchange information about remote particle groups. Instead, local i-particles are sent to remote processors. With our model, we can see which approach is better suited for a distributed hybrid architecture.

Our comparison has the goal to assess whether a system size exists at which treecodes outperform the direct code for the simulations of systems requiring high accuracy. Then we use our performance model to find which hardware-software combination gives the best performance, provided that high accuracy is ensured from the treecodes, either by decreasing θ, or increasing the multipole order. Therefore, we choose for each method the most suitable hybrid architecture. Namely, we simulate the direct code running on a system including a single host with 16 GRAPEs attached to it, since the clustered configuration has the best performance, as shown in section 3.4.2. As treecodes place a higher load on the host, the optimal system for them is a distributed 16 processor hybrid machine, each node hosting a GRAPE board. High accuracy from treecodes is obtained by setting $\theta = 0.5$ with octupole term expansion, and $\theta = 0.2$ with expansions up to the quadrupole term.

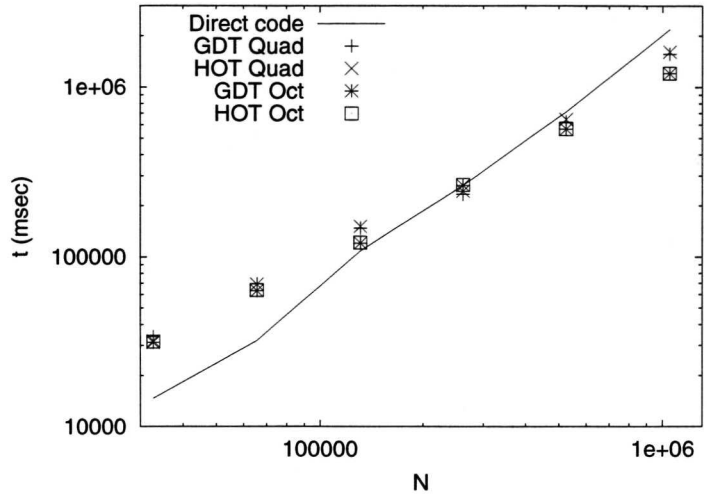


Figure 3.29: Performance of the treecode, compared to the direct code. We report here the simulated execution times of 300 code iterations. The direct code runs on a single host connected to a 16 GRAPE cluster, the treecodes run on a 16 processor machine, each node hosting a GRAPE. `Quad` refers to multipole expansions up to the quadrupole term, with $\theta = 0.2$, and `Oct` to octupole expansions, with $\theta = 0.5$. Other symbols are explained in the text.

Fig. 3.29 shows the results of our simulation. We can see in the figure how the direct code performs better for low N, but is eventually outperformed by the treecodes. The two treecode implementations show a very similar performance. Both perform better than the direct code for high particle numbers, and are faster when an expansion up to the octupole term is used. We can conclude that a distributed hybrid architecture can be the system of choice for the simulation of large astrophysical systems requiring a high accuracy, such as stellar globular clusters. A treecode equipped with the software tools for the accurate treatment of close encounters could supersede the direct code for the realistic simulation of phenomena such as globular cluster secular evolution, or black hole binary formation in merging galaxies (Hut & Makino, 1999).

3.5 Discussion

A hybrid architecture is a system with a high degree of heterogeneity among its components, whose complex interplay requires an appropriate tool in order to be understood and optimised. Performance modelling is an important tool to study the behaviour of such complex systems. We implemented and tested a simulation model able to reproduce the behaviour of hybrid architectures. We validated this model against our GRAPE-DAS system, both

for the serial and the parallel case. We showed some examples of its use for predicting the performance of other configurations, where hardware and/or software modifications have been introduced. We simulated the use of several different hardware systems, and numerical algorithms for the solution of the N-body problem, as the direct particle-particle code, and the treecode. We showed that performance simulation allows us to discover unexpected behaviours of a complex computer system, as in the case described at the end of section 3.4.2. In our case-studies, distributed hybrid architectures show their superior computational potential, as compared to "clustered" configurations, when large problems, at the higher limit of the available computational capability, are considered.

Our research is particularly focussed on the efficient integration of treecodes and hybrid architectures. This could lead to a very high performance computational environment for the solution of the N-body problem. We validated our treecode model by simulating existing configurations and comparing our results to real system measurements, even though very few measurement data were available, limiting the accuracy of our model calibration. We used our model to evaluate the performance of a hybrid architecture used to run the treecode, and highlighted that an efficient implementation of the treecode on such architecture is made difficult by an intrinsically high communication overhead. Issues like latency hiding, or partial redistribution of work to remove load imbalance, could help to solve this problem, and will be the object of further research. The model would also benefit from an accurate parameterisation of load imbalance.

In the next chapter, we describe our research in the framework of the aforementioned efficient integration of treecodes and hybrid architectures. We have implemented a version of the treecode, which makes use of pseudo-particles (Makino, 1999; Kawai & Makino, 2001) in order to represent the multipole expansion of the gravitational potential. This approach allows us to make use of the GRAPE not only for the computation of the force from the monopole term. The pseudo-particle scheme allows the GRAPE to compute force contributions from all terms of the multipole expansion. In chapter 4 the pseudo-particle method will be described, together with the accuracy and performance improvements that we introduced.

Part II

Applications

Chapter 4

Pseudo-Particle Powered Treecode: Error Analysis and Optimisation[†]

In this chapter we study the pseudo-particle scheme, which makes it possible to model higher order multipole moments for the treecode on the GRAPE. The treecode, introduced in section 1.4.2, offers excellent scaling for the simulation of self-gravitating systems, but at the cost of limited accuracy. The pseudo-particle approach, where a multipole expansion is expressed in terms of a particle distribution, provides an accuracy that it is easy to tune, and is suitable for making full use of the ultra fast GRAPE Special Purpose Device for the gravity force computation. The GRAPE is introduced in section 1.3 and extensively studied in part I of this thesis. We study the error behaviour of this approach, comparing it with the standard treecode, and introduce improvements that reduce the errors. Furthermore we present an extension of the pseudo-particle scheme, where pseudo-particles are not fixed in space, but move following the physical particle distribution. This extension decreases the computational overhead due to pseudo-particle recomputation, and optimises the scheme for the use on GRAPE and on parallel systems.

4.1 Introduction

The treecode (Barnes & Hut, 1986), introduced in section 1.4.2, is one of the most popular numerical methods for particle simulation involving long range interactions in astrophysical contexts. Its operation count scales as $\mathcal{O}(N \log N)$, which is a great improvement compared

[†]This chapter is based on work published in:

P.F. Spinnato; S.F. Portegies Zwart; M. Fellhauer; G.D. van Albada and P.M.A. Sloot: *Tools and Techniques for N-body Simulations*, in R. Capuzzo Dolcetta, editor, Proceedings of the 1st workshop on Computational Astrophysics in Italy: Methods and Tools, MemSAIt Suppl. Series vol. 1, pp. 54–65. Società Astronomica Italiana, 2003.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Pseudo-Particle Powered Treecode: Error Analysis and Optimisation*, to be submitted to Journal of Computational Physics, 2003.

with the $\mathcal{O}(N^2)$ scaling of the direct particle-particle method, where the force on a particle is computed by directly evaluating the contributions of all other particles (see section 1.4.1 and 2.2.2). The treecode speedup is obtained at the cost of a reduced accuracy. The treecode groups particles together according to a tree data structure, where each node of the tree is associated with a cubical cell in three-dimensional space, and a cell corresponding to a given node consists of the eight cells corresponding to the eight subnodes hierarchically connected to the given node. The treecode attains its $\mathcal{O}(N \log N)$ scaling by computing force on a particle i from larger and larger cells as their distance from i gets bigger and bigger. In order to decide whether a cell is far enough to be accepted for such force computation, a suitable Multipole Acceptability Criterion (MAC) must be provided. In section 4.3 below MACs are discussed further.

The force contribution is evaluated from a multipole expansion of the particle distribution contained in the cell. The accuracy of the evaluation depends on the highest term in the multipole expansion, on the MAC used, and on the actual value chosen for the MAC parameter. Usually, expansions are truncated at the quadrupole term, leading to an accuracy in the force in the order of 1% (Salmon & Warren, 1994) for commonly used MAC settings. This makes treecodes unsuitable for applications that require a high numerical accuracy. Better accuracies can be obtained by using different MAC settings, but this will greatly increase the computing cost.

Research has been carried out in order to improve the treecode accuracy, by increasing the maximal multipole expansion order (McMillan & Aarseth, 1993). We aim at developing a version of the treecode that allows a tunable accuracy, while limiting the impact on code performance. We design our code in order to be optimally run on a parallel platform that includes the GRAPE boards (Makino & Taiji, 1998). GRAPE, as described in section 1.3, is a special purpose device implementing an array of fully hardwired pipelines, each one computing the gravitational interaction between two particles in a single clock cycle.

Our code derives from the pseudo-particle approach proposed by Makino (1999), and implemented by Kawai & Makino (2001), developing an early idea of Anderson (1992). Makino and Kawai implemented a serial pseudo-particle treecode that makes use of GRAPE (Kawai, 1999; Kawai & Makino, 2001). The algorithm that they propose places the pseudo-particles in fixed positions on a spherical surface surrounding the physical distribution, and computes the pseudo-particle masses as weighted sums of the physical particle masses (see eq. (4.1) below). The pseudo-particle approach allows the treecode to take advantage of both the very high computing speed offered by the GRAPE, and makes it very easy to increase the code accuracy by increasing the maximal multipole order. In the standard treecode, multipoles are computed in terms of series expansions (see, e.g., McMillan & Aarseth, 1993). Mathematical expressions of the higher moment terms are increasingly cumbersome, and not easy to implement. Conversely, in the pseudo-particle scheme, increasing the maximal multipole order of the expansion is simply a matter of increasing the number of pseudo-particles that make up the expansion.

When one runs a code using multipole expansions, such as the treecode, with the GRAPE, a fundamental problem arises. GRAPE can only compute particle-particle interactions, hence the advantage of lumping particles to obtain a single multipole expansion is

wasted: in the standard treecode formulation, such expansions are expressed in terms of spatial derivatives of $1/r^2$, consequently GRAPE cannot compute particle-multipole interactions. This implies that all interactions involving a multipole term must be evaluated by the host computer. The key idea of the pseudo-particle approach is to use the Anderson (1992) formulation for the multipole expansion which, instead of a complicated polynomial, is given in terms of a pseudo-particle distribution. In this way, GRAPE is able to compute also the force contribution from higher multipole terms, since they are now expressed in terms of a particle distribution.

In order to have a quantitative estimate of the accuracy in the pseudo-particle scheme, we have carried out an error analysis study, comparing the accuracy of the pseudo-particle code with two implementations of the standard treecode. We compare our code with the code of Salmon & Warren (1994), and with GADGET (Springel *et al.*, 2001). We introduce a modification in the method that improves its accuracy when the physical particle distribution is highly inhomogeneous. Moreover, we modify the Makino and Kawai pseudo-particle method by introducing pseudo-particle position extrapolation, in order to decrease the overhead due to pseudo-particle re-evaluation. Instead of recomputing the pseudo-particle expansion at each iteration, we extrapolate the pseudo-particle positions for a number of time-steps; in order to accomplish this, we define a pseudo-particle velocity. Such scheme is suited for use on parallel systems hosting GRAPE boards, as discussed in section 3.4.4. We first present the pseudo-particle method, and discuss the improvement that we introduce. We continue with the error evaluation, then present the moving pseudo-particle scheme, and finally discuss our results and future work.

4.2 The pseudo-particle method

The pseudo-particle method approximates the multipole expansion of a given set of particles by means of a pseudo-particle distribution. The pseudo-particle positions are fixed, and lie on a spherical surface surrounding the real particle distribution (Makino, 1999; Kawai, 1999). Optimised distributions, with minimal number of pseudo-particles, have been obtained up to the quadrupole moment (Kawai, 1999; Kawai & Makino, 2001). The mass of each pseudo-particle is given by:

$$M_k = \frac{1}{K} \sum_{1}^{N}{}_j m_j \sum_{0}^{p}{}_l \left(\frac{r_j}{a}\right)^l (2l+1)\, P_l(\cos\gamma_{jk})\,, \tag{4.1}$$

where M_k is the mass of pseudo-particle k, K is the total number of pseudo-particles, N is the number of real particles from which the pseudo-particle expansion is computed, m_j is the mass of the real particle j, p is the maximal order of the multipole expansion, r_j is the norm of the position vector of particle j, a is the radius of the pseudo-particle sphere, P_l is the modified Legendre polynomial of order l, and finally γ_{jk} is the angle between the position vectors of particle j and pseudo-particle k.

As will be shown later, the pseudo-particle approximation based on a spherical distribution suffers from limitations in representing non-uniform mass distributions. In order

to solve this problem, we introduce a simple modification, that substantially improves the accuracy of the method. It consists in retaining the spherical pseudo-particle distribution, with pseudo-particle masses that now represent only the higher multipole moments of the real particle distribution $\mathcal{R}$, starting from the quadrupole moment. An extra pseudo-particle is added at the centre of mass of $\mathcal{R}$, with mass equal to the total mass of $\mathcal{R}$. The extra pseudo-particle accounts for the monopole and dipole moment, and improves the ability of the pseudo-particle distribution to represent non-uniform real distributions.

The higher order expansion of $\mathcal{R}$, i.e. the part that does not contains the monopole and dipole terms, is obtained with a simple expedient. It consists in computing the pseudo-particle masses still using equation (4.1), but adding to the N particles of $\mathcal{R}$ a "virtual" particle which has the effect of removing the monopole and dipole moments from the expansion of $\mathcal{R}$. This virtual particle is placed at the centre of mass of $\mathcal{R}$, and has a mass equal to the opposite of the total mass of $\mathcal{R}$. The pseudo-particle distribution resulting from this combination of the N real particles and the negative mass particle accounts for the higher order expansion of $\mathcal{R}$. Finally, we obtain the complete multipole expansion by adding to this pseudo-particle distribution the extra pseudo-particle located at the centre of mass of $\mathcal{R}$.

4.3 Error evaluation

4.3.1 Comparisons

First we validate our implementation of the unmodified pseudo-particle method against the implementation of Kawai and Makino. In Kawai & Makino (1999) the error on the potential generated by pseudo-particle expansions up to a given multipole order p is presented. The real particle distribution consists of a single particle i of unit mass placed at position $\mathbf{p} = (1, 0, 0)$ in spherical coordinates. The potential $\Phi = -1/|\mathbf{r} - \mathbf{p}|$ generated by i is computed along a straight line, at points $\mathbf{r} = (r, 2\pi/3, 0)$, for r varying within a certain range. The relative error $|(\Phi_p - \Phi)/\Phi|$, where Φ_p is the potential given by the pseudo-particle expansion up to order p, is presented in fig. 4.1 for both our implementation and the one of Kawai & Makino (1999).

Our values, labelled "PP", are in very good agreement with Kawai and Makino's values, labelled "KM". Irregularities in the error profiles lead to local differences, arguably due to differences in the exact positions of the pseudo-particles between the two implementations, which result in local differences in the value of the potential. The global trend is however very similar in the two cases.

In the next section we will study the worst-case error behaviour of our implementation, comparing our results with a similar analysis carried out in Salmon & Warren (1994).

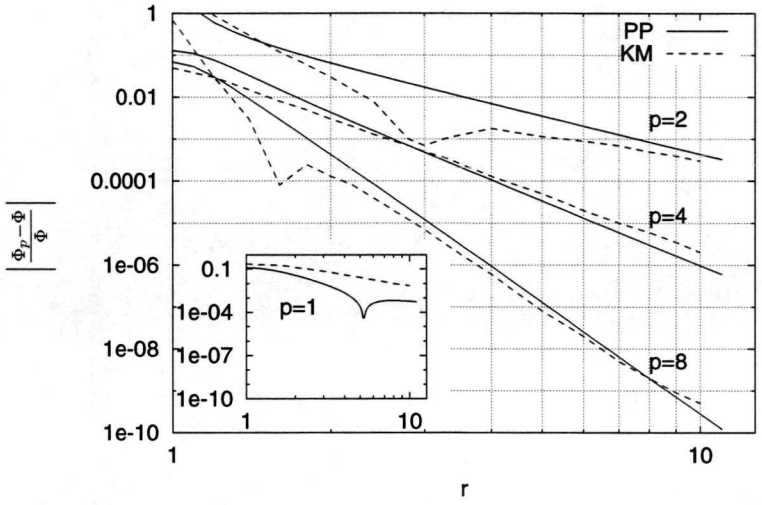


Figure 4.1: Relative error in the evaluation of the potential generated by a unit mass particle placed at position $\mathbf{p} = (1, 0, 0)$ in spherical coordinates. The error is measured at position $\mathbf{r} = (r, 2\pi/3, 0)$, and plotted as a function of r. Values from our implementation are plotted with solid lines, and labelled as PP. Values from Kawai & Makino (1999) are plotted with dashed lines, and labelled as KM. p is the maximal multipole expansion order. The pseudo-particle sphere radius is $a = 1$.

4.3.2 Worst-case error

It is important to analyse the behaviour of the method in the worst case configuration, i.e. the situation leading to the highest error in a force evaluation, even though this situation is unlikely to arise in actual simulations. This analysis gives upper bounds to the code error, and provides a good test for comparative analysis of different multipole acceptability criteria. In order to perform the worst-case analysis of our code, we follow the same procedure as Salmon & Warren (1994) (referred as SW hereafter). The worst-case configuration consists of two point particles placed at two opposite corners of a cubic cell.

Preliminary tests showed that the highest error occurs when the mass m_1 of the particle closer to the evaluation point is much lower than the other particle mass. We set then $m_1 = 10^{-5}$ and $m_2 = 1 - m_1$. We compute the gravitational acceleration exerted by the two particles along a straight line overlapping with the diagonal of the cell where the particles are located. We compute both the exact acceleration $\mathbf{a}$, and the acceleration given by the pseudo-particle expansion up to a given multipole, $\mathbf{a}_p$. From that, the error is evaluated as:

$$\epsilon_p = \frac{|\mathbf{a} - \mathbf{a}_p|}{|\mathbf{a}|}. \tag{4.2}$$

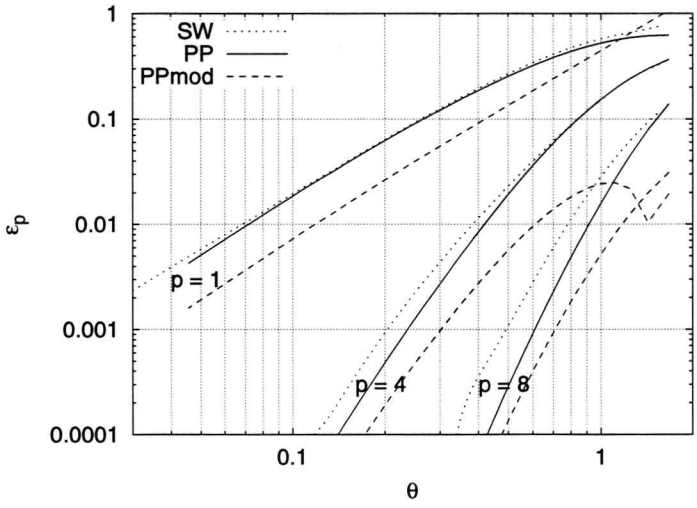


Figure 4.2: Comparison of the worst-case errors in the acceleration. Data are plot as a function of the opening parameter θ. Results from our implementations are labelled PP for the canonical pseudo-particle method, and PPmod, for the modified method where an extra pseudo-particle is added. The Salmon & Warren (1994) results are labelled SW; p is the maximal multipole expansion order.

The Multipole Acceptability Criterion (MAC) we adopt is the Minimal Distance (MD) criterion (Salmon & Warren, 1994). According to the MD MAC, a multipole expansion is accepted if

$$\frac{l}{d} < \theta \tag{4.3}$$

where l is the cell size, d is the minimal distance of the evaluation point from the cell, and θ is an input parameter, usually $\theta \lesssim 1$. The original Barnes & Hut (1986) MAC differs from the MD MAC in the definition of d. Barnes and Hut define d as the distance of the evaluation point from the centre of mass of the cell.

We evaluated the error defined in eq. (4.2) for $p \in \{1, 2, 4, 8\}$, and compared our results with the results presented by SW, fig. 5. The results are shown in fig. 4.2. Data are plotted as a function of the opening parameter θ, in order to show what is the largest error to be expected for a given value of θ. We plot our results, labelled "PP", and SW's results, labelled "SW". Data for the case $p = 2$ are omitted for the sake of readability. We also plot the error of our modified pseudo-particle method, described above. Results from this method are labelled "PPmod". Since our modified method adapts very well to highly non-uniform distributions, the distribution used for the PP method is not the worst case for the PPmod method. Numerical tests showed that the worst case is now when m_1 is about one order of magnitude smaller than m_2, with very little dependence of the errors on the precise value of the masses. We thus chose $m_1 = 0.1$. In all cases, the cell size is $l = 1$, and the

pseudo-particle sphere radius is $a = 1$.

The PP curves are in very good agreement with the SW curves, with a tendency for the PP curves to have smaller errors for lower values of θ in the high precision cases ($p \in \{4, 8\}$). The agreement of our results with the MD case of Salmon and Warren is not surprising, since we adopt the same criterion and the same geometry for the error analysis. The PPmod results are always better than the PP and SW cases, especially for the low precision cases. The improvement obtained with the PPmod method will be also observed in the statistical error analysis.

4.3.3 Statistical error

We compare the statistical error of the pseudo-particle code with the standard treecode results presented in Salmon & Warren (1994). We use this implementation as our benchmark, since multipole terms are computed there with the standard method, i.e. by means of series expansions. Specifically, we compare our results with the isolated halo case of SW. In that experiment, 4942 particles were chosen at random from a high density core distribution, and the error analysis was performed on them, using eq. (4.2) for the error estimation. Our results, obtained using the MD MAC (in)eq. (4.3), and opening angle $\theta = 1.1$, are compared with the same case presented in Salmon & Warren (1994), fig. 11. The configuration that we used includes 4096 particles. In our implementation of the pseudo-particle treecode, each non-terminal cell is associated with a pseudo-particle distribution located on the surface of a sphere whose radius is one half of the cell size. A sphere radius smaller than the cell size gives a better accuracy (Kawai, 1999), and preliminary tests gave us one half of the cell size as the optimal value for the sphere radius. The pseudo-particle distribution of a parent cell is obtained recursively from the distributions of its child cells. A particle-cell interaction now becomes a set of particle-particle interactions between the particle and the pseudo-particle distribution of that cell. The multipole expansion is computed up to the quadrupole moment.

The error distribution of the pseudo-particle scheme is shown in fig. 4.3. We computed the relative error (eq. (4.2)) for the force on each particle, then obtained the cumulative percentile distribution shown in the figure. This method of analysing the error gives much more insight into the accuracy of the code, than for instance rms or maximal error. An optimal code has a flat error distribution, so that the great majority of errors have about the same value. A code with a wide spread in error values leads to a waste of compute time, since increasing the accuracy in order to reduce large errors, results in unnecessary refinement for those force computations whose error was already small. See Salmon & Warren (1994) for a more extensive discussion.

The "PP" case in fig. 4.3 shows how much larger are the errors in the unmodified PP code with respect to the standard code. These tests are performed on a dark halo distribution (Hernquist, 1990), which is a highly inhomogeneous particle distribution, having a radial density $\rho(r) \propto [r \cdot (1 + r^3)]^{-1}$.[1] A dark halo is the result of the gravitational collapse

[1]Nowadays $\rho(r) \propto [r \cdot (1 + r)^2]^{-1}$ (Navarro *et al.*, 1997) is the most accepted density profile. We chose to use the other profile, to be consistent with the profile used by Salmon & Warren (1994).

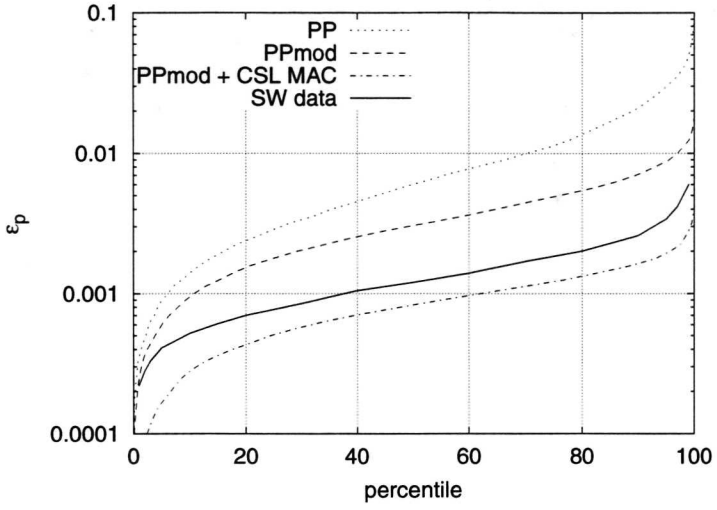


Figure 4.3: Error distribution for the PP code, compared with Salmon & Warren data (Salmon & Warren, 1994). "PP" refers to the unmodified code, "PPmod" refers to the code with the extra pseudo-particle placed at the centre of mass. "PPmod + CSL MAC" refers to the modified PP code with the MAC according to (in)eq. 4.4. The multipole expansion is up to the quadrupole moment.

of intergalactic gas due to random density peaks. Tests performed with more homogeneous particle distributions showed that the PP code accuracy is much better is those cases. The same tests, performed with the standard code, showed that the canonical treecode is less sensitive to the spatial distribution of the particles. Indeed, errors are higher for the uniform distribution, which, conversely, is the best case for the pseudo-particle code.

The pseudo-particle expansion accuracy suffers strongly from a non-uniform distribution of particles. For this reason, as already mentioned, we modified it by introducing in the multipole expansion an extra pseudo-particle located at the centre of mass of the distribution. This allows us to represent highly inhomogeneous distributions much better. The "PPmod" case in fig. 4.3 shows the error behaviour of our modified pseudo-particle treecode. The errors are now much smaller than the ones for the unmodified code. Yet, errors are still higher than the ones of the standard treecode.

We observed that most of the large errors come from distant cells. In order to control this error, we modified the MAC in such a way that a multipole expansion is accepted if:

$$\frac{l}{\frac{d}{\sqrt{d}+1}} < \theta \quad , \tag{4.4}$$

where symbols have the same meaning as in (in)eq. (4.3). The opening criterion in (in)eq. (4.4) reduces the acceptability of far-away (hence large) cells, at the cost of an increased computational load. This new Cell Size Limiting (CSL) criterion, applied to the modified pseudo-particle code, gives a remarkable improvement in the code accuracy, as shown in

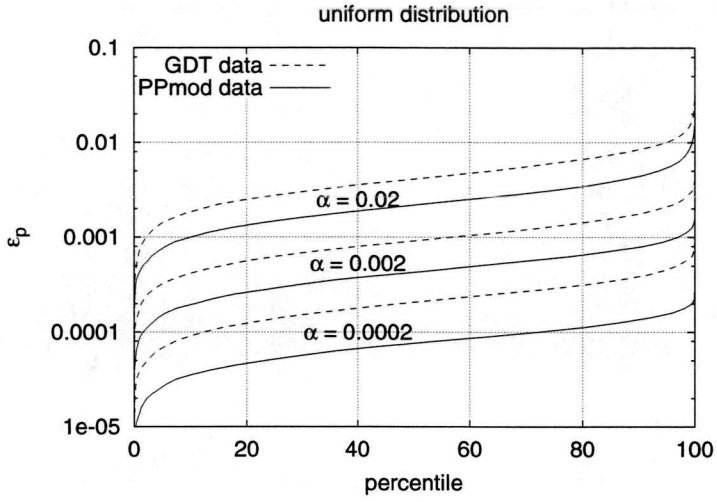


Figure 4.4: Comparison of the error distributions for our code (labelled PPmod) and the treecode GADGET (labelled GDT), where multipoles are computed in the standard way. The MAC of (in)eq. 4.5 is used here. For consistency with the standard treecode, cell-particle distances in our code are not minimal distances as in the previous cases, but are measured with respect to the cell centre of mass. The multipole expansion is up to the quadrupole moment.

the "PPmod + CSL MAC" case in fig. 4.3. Now the error of the pseudo-particle code is below the error of the standard treecode.

An extra factor that increases the pseudo-particle code accuracy is the fact that in our implementation a multipole contribution is evaluated only if there are more particles in the cell than the pseudo-particles used to represent the multipole expansion. In the present case, with expansions truncated at the quadrupole term, each expansion has 13 pseudo-particles. The pseudo-particle code accuracy benefits from this, since force from cells containing 13 particles or fewer is always computed exactly. Moreover, we also gain in performance, since fewer interactions are computed in this way. The effect of this feature of the pseudo-particle code will be studied further in next section.

4.3.4 The GADGET MAC

The criterion in (in)eq. (4.3) and the modified CSL version in (in)eq. (4.4) are based on the principle that the error from a certain cell will be small if that cell is "seen" under a small opening angle. A criterion that directly estimates the error that the cell expansion will introduce if accepted, could lead to a more efficient MAC. This approach was proposed in Salmon & Warren (1994), and is implemented in the recently developed treecode GADGET (Springel *et al.*, 2001), which we already studied in our performance modelling studies presented in chapter 3. According to the MAC discussed in Springel *et al.* (2001), a multipole expansion

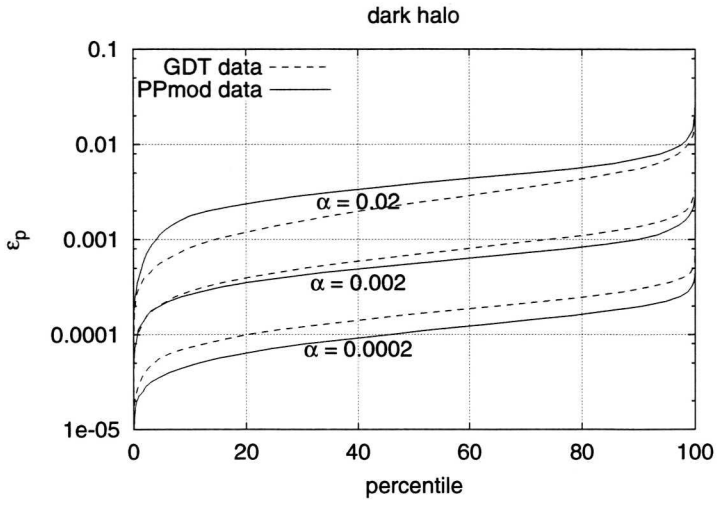


Figure 4.5: Comparison of the error distributions for a highly clustered configuration. Symbols are the same as in fig. 4.4. The multipole expansion is up to the quadrupole moment.

is accepted if:

$$\frac{Ml^4}{d^6} < \alpha |\mathbf{a}_{old}| \quad , \tag{4.5}$$

where M is the cell mass, l is the cell size, d is the particle-cell distance, α a numerical coefficient, and $\mathbf{a}_{old}$ the previous value of the acceleration on the particle currently dealt with. The left hand side of the above expression can be seen as a rough estimate of the force contribution from the hexadecapole moment of the cell (Springel *et al.*, 2001), and $\mathbf{a}_{old}$ is an estimate of the true current value of the particle acceleration. An estimate of the error introduced by truncating a multipole expansion at a certain order, is given by the contribution of the first term not included in the expansion. In the case of GADGET, the truncation is at the quadrupole term. The octupole term should then be chosen. However, the octupole term vanishes for uniform distributions, in those cases the hexadecapole term should be used. The authors chose to use the estimate of the hexadecapole term contribution in all cases. This improves accuracy, and is also cheaper to compute, because it does not involve square root evaluations. The criterion in (in)eq. (4.5), states that a cell is accepted if the estimate of the hexadecapole term contribution is less than a small fraction of the total force on the particle. If higher accuracy is required, the estimate of a higher multipole term should be used in the left hand side of (in)eq. (4.5). This MAC opens a cell only if the expected error from the cell is large. Nearby cells that would be opened with the canonical MAC (in)eq. (4.3) because they are "seen" under a large opening angle, now are not opened if their effect on the total force error is small. Because $\mathbf{a}_{old}$ is not defined in the first code iteration, the very first force evaluation is still performed using the canonical MAC.

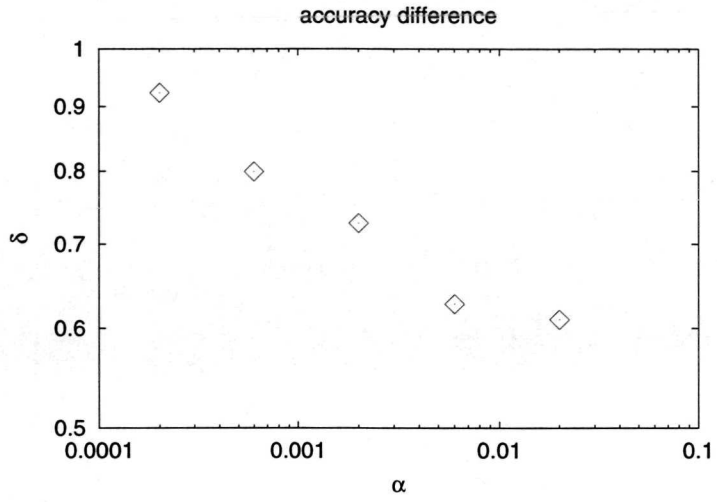


Figure 4.6: Relative difference between the error values of the two codes, measured at the 50% percentile, as a function of the accuracy parameter α.

We implemented the criterion defined in the (in)eq. 4.5, and compared our results with the standard treecode for two different particle configurations: a uniform distribution, and a highly concentrated (dark halo) distribution. Our results are presented in fig. 4.4 and 4.5 for three representative values of α. Except for the low accuracy case in the highly concentrated distribution, the pseudo-particle code shows a better accuracy than the GADGET code, especially in the uniform distribution case. This confirms the tendency of the pseudo-particle code to give better results with homogeneous distributions.

In the uniform distribution case, the relative separation

$$\delta = \frac{|\epsilon_p^{GDT} - \epsilon_p^{PP}|}{(\epsilon_p^{GDT} + \epsilon_p^{PP})/2} \tag{4.6}$$

between the results of the two codes seems to be dependent on α. In fig. 4.6 we show the values of δ, measured according to eq. (4.6) at the 50% percentile value of ϵ_p. Measures for two more values of α have been added in this case. The relative difference between the two codes tends to decrease and saturate with α. This can be explained by the fact that a smaller value of α causes a smaller size of the accepted cells. Smaller cells contain fewer particles, and if the number of particles is 13 or less (see discussion at the end of previous section), the PPmod code computes forces from the cell directly, hence with perfect accuracy. This explains why the accuracy of the PPmod code is higher for smaller values of α.

The effect of this feature of the PPmod code will be less pronounced if the number of particles is increased. In this case, fewer cells, among those that pass the MAC (in)eq. (4.5) will contain 13 particles or fewer. If the total number of particles is increased by a factor n, the number of particles in a given cell is also increased by the same factor. The fraction of

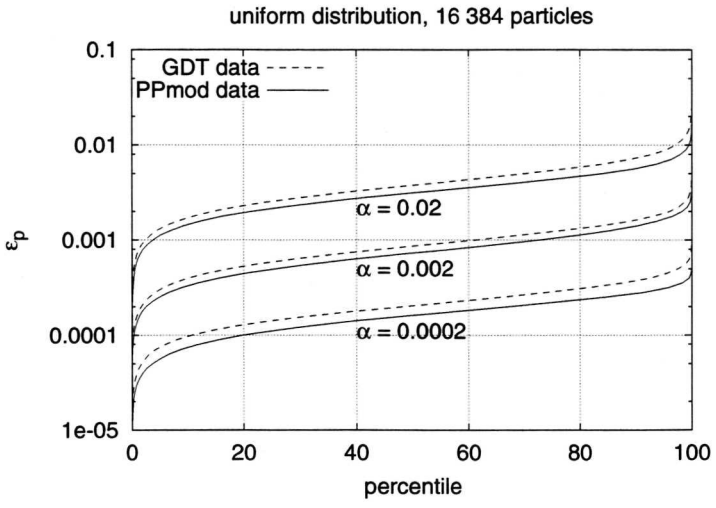


Figure 4.7: Comparison of the error distributions for a uniform distribution, and 16 384 particles. Symbols are the same as in fig. 4.4. The multipole expansion is up to the quadrupole moment.

accepted cells whose force will be computed exactly by the PPmod code will then decrease, so that the accuracy gain of the PPmod code with respect to the standard code will decrease. Fig. 4.7 shows a comparison of the PPmod code and the standard code with 16 384 particles. The separation between the results of the two codes is clearly smaller with respect to the one in fig. 4.4. For larger numbers of particles the separation between the two codes is likely to become negligible.

We used our pseudo-particle code to compare the two MACs of (in)eqs. (4.4) and (4.5), in order to show the error profile that they produce, as a function of the accuracy parameter. Our results are presented in fig. 4.8. We compare the two MACs using cases having the same accuracy, measured according to the respective accuracy parameters, i.e. the opening angle θ of the CSL MAC (see (in)eq. (4.4)), and the accuracy parameter α of the GADGET MAC (see (in)eq. (4.5)). It is clear how the GADGET MAC gives better results in terms of flatness of the error profile. The total computational load, measured in terms of the mean number of interactions per particle κ, is of the same order for cases having comparable accuracy.

4.4 Moving pseudo-particle scheme

The evaluation of the pseudo-particle masses is computationally expensive, especially when a high multipole order is required (Kawai, 1999; Makino, 1999). Moreover, when the GRAPE hardware is used, the recomputed pseudo-particle data must be reloaded at each iteration. This introduces an high overhead, limiting the convenience of the pseudo-particle approach. We propose a scheme that does not require a re-evaluation of the pseudo-particle masses

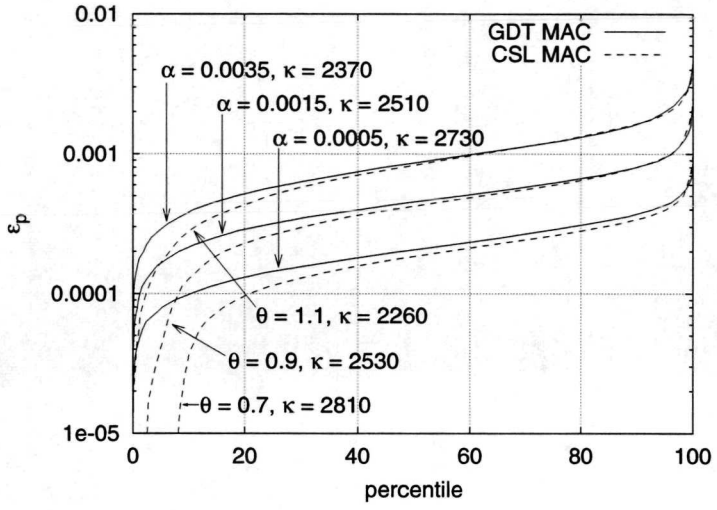


Figure 4.8: Comparison of the error profiles of the two MACs of (in)eqs. (4.4) and (4.5). θ is the opening angle of the CSL MAC of (in)eq. (4.4), α is the accuracy parameter of the GDT MAC of (in)eq. (4.5), κ is the mean number of interactions per particle.

at each iteration. Instead, we assign a velocity to the pseudo-particles, and let them move with this velocity. Velocities must be assigned so that the pseudo-particles' motion correctly reproduces the changes in the moment distribution for each cell. The advantage of this approach is that pseudo-particle data must be recomputed less frequently. Moreover, when GRAPE is used, no reload is necessary, since GRAPE contains the hardware needed to extrapolate particle positions. We compute the pseudo-particle momenta adapting the formula used to compute the pseudo-particle mass eq. (4.1):

$$\mathbf{p}_k = \frac{1}{K} \sum_{1}^{N}{}_j \, \mathbf{p}_j \sum_{0}^{P}{}_l \left(\frac{r_j}{a}\right)^l (2l+1) \, P_l(\cos\gamma_{jk}) \quad . \tag{4.7}$$

Here $\mathbf{p}_k$ is the momentum of pseudo-particle k, $\mathbf{p}_j$ is the momentum of the real particle j, all other symbols have the same meaning as in eq. (4.1).

We present below a statistical error analysis of our moving pseudo-particle scheme, similar to the analysis presented in the previous section.

4.4.1 Statistical error

In order to carry out a statistical error analysis of the moving pseudo-particle scheme, we used here a dark halo configuration. In this case, we used a configuration with 40 000 particles, in order to reduce the fraction of cells containing 13 particles or less. In this way we increase the

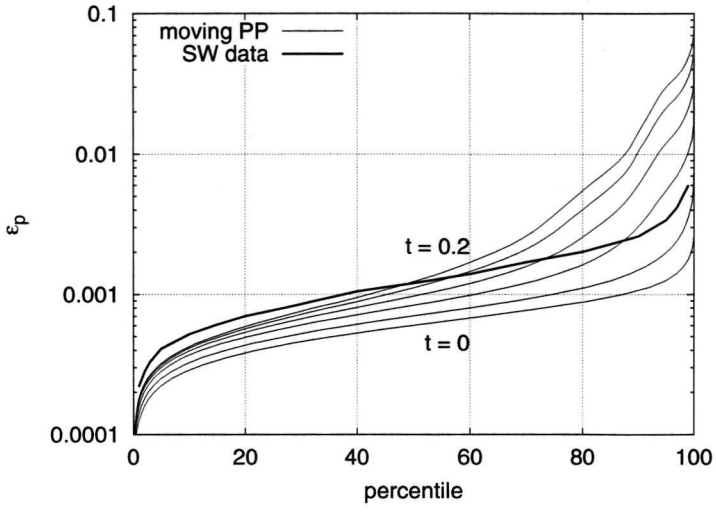


Figure 4.9: Error distribution for the moving pseudo-particle code. The error profile at various extrapolation times is compared with a canonical treecode error profile. We use the MAC defined in eq. (4.5), with $\alpha = 0.002$. The time interval between each error profile is equal to 0.04. In this case is $N = 40\,000$.

fraction of contributions from multipoles with respect to contributions from single particles. For each cell, the pseudo-particle expansion is computed only at the first iteration. After that pseudo-particle positions are extrapolated using the velocities obtained according to eq. (4.7). We set the time step $\Delta t = 0.01$, to be compared with $\min(|\mathbf{v}|/|\mathbf{a}|) \simeq 0.04$.

Fig. 4.9 shows the error percentiles for the configuration as a function of time, compared with the SW data used as a benchmark in the previous sections. Initially the extrapolated pseudo-particles reproduce the real particle dynamical configuration with an accuracy comparable with the static pseudo-particle expansion (the error profile after 0.04 time units is very close to the initial profile). Subsequently, accuracy worsens, and after 0.2 time units errors are considerably larger, with a fraction of large errors increasingly bigger (i.e. an error profile increasingly steeper). From $t = 0$ to $t = 0.2$, the particles having $\epsilon_p < 0.002$ decrease from 99.53% to 64.35%. Particles contributing with large errors to the error profile in fig. 4.9 are those located in the inner core of the distribution. Those are also the particles experiencing higher accelerations. This suggests that the pseudo-particle temporal expansion could be improved by adding a term, similar to eq. (4.7), that accounts for the pseudo-particle acceleration.

4.5 Discussion

In this chapter we presented an error analysis of both the pseudo-particle treecode, and the moving pseudo-particle scheme that we propose to reduce the compute time, and optimise the use of this method with the GRAPE. We showed the error behaviour of this method, and compared it with previous work on the standard treecode.

We modified the pseudo-particle distribution, by adding an extra pseudo-particle located at the cell centre of mass, which led to a one order of magnitude error decrease. Yet, the pseudo-particle scheme tends to be more accurate with homogeneous particle configurations. This is promising for the simulation of systems subject to the Coulomb force, which are usually characterised by nearly uniform density distributions. We also introduced a temporal expansion for the pseudo-particle scheme and showed that, as long as particles are not subject to high accelerations, the error remains close to the error of the standard case up to about 20 time steps. In order to improve the accuracy for particles with high acceleration, we are extending the pseudo-particle temporal expansion to include an acceleration term.

The implementation of a pseudo-particle treecode fine-tuned for the use of the GRAPE is currently under development. The internal architecture of a GRAPE board, where an array of pipelines (up to 96 for GRAPE-4 (Makino *et al.*, 1997), and up to 48 for the most recent GRAPE-6 (Makino *et al.*, 2000), see section 1.3) computes force concurrently on an equal number of particles, is an ideal hardware counterpart of the algorithmic strategies developed to group particles together in order to use the same list of force sources, namely the sinking strategy in Warren & Salmon (1995), or the equivalent grouping strategy in Barnes (1990). Moreover, the internal particle memory of the GRAPE board acts as a cache, which can contain up to about 44 000 particles for the GRAPE-4 (Kawai *et al.*, 1997), and 262 000 particles for the GRAPE-6 (Makino, 2003). Therefore the caching strategies developed in, e.g., Salmon & Warren (1997), where force sources are carefully grouped in logical pages such that data in the same page are likely to be accessed shortly, can be applied here in a natural way. The same force sources set can be used for several reloads of the GRAPE pipelines, and this could considerably reduce the host-GRAPE communication overhead. The use of the pseudo-particle method with the GRAPE can be improved by our pseudo-particle extrapolation scheme. Moreover, this scheme is also suitable for optimal parallelisation, because it allows to retain the multipole expansion of remote cells for a number of iterations, resulting in a substantial decrease of communication among processors.

We aim at using our moving pseudo-particle scheme as the computational core of a parallel treecode running on a hybrid architecture that includes the GRAPE. Performance simulations of this kind of hardware/software configuration were presented in section 3.4.4. We showed there that a pseudo-particle powered treecode running on the GRAPE can outperform the direct code even in case of high accuracy simulations, since the pseudo-particle treecode is able to use the GRAPE also for the evaluation of the force contribution from higher-order multipole terms. In chapter 5, we present a stellar dynamics study of a black hole spiralling in towards the Galactic centre. This study is a first step in the direction of simulating the infall of a star cluster. As discussed in chapter 5 and in section 1.7 above, this problem is very difficult to treat using either the treecode or the direct code. We plan

to develop a hybrid code to solve it. The pseudo-particle treecode is very well suited for playing the role of the treecode "phase" of this hybrid code, especially in view of using the hybrid code on a hybrid architecture including GRAPEs.

Chapter 5

The Efficiency of the Spiral-in of a Black Hole to the Galactic Centre[†]

In this chapter, we use the direct particle-particle method, the treecode, and the particle-mesh code, introduced in section 1.2 and 1.4, to study the efficiency at which a black hole or dense star cluster spirals in to the Galactic centre. As introduced in section 1.7, this process is driven by a drag force, called dynamical friction, that results from the combined gravitational pull exerted by a star distribution on a massive body moving through the system.

This phenomenon takes place on a dynamical friction time scale, which depends on the value of the so-called Coulomb logarithm ($\ln \Lambda$). We determine the accurate value of this parameter using the three methods mentioned above with up to two million plus one particles. We find that the three different techniques are in excellent agreement. Our result for the Coulomb logarithm appears to be independent of the number of particles. We conclude that $\ln \Lambda = 6.6 \pm 0.6$ for a massive point particle in the inner few parsec of the Galactic bulge. For an extended object, like a dense star cluster, $\ln \Lambda$ is smaller, with a value of the logarithm argument Λ inversely proportional to the object size.

5.1 Introduction

The region near the Galactic centre is populated by very young objects, such as the Quintuplet star cluster (Nagata *et al.*, 1990; Okuda *et al.*, 1990), the Arches cluster (Nagata *et al.*, 1995) and the central star cluster (Tamblyn & Rieke, 1993; Krabbe *et al.*, 1995), which are of considerable interest for the astronomical community. One of the more interesting conundrums is the presence of stars as young as few Myr (Tamblyn & Rieke, 1993;

[†]This chapter is based on work published in:
P.F. Spinnato; M. Fellhauer and S.F. Portegies Zwart: *The Efficiency of the Spiral-in of a Black Hole to the Galactic Centre*, Monthly Notices of the Royal Astronomical Society, in press, 2003.

Krabbe *et al.*, 1995) within a parsec from the Galactic centre (Gerhard, 2001). *In situ* formation is problematic, due to the strong tidal field of the Galaxy, which makes this region inhospitable for star formation. One possible solution is provided by Gerhard (2001), who proposes that a star cluster of 10^6 $M_\odot$, where $M_\odot$ is a solar mass, spirals in to the Galactic centre within a few million years from a distance $\gtrsim 30$ pc. The infall process is driven by dynamical friction (Chandrasekhar, 1943). A quantitative analysis of this model by McMillan & Portegies Zwart (2003) confirms Gerhard's results. The main uncertainty in the efficiency of dynamical friction, and therewith the time scale for spiral-in, is hidden in a single parameter, called the Coulomb logarithm $\ln\Lambda$. Accurate determination of this parameter is crucial for understanding this process. Nevertheless, a precise value of $\ln\Lambda$ for the Galactic central region is not available. In the work presented in this chapter, we determine $\ln\Lambda$ for the Galactic centre. We focus on the efficiency of the interaction between an intermediate mass black hole (BH hereafter) and the stars in the Galactic central region. In section 5.4 we comment on how this approach can be applied to star clusters that sink to the Galactic centre.

Dynamical friction is important for a large variety of astronomical phenomena, e.g. planet migration (Goldreich & Tremaine 1980; Cionco & Brunini 2002), core collapse in dense star clusters (Portegies Zwart *et al.*, 1999) or mergers in galaxy clusters (Makino 1997; Cora *et al.* 1997; van den Bosch *et al.* 1999). The physics of the infall process of a satellite in the parent galaxy is basically the same as in the case of a BH spiralling in to the Galactic centre. The relevant parameters, however, are quite different in the two cases. For example, an inspiraling galaxy has finite size, whereas a BH is a point mass. Dynamical friction also plays an important role in the evolution of the black hole binary formed after the merging of two galaxies both hosting a BH at their centre (Milosavljević & Merritt, 2001). In this case, dynamical friction is important in the early phase of galaxy merging, when the BHs orbits converge and become bound.

In the classical study of Chandrasekhar (1943), dynamical friction is driven by the drag force experienced by a point mass moving through an infinite medium of homogeneous density. The consequences of finiteness and non-homogeneity have been analysed in various works (see Maoz 1993; Milosavljević & Merritt 2001). Just & Peñarrubia (2003) carried out an analytical study of dynamical friction in inhomogeneous systems, leading to a value of the Coulomb logarithm that depends on the infalling object position. Colpi & Pallavicini (1998) developed a general theoretical framework for the interaction of a satellite with a primary galaxy, able to describe dynamical friction in finite, inhomogeneous systems. They applied their theory of linear response to orbital decay of satellites onto a spherical galaxy (Colpi, 1998) and short-lived encounters with a high-speed secondary (Colpi & Pallavicini, 1998). They studied evolution of satellites in isothermal spherical haloes with cores (Colpi *et al.*, 1999), extended in Taffoni *et al.* (2003), treating satellite finite size and mass loss. Still, the original expression of Chandrasekhar is used to model dynamical friction in many astronomical situations (see Binney & Tremaine 1987, § 7.1; Hashimoto *et al.* 2003). The cases we study here are characterised by a point mass, with a very small mass compared to the primary system. Therefore Chandrasekhar's formulation is appropriate in our cases.

We determine the value of $\ln\Lambda$ for a BH spiralling-in to the Galactic centre by means

of self-consistent N-body simulations. This is by far not an easy task. N-body models either lack in the number of particles (a direct N-body code can treat up to about 10^5 particles, compared to 10^8 for the real system) or have to introduce softening (Aarseth, 1963) and approximation of the force calculation (treecode (Barnes & Hut, 1986) or particle-mesh code (Hockney & Eastwood, 1988)). The softening parameter ϵ was introduced to limit the strength of the mutual gravitational interaction during close stellar encounters. Without softening, the very high accelerations experienced by the encountering bodies would cause very tiny integration steps, which would result in a n effective freeze of the global system evolution, with consequent dramatic performance degradation. The use of this approximation should not invalidate the numerical results, as long as the simulated system is studied on a time scale shorter than the relaxation time scale (Binney & Tremaine 1987, ch. 4, see also discussion in section 5.3.4 below). The dynamical friction time scale of the systems we simulate is in all cases shorter than the relaxation time scale, so we can safely use the approximate methods.

Nevertheless, since close encounters have an important effect on dynamical friction, decreasing their strength by means of softening also decreases the strength of dynamical friction, i.e. lowers the value of $\ln \Lambda$. The same role of softening is played, in the particle-mesh code, by the grid cell size l.

Our methodological approach for the present work (see fig. 5.1) consists of comparing the "exact" results obtained with the direct method for low particle numbers (up to 80 000) with the results of the treecode, which are less accurate and are influenced by force softening, to understand how the softening ϵ influences the results and how they have to be scaled according to the value of ϵ. Then the results of the treecode are compared to the results of the particle-mesh code, to see how softening (tree) and grid-resolution l (particle-mesh) can be compared and scaled. Finally, having the right scaling between the different codes, we will be able to perform high particle number simulations (up to $4 \cdot 10^7$) with the particle-mesh code to obtain the value of the Coulomb logarithm for the inner Galactic Bulge.

5.2 Methods and model

5.2.1 Direct method

For our direct N-body calculations we used the `kira` integrator module of the Starlab software environment[1] (Portegies Zwart *et al.*, 2001), introduced in section 1.4. Conceived and written as an independent alternative to Aarseth's NBODY4 and NBODY5 (Aarseth, 1985, 1999), the workhorses of collisional N-body calculations for the past 25 years, `kira` is a high-order predictor-corrector scheme designed for simulations of collisional stellar systems. This integrator incorporates a Hermite integration scheme (Makino & Aarseth, 1992) and a block time step scheduler (McMillan, 1986) that allows homogeneous treatment of all objects in the system.

While `kira` is designed to operate efficiently on general-purpose computers, it achieves

[1] See: `http://manybody.org`

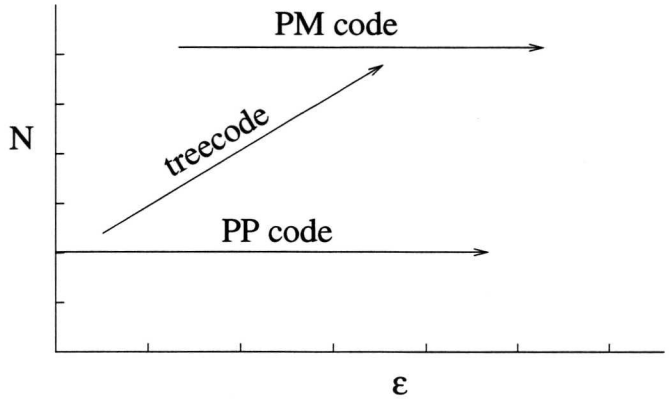


Figure 5.1: A sketch of the strategy that we adopt in order to explore the ϵ-N parameter space.

by far its greatest speed when combined with GRAPE-6 special purpose hardware[2] (see section 1.3). For the work presented here we performed simulations with the GRAPE-6 system at the University of Tokyo with up to 80 000 particles.

5.2.2 Treecode

The hierarchical treecode is widely used for the simulation of collisionless systems. We described it in section 1.4.2, and studied it extensively in chapter 3 and 4. Our treecode simulations were initially performed with both a code written by Jun Makino (Makino, 1991b), and with GADGET (Springel *et al.*, 2001). We also used GADGET in the performance simulation work described in section 3.4.3, and in the pseudo-particle treecode accuracy analysis in section 4.3.4. In GADGET each particle is assigned an individual time-step, and at each iteration only those particles having an update time below a certain time are selected for force evaluation. This criterion was originally introduced in the direct N-body code (see section 2.3.1).

This code is parallelized using MPI (Message Passing Interface Forum, 1997). In the parallel version, the geometrical domain is partitioned, and each processor hosts the particles located in the domain partition assigned to it. The computation of forces on the selected i-particles is performed by scattering the particle data to remote processors. Then partial forces from the particles hosted by the remote processors are computed locally. Finally, calculated forces are received back by the i-particle host, and added up resulting in the total force on the i-particles. We run our parallel treecode simulations on the DAS-2 distributed supercomputer, mentioned in section 2.2.1.

[2]See: `http://www.astrogrape.org`

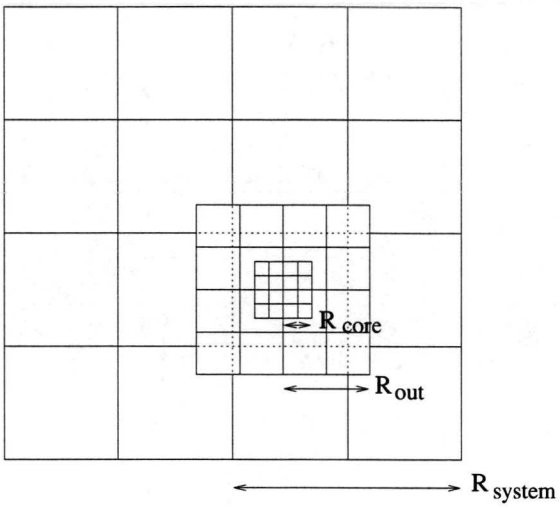


Figure 5.2: The different grids of SUPERBOX for a number of cells per dimension $n = 4$. The finest and intermediate grids are focussed on the object of interest. Each grid is surrounded by a layer of two halo cells. Such haloes are not shown here.

5.2.3 Particle-mesh code

To perform calculations using several millions of particles we use a particle-mesh (PM) code named SUPERBOX (Fellhauer *et al.*, 2000). As mentioned in section 1.2, in the particle-mesh technique densities are derived on Cartesian grids. Using a fast Fourier transform algorithm these densities are converted into a grid-based potential. Forces acting on the particles are calculated using these grid-based potentials, making the code nearly collisionless. SUPERBOX in particular completely neglects two-body relaxation, causing it to retain only a small amount of grid-based relaxation (Fellhauer *et al.*, 2000).

The adopted implementation incorporates some differences to standard PM-codes. State-of-the-art PM codes use a cloud-in-cell (CIC) scheme to assign the masses of the particles to the grid cells. Therefore the mass of a particle i is split up into neighbouring cells according to its distance to the centre of the cell. Forces are then calculated by adding up the same fractions of the forces from all cells to particle i. In contrast, SUPERBOX uses the "old-fashioned" nearest-grid-point scheme, where the total mass of the particle is assigned to the grid cell the particle is located in. Forces acting on the particle are then calculated only from the forces acting on this particular cell. To achieve similar precision as CIC, SUPERBOX uses space derivatives up to the second order to compute the forces.

To achieve high resolution at the places of interest, SUPERBOX incorporates for every simulated object (e.g. each galaxy and/or star cluster or disc, bulge and halo) two levels of sub-grids co-moving with the objects of interest while the latter are moving through the

simulated area (see fig. 5.2). This provides higher resolution only where it is necessary.

5.2.4 The theory of the Coulomb logarithm

Dynamical friction affects a mass moving in a background sea of lower mass objects. A practical expression for the strength of the drag force on a point particle with mass M_{BH} is (Binney & Tremaine 1987, p. 424):

$$\frac{d\mathbf{v}_{BH}}{dt} = -4\pi G^2 \ln\Lambda \rho M_{BH} \frac{\mathbf{v}_{BH}}{v_{BH}^3} \left[\mathrm{erf}(X) - \frac{2X}{\sqrt{\pi}} e^{-X^2}\right]. \tag{5.1}$$

Here $X = v_{BH}/(\sqrt{2}\sigma)$, where σ is the Maxwellian velocity dispersion, and ρ the background stellar density.

The classical value of Λ is (Binney & Tremaine 1987, p. 423)

$$\Lambda = \frac{b_{max} v_{typ}^2}{G(M_{BH} + m)} \quad . \tag{5.2}$$

Here b_{max} is the largest possible impact parameter for an encounter between the massive point particle and a member of the background population, v_{typ} is the typical speed of the objects in the background population, and m is the mass of each of the background stars. Eq. (5.2) can then be generalised to

$$\Lambda = \frac{b_{max}}{b_{min}} \quad . \tag{5.3}$$

Here b_{min} is the distance below which an encountering particle is captured, instead of being scattered by the massive object. It is somewhat smaller than the 90° turn-around distance. With the direct N-body technique, Λ can be measured precisely. However, with approximate N-body methods, such as the treecode or the PM code, we have to take care of the interference of the softening length/cell size with b_{min}, as discussed in section 5.2.5.

McMillan & Portegies Zwart (2003) obtained an analytic expression for the distance $r(t)$ of the BH to the Galactic centre, with the assumptions that the BH's orbits are nearly circular, and the mass profile of the Galaxy is given by a power law:

$$M(R) = AR^{\alpha} \, . \tag{5.4}$$

They obtained:

$$r(t) = R_0 \left[1 - \frac{\alpha(\alpha+3)}{\alpha+1} \sqrt{\frac{G}{AR_0^{\alpha+3}}} \, \chi \, M_{BH} \, \ln\Lambda \, t\right]^{\frac{2}{3+\alpha}} \quad , \tag{5.5}$$

where

$$\chi = \mathrm{erf}(X) - \frac{2X}{\sqrt{\pi}} e^{-X^2} \quad \text{and} \quad X = \frac{v_{BH}}{\sqrt{2}\,\sigma} \, ,$$

σ being the velocity dispersion. In McMillan & Portegies Zwart (2003) the value of X in the Galactic centre is also computed, resulting in $X = \sqrt{2-\alpha}$. Finally, we take R_0 equal to the half-mass radius of our system R_{hm} (see section 5.2.6). The best fit of eq. (5.5) on the simulation data gives the value of $\ln\Lambda$ for that simulation. The values obtained, for all simulation performed, are reported in the last column of tables 5.3, 5.4 and 5.5.

5.2.5 The role of softening in the determination of the Coulomb logarithm

Softening was introduced in numerical stellar dynamics to limit the strength of mutual forces during close stellar encounters, mainly for computational performance purposes. It consists in a modification of the Newton law for the gravity exerted by a particle j on a particle i, as follows:

$$\mathbf{a}_i = G\frac{m_j}{(r_{ij}^2+\epsilon^2)^{(3/2)}}\mathbf{r}_{ij}\ , \tag{5.6}$$

where $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$, and ϵ is the softening parameter. As $\mathbf{r}_{ij} \to 0$, the presence of ϵ causes the force to change from inverse square to elastic, with constant Gm_im_j/ϵ^3. In this way the strength of the mutual force between encountering particles is limited.

Softening was first used by Aarseth (1963) in a particle-particle (PP) context (see fig. 1.1 and caption therein). Accuracy requirements soon led to a more precise treatment of close encounters and binaries by means of an analytic approach (Kustaanheimo & Stiefel 1965; Aarseth 1972; Mikkola & Aarseth 1990). The softened force in eq. (5.6) is used in the treecode, where high accuracy in close encounters treatment is not essential. Here we will use the softening both in the treecode simulations, where it is necessary, and in the PP code simulations, where it is used to compare the results of the two codes, in order to study the relation between ϵ and $\ln\Lambda$.

For the PM code, as described in section 5.2.3, force is not computed by using the Newton force, or the softened force in eq. (5.6). Instead, the fact that the gravitational potential on each grid point of the mesh is obtained from a density field defined on the same mesh, leads to an accuracy for the force on each particle limited by the cell size of the grid, l.

Here, we are concerned with the accuracy of the computation of the encounters experienced by a black hole spiralling-in to the Galactic centre. Since the softening (PP and treecode) and the cell size (PM code) affect this accuracy, we will use ϵ and l to quantify the accuracy decrement in our simulations. In section 5.3.5 we will study quantitatively the dependence of $\ln\Lambda$ on ϵ and l.

The reference value for ϵ in the work presented here will be $\epsilon_0 = 0.003\,735$ (units given below in table 5.1). This value, according to Athanassoula *et al.* (2000), is of the same order of magnitude as the optimal softening for a Dehnen sphere distribution (Dehnen, 1993). This distribution is similar to the power law distribution that we use, at least for what concerns the high central density peak, which is the key physical factor in the determination of the optimal softening. For an 80 000 particle distribution, ϵ_0 is about 15 times smaller than

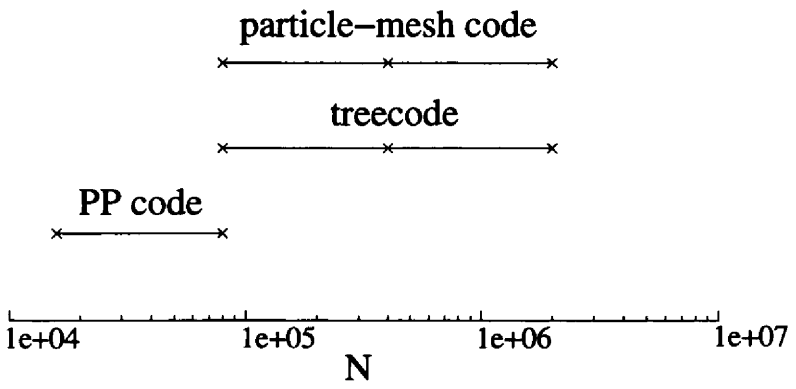


Figure 5.3: Particle ranges for the simulations performed by each method. Crosses denote the particle values used.

the mean inter-particle distance ℓ at the initial BH position $R_0 \simeq 0.87$ (see section 5.2.6). This value for ϵ is small enough to avoid spurious effects in the force between a star and its neighbours, but is sufficient to inhibit very close encounters. The expression for ℓ can be obtained as:

$$\ell = n^{-\frac{1}{3}} = \left(\frac{\rho}{m}\right)^{-\frac{1}{3}} = \sqrt[3]{\frac{4\pi R^{3-\alpha}}{NA\alpha}}\,, \tag{5.7}$$

where n is the star number density, and

$$\rho = \frac{1}{4\pi R^2}\frac{dM}{dR} = \frac{A\alpha}{4\pi}R^{\alpha-3}\,.$$

We used the expression in eq. (5.4) for M, and the fact that the N stars in the system have the same mass $m = 1/N$.

One of the effects of softening is a damping in the BH infall at very small values of the galactocentric distance, more noticeable as N increases. This can be explained with the fact that the inter-particle distance ℓ decreases as the BH approaches the Galactic centre (see eq. 5.7). When ℓ becomes comparable to 2ϵ, the role of softening in the force equation becomes dominant, since particles begin to "overlap". With $N = 400\,000$, we get $\ell = 2\epsilon_0$ when $R \simeq 0.064$, which is close to the value at which the damping arises, as fig. 5.13 below clearly shows.

5.2.6 Initial condition

We generate the initial mass distribution according to the power law given by eq. (5.4), with $\alpha = 1.2$, which reproduces the mass distribution in the centre of the Galaxy, according to Mezger *et al.* (1999). The scale factor is $A = 4.25 \cdot 10^6 \mathrm{M}_\odot$, corresponding to 0.44 in the N-body standard units (Heggie & Mathieu, 1985), which are reported in table 5.1. We use the standard units hereafter, unless other units are explicitly reported. The distributions

$$
\begin{aligned}
G &= 1\ [\mathrm{V}]^2[\mathrm{L}]/[\mathrm{M}] \\
&\equiv 4.3007 \cdot 10^{-3}\ \mathrm{km^2 pc/s^2 M_\odot} \\
&\equiv 4.49842 \cdot 10^{-3}\ \mathrm{pc^3/Myr^2 M_\odot} \\
1\ [\mathrm{L}] &= 8\ \mathrm{pc} \\
1\ [\mathrm{M}] &= 1.18 \cdot 10^8\ \mathrm{M_\odot} \\
1\ [\mathrm{V}] &= \sqrt{\frac{G \cdot 1\ [\mathrm{M}]}{1\ [\mathrm{L}]}} = 251.86\ \mathrm{km/s} \\
1\ [\mathrm{T}] &= \sqrt{\frac{1\ [\mathrm{L}]^3}{G \cdot 1\ [\mathrm{M}]}} = 0.031\ \mathrm{Myr}
\end{aligned}
$$

Table 5.1: Conversion table between the N-body units used in our work, and physical units. Here [L], [M], [T], and [V] are respectively the length, mass, time, and velocity units. The N-body units are such that $G = 1$, $M_{tot} = 1$, and $E_{tot} = -0.25$.

that we generate are truncated at $R = 1.7 = 13.6\,\mathrm{pc}$, with a total mass within this radius $M_{tot} = 1$. The particles have equal mass m. Particles are assigned Maxwellian velocities, then the system is virialised to dynamical equilibrium. Then, before inserting the black hole (BH) particle, we let the system evolve for a few crossing times. The system reaches a stable configuration, whose mass profile is no more perfectly reproduced by Eq. 5.4. The best fit for A and α on the mass profile of the stable configuration gives:

$$
\begin{aligned}
A &= 0.53\,, \\
\alpha &= 0.9\,.
\end{aligned}
\tag{5.8}
$$

In fact, the mass profile having these coefficients diverges from the original one as the distance R increases. On the other hand, in the region $R < 2$, where we study the BH infall, the discrepancy between the two mass profiles is small. The relaxed profile values are within 10% of the initial profile values. Nevertheless, for consistency we will use the values in eq. (5.8) for A and α hereafter. This results in values of $\ln\Lambda \simeq 10\%$ smaller than the ones given by a mass profile with coefficients $\alpha = 1.2$ and $A = 0.44$.

The BH particle is placed at the half-mass radius $R_{hm} \simeq 0.87$ with a circular orbit velocity, and its mass is $M_{BH} = 0.000\,528$. The background particles number varies from 16 000 to 2 million. The low particle number simulations are performed with the PP code, the intermediate and high number simulations with the treecode and the PM code. Fig. 5.3 shows the range of N for each code. This allows us to span a large range in particle number, so that the influence of granularity in the BH motion towards the Galaxy centre can be studied.

In contrast to the other models, we choose physical units for the PM code simulations. The conversion factors from physical units to N-body units are shown in table 5.1, where

n	outer	middle	inner
32	10.00	0.69	0.17
64	4.67	0.32	0.08
128	2.26	0.15	0.04

Table 5.2: Resolutions (i.e. cell sizes) of the different grid levels for the different choices of n in the PM code. n denotes the number of cells per dimension. The cell sizes of the different grid-levels (outer, middle and inner) are given in pc.

[L] denotes the unit length in N-body units, [M] the unit mass, [V] the unit velocity and [T] the unit time.

The parameters of the PM calculations are chosen in the following way: the grid sizes are kept constant at

$$\begin{aligned} R_{\text{system}} &= 140.0 \text{ pc} \\ R_{\text{out}} &= 9.6 \text{ pc} \\ R_{\text{core}} &= 2.4 \text{ pc} \end{aligned} \tag{5.9}$$

and are focussed on the center of mass of the "bulge" model, as sketched in fig. 5.2. To change the resolution we alter the number of grid cells per dimension from 32 up to 128. With this choice the cell sizes listed in table 5.2 are achieved.

To speed up the simulations, the time step in the PM code simulations should be as large as possible, but small enough to prevent spurious results. Therefore we started with a time step of 1 000 yr and reduced it to 200 and 50 yr. The results of the 200 yr and 50 yr time step do not differ from each other, therefore the global time step is chosen to be 200 yr. Conversely, the time step in the treecode and direct code simulations is variable and different for each particle. Time step values are in this case in the range 2–30 000 yr, with about 90% of them in the range 100–300 yr.

5.3 Results

We will now study the dependence of our results on the number of particles N in section 5.3.2, and compare the various N-body methods with identical initial realisations in section 5.3.3. After having convinced ourselves that the various techniques produce consistent results, we continue by studying the effect of softening/cell size (section 5.3.4) and black hole mass (section 5.3.6) on the value of the Coulomb logarithm in the inner part of the Galaxy.

Our simulations aimed at several goals. 1) understanding the scaling of the system dynamics with respect to the number of particles N, and within this scaling, how results from different methods compare with each other. 2) How, at a fixed value of N, the softening parameter influences the dynamics, changing the value of $\ln \Lambda$. The particle-mesh method does not make use of softening. The cell size in the PM code can be seen in this context as a

N	ϵ/ϵ_0	M_{BH}/m	ϵ/b_{min}	$\ln\Lambda$
16K	0	8.5	0	3.8
16K	1	8.5	2.6	3.6
80K	0	42.3	0	6.6
80K	0.01	42.3	0.03	6.0
80K	0.1	42.3	0.3	5.3
80K	1	42.3	2.6	4.8
80K	2	42.3	5.3	3.5
80K	8	42.3	21.2	2.8
80K	16	42.3	42.4	1.8

Table 5.3: Overview of the PP runs. N is the number of particles, ϵ is the softening parameter, $\epsilon_0 = 0.003\,735$, M_{BH}/m is the ratio between the BH mass and a particle mass, and ϵ/b_{min} the ratio between the softening parameter and the minimal impact parameter.

N	ϵ/ϵ_0	M_{BH}/m	ϵ/b_{min}	$\ln\Lambda$
80K	1	42.3	2.6	4.7
400K	1	211.3	2.6	5.0
2M	1	1056.5	2.6	4.9
80K	0.1	42.3	0.3	5.7
80K	2	42.3	5.3	4.1
80K	8	42.3	21.2	3.0
80K	16	42.3	42.4	2.0
80K	32	42.3	84.7	1.6
80K	1	84.5	1.3	5.4
80K	1	169.0	0.7	4.6
400K	1	422.6	1.3	4.6
400K	1	845.2	0.7	4.2

Table 5.4: Overview of the treecode runs. Meaning of symbols is the same as in table 5.3 above.

softening length. In our framework, it is crucial to understand the relation between the PP code and treecode softening parameter and the PM code cell size. 3) We also study how the BH mass influences the infall time. We doubled and quadrupled the BH mass, and observed how this affects the value of $\ln\Lambda$.

A resume of all the runs that we performed is reported in table 5.3 for the PP code runs, table 5.4 for the treecode runs, and finally table 5.5 for the PM code runs. In all of our runs, the system remains in equilibrium during the whole BH infall, with no significant mass loss from stellar escapes, and a mass profile independent of time.

N	n	m [$M_\odot$]	l [pc]	N_c [$\frac{\#}{\text{cell}}$]	m_c [$\frac{M_\odot}{\text{cell}}$]	$\frac{M_{BH}}{m_c}$	$\frac{l}{b_{min}}$	$\ln\Lambda$
80K	16	1475	1.60	46.3	68287.0	0.9	114.3	n/a
80K	32	1475	0.69	3.6	5375.4	11.6	49.3	1.9
400K		295	0.69	18.2	5375.4	11.6	49.3	2.1
2M		59	0.69	91.1	5375.4	11.6	49.3	2.2
80K	64	1475	0.32	0.4	546.3	114.5	22.9	3.0
400K		295	0.32	1.9	546.3	114.5	22.9	3.4
2M		59	0.32	9.3	546.3	114.5	22.9	3.0
80K	128	1475	0.15	0.04	61.9	1011	10.7	2.8
400K		295	0.15	0.2	61.9	1011	10.7	3.7
2M		59	0.15	1.0	61.9	1011	10.7	3.8
2M	256	59	0.076	0.1	7.4	8483	5.4	4.1

Table 5.5: Overview of the PM runs. N is the number of particles, n the number of grid cells per dimension, m the particle mass, l the intermediate grid cell size, N_c the average number of particles per cell, m_c the average mass of a cell, M_{BH}/m_c the ratio between the BH mass and the cell mass, and finally l/b_{min} the ratio between the cell size and the minimal impact parameter.

Before we start with the analysis of the results of our simulations, we report on the performance of the PP code and the treecode runs.

5.3.1 Code performance

In table 5.6 we give the average time, in seconds, needed to evolve the system for one N-body time unit (N-body time units are given in table 5.1). We report the data concerning the runs with $N = 80\,000$ and $M_{BH} = 0.000\,528$, for both the PP and the treecode runs. The PP runs have been executed on a partition of the GRAPE-6 (see section 1.3.2) including four GRAPE boards, for a peak-performance of about four TFlop/s. The treecode runs have been executed on the DAS-2 (mentioned in section 2.2.1), using a varying number of nodes, as reported in table 5.6. This varying number of PEs obviously affects the performance figures of the treecode runs; in order to obtain an homogeneous set of data, we normalised the figures to 32 PEs assuming a linear scaling, i.e. we halved the timing values measured on 16 PEs, and doubled the values measured on 64 PEs. The peak performance of the normalised system is 32 GFlop/s.

The normalised data are plotted in fig. 5.4, together with the PP code values (note the shift in the X-axis, in order to show the value for $\epsilon = 0$ on a log-log plot). We can see from the figure that the normalised treecode data are not heavily influenced by ϵ, while the PP code runs are much faster as ϵ increases. A possible explanation for this is that, as ϵ gets bigger, the chance for a close encounter gets smaller. Since the role of ϵ is to reduce the strength of the gravitational interaction at low interparticle distance to prevent close

PP code		treecode		
ϵ/ϵ_0	s/[T]	ϵ/ϵ_0	s/[T]	PEs
0	1400 ± 400	0.1	183 ± 11	32
0.1	1300 ± 200	1	529 ± 84	16
1	1038 ± 19	2	126.3 ± 5.2	64
2	785 ± 15	8	349 ± 38	16
8	485.3 ± 7.9	16	213.8 ± 4.3	32
16	440.5 ± 4.8	32	223 ± 12	32

Table 5.6: Performance of the PP and the treecode runs. We report the averaged number of seconds needed to advance the system for one N-body unit. For all runs we have $N = 80\,000$ and $M_{BH} = 0.000\,528$. The reference value for the accuracy parameter is $\epsilon_0 = 0.003\,735$.

encounters, a larger ϵ implies a lower chance for close encounters to occur. The time advance of the PP code is heavily affected by close encounters, as its high numerical precision can only be assured by a detailed, and costly, treatment of particle trajectories during close encounters. Hence, a reduced frequency of close encounters speeds up the execution of the PP code. The treecode does not include a special treatment for close encounters, hence its execution speed is not affected by a change in the close encounters frequency.

Fig. 5.4 also shows that the runs with the treecode are faster than those with the PP code, especially for low ϵ values. This effect is even much larger if we take into account that the PP runs have been performed on a four TFlop/s system, while the normalised treecode runs have been performed on a 32 GFlop/s system. Normalising the PP code runs on this performance would result in values 125 times slower. This is again the price of the high numerical precision of the PP code. In return for this, the energy conservation of the PP code is in the order of 10^{-6}, while the treecode conserves the energy within about 1%.

5.3.2 Dependence of $\ln \Lambda$ on N

In order to obtain a precise measure of $\ln \Lambda$, ideally one would run a direct N-body simulation with N of the order of the number of stars in the Galactic bulge, which amounts to $\sim 10^8$. Such high number makes a direct simulation unfeasible, and imposes the use of approximate methods instead. In order to evaluate the reliability of the approximate methods, we compared the PP code runs with the treecode runs. The PP code runs give a reliable picture of the system dynamics at low particle numbers, i.e. at high granularity. Using the treecode we can reach a much higher number of particles, up to two million, which still is two orders of magnitude lower than the real system. A comparison of the results from the two methods allows us to estimate the validity of the treecode runs, up to 2 million particles. Then we can compare the treecode runs and the PM runs, in order to validate the results from the latter, which has the capability to simulate systems of about 100 million stars. In this way we will eventually be able to study the infall of a BH into the Galactic centre in a simulation

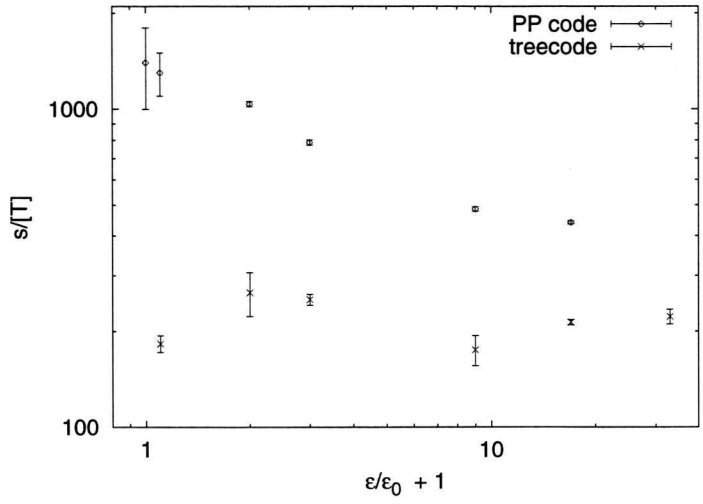


Figure 5.4: Performance of the PP and the treecode runs. We plot here the averaged number of seconds needed to advance the system for one N-body unit. For all runs we have $N = 80\,000$ and $M_{BH} = 0.000\,528$. The PP code runs are executed on a GRAPE-6 partition including four GRAPE boards, the treecode runs are executed on the DAS-2, with varying number of nodes. The values plotted here are normalised to 32 PEs. Note the shift in the X-axis, where we plot $\epsilon/\epsilon_0 + 1$.

environment with a realistic value of N.

In fig. 5.5 we show the evolution of the BH distance from the centre of mass of the system for three treecode simulations. N varies from 80 000 to 400 000 and 2 million, with $\epsilon = \epsilon_0 = 0.003\,735$, corresponding to about 0.03 pc. In fig. 5.6 we present a similar figure from PM code simulations. Here is $N \in \{80\,000, 400\,000, 2\,000\,000\}$, with 32 cells per dimension, resulting in a cell size of about 0.69 pc.

Fig. 5.5 and 5.6 show that increasing N results in a much smoother motion of the BH in its infall towards the centre of the Galaxy. The BH infall rate (though very different in the two cases) is not much affected by a change in N. Accordingly, the value of $\ln\Lambda$ for each of the two sets above is consistent, as values in table 5.4 (first three rows) and table 5.5 (rows with $l = 0.69$) show.

In order to study further the extent of the influence of N on the infall rate of the BH, and hence in $\ln\Lambda$, we compare in fig. 5.7 results from PM code simulations with increasing grid refinement, and extreme difference in N. To quantify the grid resolution, we use the cell length at intermediate refinement, which is the cell length pertaining to the physical region where the BH evolves for most of its infall. We measure this length in units of $\epsilon_0 = 0.003\,735$, which makes the comparison with the softening parameter of the treecode easier. N has no strong influence on the infall rate, except for the case where the cell size is $l = 0.15$ pc $\simeq 5\epsilon_0$. In this case the simulation with $N = 80\,000$ (data not reported in the figure), shows an incorrect BH infall, comparable to the case $l = 0.32 \simeq 10\epsilon_0$. This can be explained by the

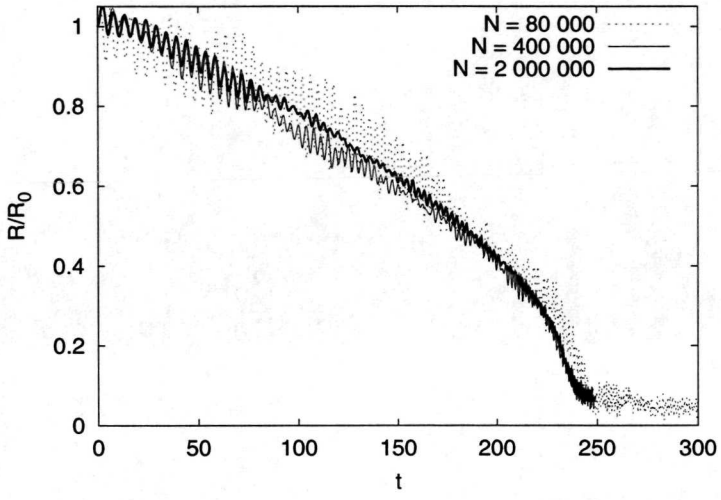


Figure 5.5: Time evolution of the radial distance of the black hole to the Galactic centre. The various curves (identified in the top right corner) present results obtained with the treecode. the X-axis is presented in N-body time units: one N-body time unit corresponds to about 0.031 Myr. The distance of the black hole to the Galactic centre (Y-axis) is given in terms of its initial distance. In these simulations is $\epsilon = 0.003\,735 \simeq 0.03$ pc and $M_{BH} = 0.000\,528$.

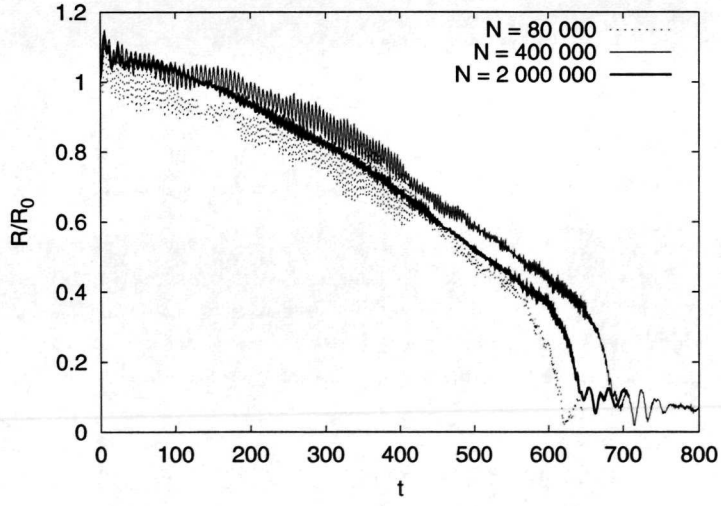


Figure 5.6: Same as fig. 5.5 above, but for PM code simulations. The intermediate grid cell size is here $l = 0.69$ pc, and $M_{BH} = 0.000\,528$.

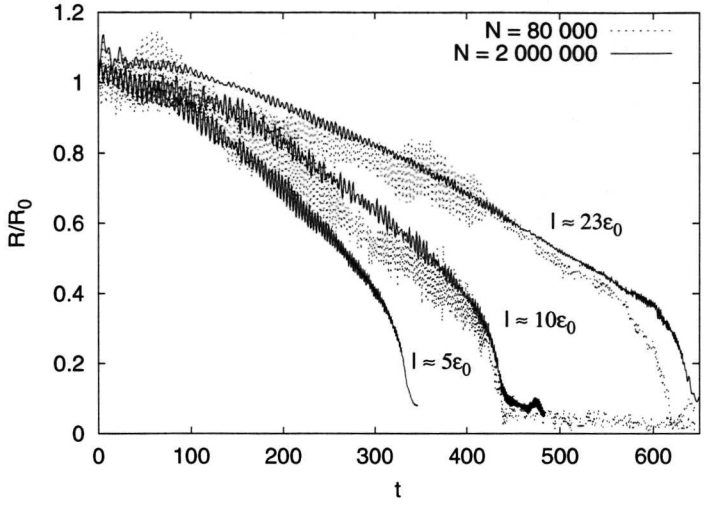


Figure 5.7: Black hole infall at various cell sizes, and large difference in N. Results here are from PM code simulations. The case $N = 80\,000$, $l \simeq 5\epsilon_0$ is not shown for readability reasons, since it would overlap with the $l \simeq 10\epsilon_0$ results.

fact that in the low l, low N case, the cells are so small, and the particles so few, that many cells in the PM grid are empty (see also the N_c column in table 5.5, which gives the average number of particles per cell). When $N_c \ll 1$, the density field is incorrect, with many grid points having a null value, because the corresponding cell is empty. In this condition, the gravity field computed by the PM code becomes unreliable, affecting the numerical results, as in the simulation with $N = 80\,000$ and $l \simeq 5\epsilon_0$.

5.3.3 Comparison of the codes

In this section we compare the results obtained from the various codes, to check their consistency. The comparison of the PM results with the two other codes results is particularly critical, since the PM code computes forces using a different mathematical approach, i.e. a grid based force derivation vs a direct particle-particle computation for the PP code, or particle-multipole computation for the treecode. A consequence of this is a different parameter to tune the accuracy of the simulation, namely the cell size l for the PM code, and the softening length ϵ for the other two codes. We will study here how these two parameters influence the black hole infall.

In fig. 5.8 we show the time evolution of the galactocentric BH distance R simulated by the PP code, accompanied by a plot of the time evolution of $\Delta R/R_{PP}$ for treecode and PM simulations, where $\Delta R = (R - R_{PP})$. The relative difference $\Delta R/R_{PP}$ remains small for a large fraction of the infall, and the final discrepancy is mostly due to the small values of the quantities at that point, which are likely to amplify relative differences. As the following

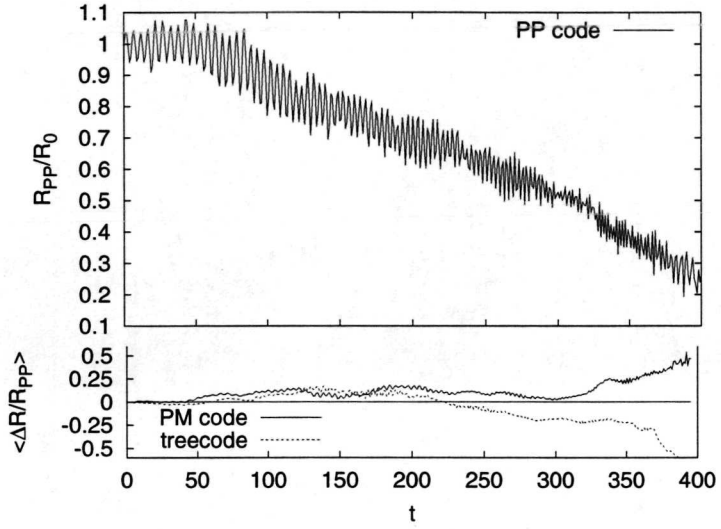


Figure 5.8: Top panel shows a black hole infall simulated by the PP code, with $N = 80\,000$, $M_{BH} = 0.000\,528$ and $\epsilon = 8\epsilon_0$. Bottom panel shows a comparison of the PP results with treecode and PM results. Parameter values are in all cases the same, except for the PM cell size, which is $l = 10\epsilon_0$. Plotted values are averages over 10 time units.

figures also show, the BH infall is predicted with very good consistency between the codes.

In fig. 5.9 selected treecode runs with $N = 80\,000$ and increasing ϵ are compared with the direct code runs having the same values of N and ϵ. At the same time, the figure shows how the infall time increases (and implicitly how $\ln\Lambda$ decreases), as ϵ increases. Fig. 5.9 and table 5.7 show that the results from the treecode, the PP code, and the PM code are in good agreement. The agreement of the results from the three methods, and the scaling of $\ln\Lambda$ with ϵ, will be further studied quantitatively in section 5.3.5.

In order to understand how the cell length l of the PM code and the softening parameter ϵ of the PP code and treecode relate with each other, we compare in fig. 5.10 the results from the PM code and treecode simulations with 80 000 particles. The BH infall as shown in fig. 5.10 depends on the value of l or ϵ. Remarkably, l and ϵ seem to play the same role not only qualitatively, but also quantitatively: in a PM run, a given value of l induces an infall which is very similar to the infall, in a treecode run, with ϵ assuming that same value. In section 5.3.5 this relation will be studied further.

5.3.4 The effect of softening/grid

The influence of the softening parameter on the BH dynamics has been studied by running a number of simulations with the three codes. In table 5.7 we report the value of $\ln\Lambda$ obtained from our simulations. For the PP code and treecode simulations, we increase ϵ from 0 to $32\epsilon_0 = 0.1195 \simeq 0.96\,\mathrm{pc}$. For the PM code, we increase l from $2.5\,\epsilon_0$ to $23\,\epsilon_0$. In all cases is

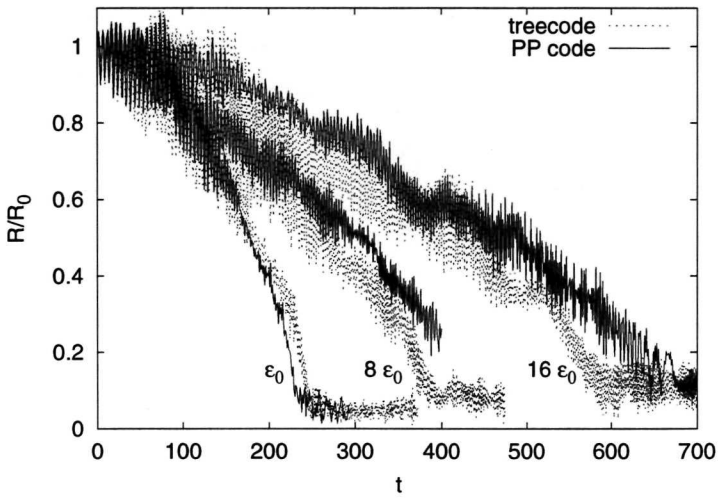


Figure 5.9: Comparison of results from the PP code with results from the treecode, at different values of ϵ. For all cases shown here is $N = 80\,000$ and $M_{BH} = 0.000\,528$. The PP simulation with $\epsilon = 8\epsilon_0$ has been already shown in fig. 5.8.

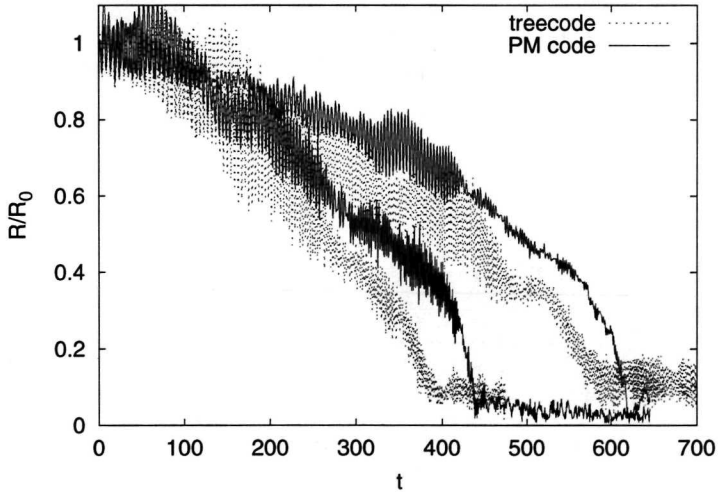


Figure 5.10: Comparison of PM results with treecode results. PM simulations have cell size l equal to resp. $10\epsilon_0$ and $23\epsilon_0$; softening parameters in the treecode runs are resp. $8\epsilon_0$ and $16\epsilon_0$. In all the above cases, is $N = 80\,000$ and $M_{BH} = 0.000\,528$.

ϵ/ϵ_0	PP code	treecode	l/ϵ_0	PM code
0	6.6			
0.01	6.0			
0.1	5.3	5.7		
1	4.8	4.7		
2	3.5	4.1	2.5	4.1
8	2.8	3.0	5	3.8
16	1.8	2.0	10	3.0
32		1.6	23	2.2

Table 5.7: $\ln\Lambda$ versus ϵ from PP code, treecode, and PM code runs. For the PP code and treecode runs is $N = 80\,000$. For the PM code runs is $N = 2$ million. The reference value for the accuracy parameter is $\epsilon_0 = 0.003\,735$.

$M_{BH} = 0.528 \cdot 10^{-3} \simeq 62\,300\,\mathrm{M}_\odot$.

For the PP code and the treecode, we selected $N = 80\,000$ as a suitable value. The relaxation time eq. (1.2) is for this value of $N\,t_r \simeq 0.1N/\ln N \cdot R^{3/2} \simeq 2000$, about one order of magnitude larger than the typical BH infall time, so that the system is collisionless, and we can confidently use the treecode to simulate it. With this choice for N, the BH mass is $M_{BH}/m \simeq 42.3$, (see table 5.4). As a cross-check, we ran two PP runs with $N = 16\,000$ which, as expected, gave incorrect results (see table 5.3). This is due to both a too small M_{BH}/m ratio, and a too short relaxation time ($t_r \simeq 400$ in this case). We did not increase ϵ above $32\,\epsilon_0$, since at this point ϵ is already much bigger than b_{min} (see table 5.4), and the infall time is now close to t_r.

For the PM code simulations, we used $N = 2$ million in order to have enough particles to fill all the cells, even for the simulations with a small l. As table 5.5 shows, for $l = 0.076\,\mathrm{pc} \simeq 2.5\,\epsilon_0$ the average number of particles per cell is already $N_c = 0.1$. Since a PM simulation gives incorrect results for $N_c \ll 1$ (see also the discussion at the end of section 5.3.2), we did not decrease l below $2.5\,\epsilon_0$.

The decrease of the value of $\ln\Lambda$ as ϵ or l increases is clear from table 5.7. In the next section we focus on the relation between Λ and ϵ, and provide a fitting formula for $\ln\Lambda(\epsilon)$. We use hereafter ϵ to refer either to the softening length of the PP and treecode, or the cell size of the PM code. As shown on fig. 5.10 and discussed above, these two parameters play the same role even quantitatively in affecting $\ln\Lambda$. In this respect, we refer to ϵ as a generic accuracy parameter.

5.3.5 Determination of $\ln\Lambda$

We will study here the relation between ϵ and $\ln\Lambda$. As just said above, in this context ϵ will be used as the accuracy parameter, and it will refer to either the softening length used in the PP and treecode, or to the cell size in the PM code.

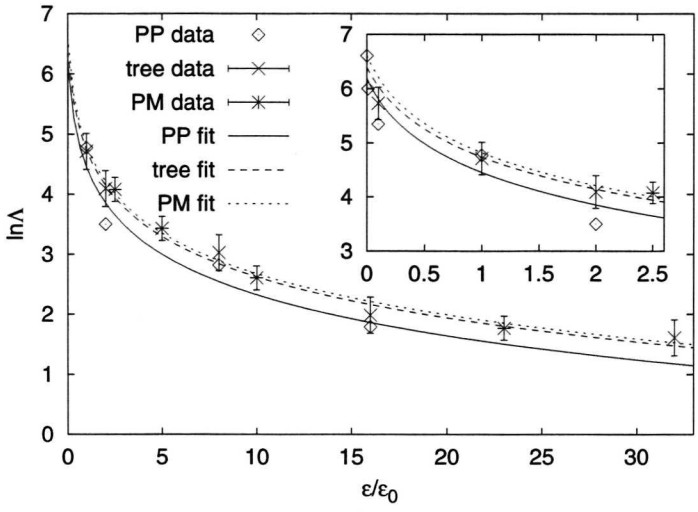


Figure 5.11: $\ln\Lambda$ vs ϵ, and best fit for $\ln\Lambda = K - \ln(a+\epsilon)$. Values for K and a are given in table 5.8. The inset in the figure is a magnification of the low ϵ region. In all cases is $M_{BH} = 0.000\,528$. For the PP and treecode runs is $N = 80\,000$, for the PM code runs is $N = 2\,000\,000$. Error bars are omitted from the PP values to improve readability. For the same reason, $\ln\Lambda$ values for $\epsilon/\epsilon_0 < 1$ are shown only in the inset.

A mathematical expression for the relation between ϵ and $\ln\Lambda$ can be found by considering how softening affects two body scattering. The role of ϵ is to prevent too close stellar encounters. In this respect, the effect of introducing a softening length is to increase the minimal impact parameter. Hence, we can define an effective impact parameter $b_{eff} = b_{min} + \epsilon$, and we modify eq. (5.3) to become:

$$\ln\Lambda = \ln\frac{b_{max}}{b_{eff}} = \ln\frac{b_{max}}{b_{min}+\epsilon}\,. \tag{5.10}$$

We will now fit this equation with the values reported in table 5.7. In order to perform the fit, we change eq. (5.10) in a more suitable form, as follows:

$$\ln\Lambda = \ln b_{max} - \ln(b_{min}+\epsilon) = K - \ln(a+\epsilon)\,. \tag{5.11}$$

We will refer to b_{max} and b_{min} as the theoretical values of the maximal and minimal impact parameters, as they can be obtained from eq.s (5.2) and (5.3), and K and a as the corresponding experimental values obtained with the fit.

The best fits for K and a with respect to simulation values are reported in table 5.8 for all codes. Such fits have been performed with a fixed value for R_0, i.e. the $R_0 = R_{hm}$. In fact, the not perfectly circular orbit of the BH results in an oscillatory behaviour for the BH

	PP code	treecode	PM code
K	-0.94 $\pm$ 0.21	-0.64 $\pm$ 0.10	-0.59 $\pm$ 0.05
$a \cdot 10^{-3}$	0.80 $\pm$ 0.28	0.88 $\pm$ 0.20	0.74 $\pm$ 0.08
$\Delta(\ln \Lambda)$	0.6	0.3	0.2

Table 5.8: Best values for the parameters K and a, and error on $\ln \Lambda$ for the fit of $\ln \Lambda = K - \ln(a + \epsilon)$.

galactocentric radius. In this case, having R_0 fixed could not be an appropriate choice for the fit. We checked whether having R_0 as a free parameter in the fit leads to different results in $\ln \Lambda$. We obtained values for $\ln \Lambda$ within the error bars in fig. 5.11, and values for R_0 within $R_{hm} \pm 0.05$. We can conclude that, although the galactocentric BH radius does not decrease smoothly, but in an oscillatory fashion, having R_0 fixed to the actual initial BH radius in the simulations leads to correct fits for the value of $\ln \Lambda$. With respect to the PM values, a further peculiarity is that when the BH enters the finest grid area, i.e. approximately at $R = 0.3$, the value of l decreases (see section 5.2.3 and fig. 5.2). This causes b_{eff} to become smaller, increasing the value of $\ln \Lambda$. In fact, a fit of the PM data limited to values of $R > 0.3$ gives values of $\ln \Lambda$ systematically higher by $\simeq 0.3 \simeq 2\Delta(\ln \Lambda)$.

From the PP code value of K in table 5.8 we obtain for b_{max} the experimental value $b^E_{max} = e^K \simeq 0.39$. This value is much smaller than what one would expect. Since b_{max} has the meaning of the maximal impact parameter, a natural choice is to assign it a value of the order of the system size, which in our case would result in $b_{max} = 2$. The maximal radius for dynamical friction in our system is then about one quarter of what it is customarily assumed. PM and treecode values are slightly higher, but still much smaller than $b_{max} = 2$. Also a is smaller than the theoretical value $b_{min} = G \cdot (M_{BH} + m)/v^2_{typ} = 1.41 \cdot 10^{-3}$, by a factor 3. The a value for all codes is perfectly consistent.

An explanation for the discrepancy between the values of b_{max} and b^E_{max} is that the BH, while moving to the Galactic centre, is off-centre with respect to the density peak (in fact the BH is spiralising towards it). With respect to the BH position, the density distribution is then asymmetric. This density peak clearly has a greater influence on the BH dynamics, contributing more than the other regions of the system to the dynamical friction on the BH. This leads to a value of b_{max} affected by the galactocentric BH radius. This approach is studied in detail by Hashimoto *et al.* (2003), who propose the galactocentric radius as a value for b_{max} in the context of the spiral-in of satellite galaxies.

In our simulations, the galactocentric radius varies from $R \simeq 0.9$ at the beginning of a simulation, to $R \simeq 0$ at the end of it. The value of b^E_{max} that we find is within this range, and it can be interpreted as an order 0 estimate of a maximal impact parameter that depends on the galactocentric BH radius.

In order to explore this aspect further, we simulated the infall of the same BH, starting at the quarter mass radius $R_{qm} \simeq 0.43$, for ϵ ranging from 0 to $16\epsilon_0$. What we expect is a smaller value of b^E_{max}, hence smaller values of $\ln \Lambda$. All simulations are performed with the treecode, except for the $\epsilon = 0$ case, which is simulated with the PP code. Our results are

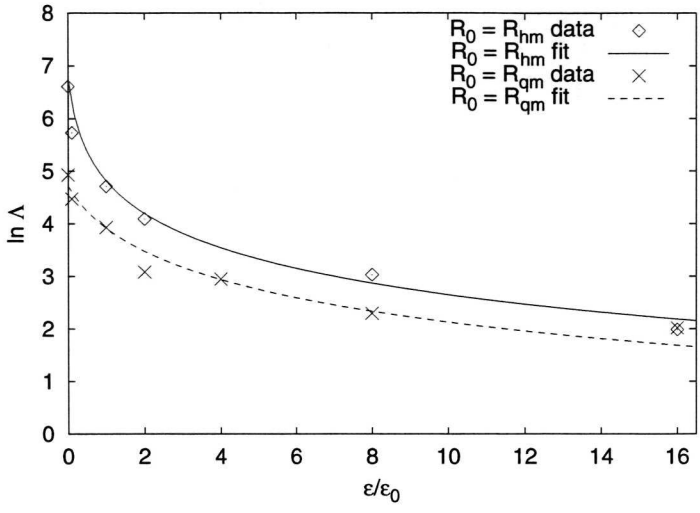


Figure 5.12: Comparison of $\ln\Lambda$ vs ϵ at different initial galactocentric BH radii. The smaller values of $\ln\Lambda$ for the cases with $R_0 = R_{qm}$ indicate that b_{max} is influenced by the galactocentric BH radius.

in fig. 5.12. We can see there that the values of $\ln\Lambda$ are smaller for the cases when the BH starts at the quarter mass radius. A fit on these data gives $K \simeq -1.1$, which implies $b^E_{max} \simeq 0.33$, which is smaller than the value of b^E_{max} obtained for the BH starting from the half mass radius. Our findings support the argument of Hashimoto *et al.* (2003).

5.3.6 Varying black hole mass

We also studied the effect of a variable BH mass on the value of $\ln\Lambda$. We simulated, using the treecode, the infall of a BH of mass two times and four times larger than the default mass $M_0 = 0.000\,528 \simeq 0.62 \cdot 10^5 \mathrm{M}_\odot$. We studied this infall in both the 80 000 particles configuration, and the 400 000 particles configuration. In all cases, we used our standard value for ϵ, i.e. $\epsilon_0 = 0.003\,735$. In fig. 5.13 the distance r of the BH from the centre of mass of the system is shown for all the cases mentioned above, together with the $M_{BH} = M_0$ cases. From eq. (5.11) and table 5.8, the appropriate value for $\ln\Lambda$ in the above cases is:

$$\begin{aligned} \ln\Lambda &= K - \ln(a + \epsilon_0) \pm \Delta = \\ &\quad -0.64 - \ln(0.000\,88 + 0.003\,735) \pm 0.3 \simeq 4.7 \pm 0.3 \,. \end{aligned}$$

We also show in fig. 5.13 the analytic curve $r(t)$, as given by eq. (5.5), with $\ln\Lambda = 4.7$. An error bar gives, for each analytical curve, the spread corresponding to a variance $\Delta(\ln\Lambda) = 0.3$.

The results shown in fig. 5.13 are consistent with the hypothesis that a variation in the BH mass has a little effect in the value of $\ln\Lambda$. In fact, $\ln\Lambda$ shows a logarithmic dependence on M_{BH} through the parameter b_{min}, which depends linearly on M_{BH} (see eq.s (5.2)

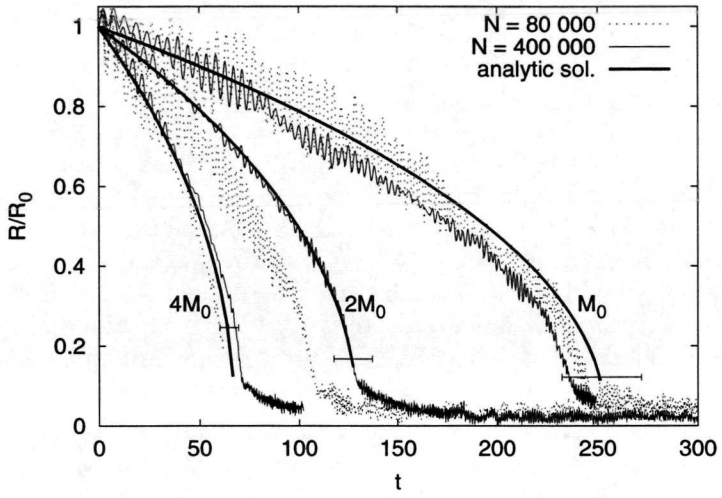


Figure 5.13: Black hole infall for different values of the BH mass, and different values of N. Simulations are performed with the treecode. Simulation results are compared with the analytic solution, eq. (5.5), with $\ln\Lambda$ obtained from eq. (5.11) and table 5.8. The error bars at the bottom of the analytic curves correspond to a variance $\Delta(\ln\Lambda) = \pm 0.3$.

and (5.3)). Assuming that also the experimental value a depends linearly on M_{BH}, we obtain $\ln\Lambda \simeq 4.6 \pm 0.3$, and $\ln\Lambda \simeq 4.4 \pm 0.3$ respectively for the 2 M_{BH} case and the 4 M_{BH} case. This results in a small displacement towards the right of the corresponding analytic curves in fig. 5.13, which does not affect the conclusions that can be drawn from the figure. The theoretical curve fits very well with the $M = 2\,M_0$, $N = 400\,000$ case. The other simulation curves are within, or very close to, the error in $r(t)$ associated to the error in $\ln\Lambda$. We can conclude that a variation in the mass of the infalling object has little influence in the value of $\ln\Lambda$, which is important in view of extending this work to the case of the infall of a star cluster.

The fitting formula for $\ln\Lambda$ vs ϵ was obtained from simulations with $M_{BH} = M_0$. This formula predicts $\ln\Lambda$ for the cases with $M_{BH} > M_0$ with a very good accuracy, showing that it can be applied in a more general context, in order to forecast the value of the Coulomb logarithm.

Fig. 5.13 also shows a damping in the BH infall at very small values of R, especially for the $N = 400\,000$ case. This effect, described in section 5.2.5, is clearer in the $N = 400\,000$ case, since the particle density is higher in this case, compared to the $N = 80\,000$ case.

5.3.7 Comparison with related work

Milosavljević & Merritt (2001) study the dynamical evolution of two black holes, each one

at the centre of a power law cusped galaxy core. They simulate the merging of the two galaxies, and observe the evolution of the two black holes, which form a hard binary at the centre of the merged galaxy. In section 3 of their paper they discuss the value of $\ln\Lambda$ in their simulations. They measure the decay rate of the two black holes, and compare this value with theoretical estimates. When they compare their experimental decay rate with an estimate for the case of the infall of an isolated black hole, they find a theoretical estimate about 6 times lower than the measured value, under the assumption that $\ln\Lambda \simeq 1.6$. If the value of $\ln\Lambda$ is not theoretically pre-determined, and is instead obtained from the decay rate equation, the result is $\ln\Lambda \simeq 10$. Similarly, they compare the experimental value with an estimate for the case of two mutually spiralling spherical distributions of matter. In this case they assume $\ln\Lambda \simeq 1.0$, and obtain an estimate for the decay rate about a factor of 2 lower than the observed value. Determining $\ln\Lambda$ from the measurement would give in this case $\ln\Lambda \simeq 1.87$. The values of $\ln\Lambda$ that we find are between the two values above.

The value for $\ln\Lambda \simeq 1$ that they assume in their theoretical estimates, comes from a derivation that they present in appendix A of the same work. This derivation is based on results of Maoz (1993). Under the assumption that the stellar density obeys a power law centered on the BH position:

$$\rho(r) = \rho_0 \left(\frac{r}{b_{min}}\right)^{-\alpha} , \tag{5.12}$$

they obtain $\Lambda \simeq 1/\alpha \simeq 1$, which actually implies $b_{max} \simeq b_{min}$, whereas it is customary to consider $b_{max} \gg b_{min}$.

Their assumption in fact is valid only when the BH is close to the centre of the power law distribution. In their context this is true when: 1) the separation between the two BHs is much larger than the half mass radius of the two galaxies. In this case each BH is at the centre of its own galaxy, and at the same time its motion is not yet heavily perturbed by the other galaxy. 2) the BH binary has hardened, and occupies the centre of the merged galaxy.

During the transient phase, when the two BHs have not yet formed a binary, the density distribution that affects the motion of the BHs is double-cusped, with a BH in each of the two cusps. This is substantially different from the density distribution modelled by eq. (5.12).

This qualitative argument would make the density distribution in eq. (5.12) inapplicable during the transient phase, and could explain why Milosavljević & Merritt (2001) find a higher than expected value of $\ln\Lambda$ in the transient. The analytical evaluation of $\ln\Lambda$ according to the technique used by them is by no means trivial, when symmetry arguments cannot be straightforwardly applied. We will address this issue in future developments of the present work; the theory of linear response of Colpi & Pallavicini (1998) could be very useful in this context.

5.4 Applications to star clusters

Recent observations of the Galactic Centre have revealed a population of very young clusters with ages less than 10 Myr. The presence of such stars inside the inner parsec of the Galaxy

is puzzling, as the strong tidal field in the Galactic centre easily prevents star formation. The origin of these stars is therefore debated (Gerhard, 2001; McMillan & Portegies Zwart, 2003). Morris (1993) proposed that a star cluster at some distance from the Galactic centre could spiral-in due to dynamical friction (see also Gerhard, 2001). The efficiency of dynamical friction depends sensitively on the actual value of the Coulomb logarithm $\ln\Lambda$.

5.4.1 Sinking of massive black holes in the Galactic centre

We performed N-body simulations for a large range of conditions. In section 5.3.2 we varied the number of particles, in section 5.3.4 we varied the size of the object, and in section 5.3.6 we varied its mass. With direct N-body simulations we measured the actual value of the Coulomb logarithm $\ln\Lambda$. We study the behaviour of $\ln\Lambda$ for various types of N-body solvers and particle numbers. We also study the behaviour of $\ln\Lambda$ as a function of the softening length ϵ. Only the direct N-body code can perform a true measurement of the Coulomb logarithm, because it is able to resolve even the smallest length scales and time scales. This, however, makes the direct code very slow and, even using the very fast GRAPE-6 special purpose device, we are able to perform simulations with only 10^5 particles. This is a small number compared to the actual number of stars in the Galactic centre. With approximate methods (treecode and particle-mesh) we are able to increase the number of particles up to 2 million. The cost of this is a lower accuracy in calculating stellar motion below a typical length scale ϵ. We studied how this length scale influences $\ln\Lambda$, by affecting the value of the minimal impact parameter.

5.4.2 Young dense clusters in the Galactic centre

The study of the dependence of $\ln\Lambda$ on ϵ described above is also of astronomical interest, because ϵ can be interpreted as the typical length of a finite size infalling object. Based on this, our analysis of the dependence of $\ln\Lambda$ on ϵ can be seen as a first approach to the study of the infall of a star cluster of typical size ϵ toward the Galactic centre. We found (see fig. 5.11) that the value of $\ln\Lambda$ decreases quite rapidly as ϵ increases, with the logarithm argument $\Lambda \propto 1/\epsilon$. The typical size of the compact young clusters observed in the Galactic bulge is $\simeq 0.3\,\mathrm{pc}$ (Figer *et al.*, 1999), which corresponds to $\epsilon \simeq 10\epsilon_0$. With this value of ϵ, from eq. (5.11) and table 5.8, we obtain $\ln\Lambda \simeq 2.9$, about 60% less than the value for a point mass. The infall time is roughly doubled. For our choice of object mass, $M \simeq 62\,300\ \mathrm{M}_\odot$, and initial galactocentric radius, $R_0 \simeq 7$ pc, we have an infall time that increases from $\simeq 6\,\mathrm{Myr}$ for the point mass, to $\simeq 12.5\,\mathrm{Myr}$ for an object of typical size $\simeq 10\epsilon_0 \simeq 0.3\,\mathrm{pc}$.

We also studied the uncertainty associated with the maximal impact parameter b_{max}. We found that for an infall to the Galactic centre, the infalling object is mostly influenced by the density peak at the Galactic centre itself. A good choice for b_{max} is then $b_{max} \simeq \beta R_0$, where R_0 is the initial galactocentric radius, and $\beta \simeq 0.5$

5.5 Discussion

We simulated the evolution of a massive particle in a sea of lighter particles in a self gravitating system. The main goal of this simulations is to obtain an accurate value of the Coulomb logarithm ($\ln\Lambda$). This helps us to understand the dynamics of the Galactic bulge and the rate at which intermediate mass black holes sink to the Galactic centre. We also study the effect of the finite size of the inspiraling object.

We ran both N-body particle-particle (PP) simulations, softened treecode simulations, and particle-mesh (PM) simulations. The comparative simulations are performed for 80 000 particles, and all result in the same value of $\ln\Lambda$. For a point particle near the Galactic centre we find $\ln\Lambda = 6.6 \pm 0.6$. In addition we measure the change in the Coulomb logarithm with respect to the softening parameter ϵ, which reveals $\Lambda \propto 1/\epsilon$. We argue that ϵ can be interpreted as the typical length of a finite size object, such as a star cluster, so that $\ln\Lambda$ as a function of ϵ can be seen as a first approximation of the dependence of the Coulomb logarithm on the size of an infalling star cluster.

We also observed a value of the maximal impact parameter b_{max} different from the customarily assumed value, which is proportional to the system size. We found that our results are more consistent to a value of b_{max} linearly dependent on the BH galactocentric radius, which is in agreement with Hashimoto *et al.* (2003).

We performed simulations with up to two million particles using a treecode. The obtained value of $\ln\Lambda$ does not depend on the number of particles. Apparently, 80 000 particles is already enough to eliminate any granularity for our choice of initial conditions. The results of the treecode, at the low N-limit, are in excellent agreement with the PP simulations, and we find the same scaling with respect to ϵ. Increasing the black hole mass reduces the time scale for spiral-in as expected from theory (see McMillan & Portegies Zwart, 2003).

Finally we compared the results of our PP and treecode simulations with a particle mesh (PM) method. We compared the methods for N up to two million. The results of our PP, treecode, and PM calculations are in good agreement. The cell size in the PM model is directly comparable to the softening length ϵ in the PP and tree methods.

This work is a first step in the direction of performing a simulative study of the infall of a young star cluster to the Galactic centre (see section 5.4.2). As discussed in section 1.7, a star-by-star simulation of a cluster infall is problematic. The total number of particles, including both the cluster particles and the Galactic centre particles, is by far larger than the number a direct code can manage. On the other hand, the use of a treecode would lead to an incorrect treatment of the cluster dynamics, resulting in a too fast, unrealistic cluster evaporation. A solution for this problem is the development of a hybrid code, consisting of a direct code "phase" that is responsible of the simulation of the cluster, and a treecode "phase" that simulates the galactic centre. The data exchange between the two phases is negligible: the treecode input is the current mass of the cluster, and the direct code input is the current value of the galactic gravitational force.

A hybrid architecture of the kind described in part I is an ideal hardware platform for

this hybrid code simulations, and the pseudo-particle treecode described in chapter 4 lends itself very well to being included in the hybrid code as its low accuracy component.

Part III

Conclusions

In this thesis we studied the use of hybrid architectures as a tool to accelerate the numerical solution of the N-body problem. We can define the N-body problem as the challenge to understand the motion of N point-like particles, subject to their mutual interactions. This definition is inspired by the one given on page 14 of the profound and delightful book on the gravitational million body problem, written by Douglas Heggie and Piet Hut (2003). In this thesis, we also focus on the gravitational N-body problem because of its highly demanding computational requirements, that make it interesting and stimulating for a computational scientist to examine the computational characteristics, and explore the possible avenues to speed-up the problem's numerical solution.

The analytic insolubility of the gravitational N-body problem for $N > 2$ led to the development of numerical techniques to study it, as discussed in section 1.2. The need to retain the full $\mathcal{O}(N^2)$ direct particle-particle scheme for the simulation of collisional systems, i.e. systems requiring high computational accuracy (see section 1.2), led in turn to the development of the GRAPE, specialised hardware to accelerate the computation of the gravitational force interactions, introduced in section 1.3.

The gravitational force computational kernel of the $\mathcal{O}(N^2)$ scheme is so demanding in terms of hardware power, that its execution on the one Tflop/s GRAPE-4 can be driven by a host workstation which is about 10^4 times slower (see section 1.2, fig. 1.5, and our performance analysis work presented in chapter 2).

Our research aims at exploring the possibilities of using the GRAPE to boost N-body simulations other than those of astronomical collisional systems, which is the native domain of application of the GRAPE. For the simulation of collisionless systems, fast and more sophisticated methods like the treecode, the FMM (Fast Multipole Method), or the PM (Particle-Mesh), have been developed, as described in section 1.2. They reduce the computational complexity of the N-body problem to $\mathcal{O}(N \log N)$, and even $\mathcal{O}(N)$, trading higher speed for lower accuracy.

Using the GRAPE with these methods is not straightforward, since the particle-particle force computation, while still relevant, is no longer the most computationally expensive task of the application. In this case, the relatively high computational load of the GRAPE-host

could make the host activity the system bottleneck.

The use of Performance Modelling techniques allows us to study the interplay of the general purpose host, the special purpose device, and the application executed on the hybrid architecture. Chapter 3 describes our performance models of hybrid architectures which include GRAPE boards to accelerate the execution of direct particle-particle codes and treecodes. We show that a distributed hybrid architecture running a treecode optimised for the GRAPE has the potential to outperform a serial-host GRAPE4-like monolithic system running a direct code, even for simulations requiring high computational accuracy.

This supports our idea of hybrid architectures as an effective tool to speed up the execution of applications characterised by a heavy and small computational kernel, which is amenable to hardware implementation, but also including auxiliary relevant computational requirements. Further support for the hybrid architecture paradigm comes from the fact that the GRAPE-6 architecture (see section 1.3) is essentially an instance of this class of computer systems.

In order to efficiently use the hybrid architecture, the software application that is run on it needs to be fine-tuned. A fine-tuning of the treecode to run optimally on a platform which includes the GRAPE is the pseudo-particle scheme (see chapter 4). In this formulation, the gravitational potential of the particle aggregates created by the treecode domain partition, is converted from multipole expansion to a pseudo-particle distribution. In this way, the GRAPE is also able to compute the force due to particle aggregates, allowing for an optimal execution of the treecode on the hybrid machine.

In chapter 4 we described the pseudo-particle approach, and presented the accuracy improvement that we developed. We showed that the original pseudo-particle formulation (Makino, 1999; Kawai & Makino, 2001) is less accurate than the canonical multipole-based treecode, especially in the case of highly inhomogeneous matter distributions. We introduced an improvement, consisting in an extra particle added to the pseudo-particle set, located at the centre of mass of the real particle distribution. In such a way the pseudo-particles approximate inhomogeneous real particle distributions more closely, with a significant accuracy benefit.

We also developed a temporal expansion scheme for the pseudo-particle approach, where we define a pseudo-particle velocity. In this way, we do not re-compute the pseudo-particles at each time step, but we let them move, following the real particle evolution. This not only reduces the overhead from calculating the pseudo-particle, but also optimises the communication with the GRAPE. The GRAPE needs to know the pseudo-particle expansion of particle aggregates, which must be communicated by the host at each time step. With our scheme, pseudo-particle expansions can be retained in the GRAPE memory for a number of time steps, and evolved locally by the GRAPE extrapolation pipeline (see section 1.3 and fig. 1.4), because pseudo-particles now have a velocity assigned to them.

In chapter 5 we presented an actual example of N-body simulative study. Namely, we carried out a comparative study of the infall of a massive black hole towards the Galactic centre, in order to measure the Coulomb logarithm. The Coulomb logarithm is the parameter that quantifies the efficiency of a massive body slow-down due to its gravitational interaction

with a background of lighter stars. If the body is orbiting a centre of gravity, as in our case, its deceleration results in an inspiraling trajectory.

We used a direct particle-particle code, a treecode, and a particle-mesh code for our simulations, in order to understand how particle granularity and code inaccuracy influence the measure of the Coulomb logarithm. The direct code is accurate, but limited in the number of particles because of its $\mathcal{O}(N^2)$ scaling. The other codes are less accurate, but allow for higher numbers of particles, hence low particle granularity, which gives a smoother representation of the gravity field. Our measure of the Coulomb logarithm appears to be independent of the number of particles, at least for the range of physical parameters chosen. The parameter that quantifies the numerical inaccuracy influences the value of the Coulomb logarithm. The logarithm argument is inversely proportional to the inaccuracy parameter.

Future work

The work presented in chapter 5 is the first step towards the study of the infall of a star cluster, i.e. an extended object. As described in section 1.7, phenomena of this kind, consisting of the interaction of a collisional and a collisionless system, are not straightforward to simulate, either with a direct code or an approximate method. Our solution is to devise a hybrid code, where the star cluster collisional "phase" is simulated by the direct code, and the Galactic centre collisionless "phase" is simulated by the treecode.

This hybrid code makes high demands of the host computing platform, which must provide both high computational power for the gravitational force evaluation, and comparably high power for the other tasks of the software, otherwise the treecode execution would become the system bottleneck. A hybrid architecture of the kind discussed in this thesis would be an ideal solution for this problem.

Our group is planning to realise a hybrid system including GRAPE-6 boards, configured to optimally run the hybrid code. The computational asymmetry inherent in the coexistence of a direct code "phase" and a treecode "phase" calls for a corresponding asymmetry in the hardware architecture. The direct code requires a large computational power for the gravitational force kernel evaluation, and has few requirements for the other tasks. A single host having a number of GRAPE boards connected to it would fulfill these needs. On the other hand, the treecode has a relatively lower need for the force evaluation part, but needs more power for the other general purpose tasks. A homogeneous distributed configuration with each node of a parallel host connected to the same number of GRAPEs would be more appropriate in this case.

The ideal architecture to run the hybrid code would then be an amalgam of the two platforms. One node should be connected to a higher number of GRAPEs with respect to the other nodes. A performance model will be very useful to configure this system, and fine-tune it to obtain the best performance when running the hybrid code. Our work on performance modelling that was presented in part I will be the basis of this model.

We have presented an example of the potential of hybrid architectures to solve specific,

highly demanding computational problems. We focussed on hybrid architectures for the gravitational N-body problem, but other fields of Computational Science can profit from this architectural paradigm. The use of special hardware like the APE series (Tripiccione, 1999) developed for Quantum Chromo-Dynamics simulations, or the MDM machine (Narumi *et al.*, 1999, 2000), the sibling of GRAPE used for Molecular Dynamics simulations and mentioned in section 1.3.3, can be enhanced by a hybrid architecture approach. We believe that hybrid architectures can be the platform of choice for a sizable number of computational scientists. With this thesis we hope to contribute to a move in this direction.

Bibliography

Aarseth, S. J. (1963). Dynamical evolution of clusters of galaxies, I. *Monthly Notices of the Royal Astronomical Society* **126**, 223–255.

Aarseth, S. J. (1972). In M. Lecar (Ed.), *The Gravitational N-body Problem*, Dordrecht, pp. 373. Reidel.

Aarseth, S. J. (1985). Direct methods for N-body simulations. In J. Brackhill & B. Cohen (Eds.), *Multiple Time Scales*, Orlando, pp. 377–418. Academic Press.

Aarseth, S. J. (1999). From NBODY1 to NBODY6: The growth of an industry. *Publications of the Astronomical Society of the Pacific* **111**, 1333–1346.

Aarseth, S. J. (2001). NBODY2: A direct N-body integration code. *New Astronomy* **6**, 277–291.

Adve, V. S., Bagrodia, R., Browne, J. C., Deelman, E., Dube, A., Houstis, E. N., Rice, J. R., Sakellariou, R., Sundaram-Stukel, D. J., Teller, P. J., & Vernon, M. K. (2000). POEMS: End-to-end performance design of large parallel adaptive computational systems. *IEEE Transactions on Parallel and Distributed Systems* **26**, 1027.

Adve, V. S. & Sakellariou, R. (2000). Application representations for multiparadigm performance modeling of large-scale parallel scientific codes. *International Journal of High Performance Computing Applications* **14**, 304.

Aida, K., Takefusa, A., Nakada, H., Matsuoka, S., Sekiguchi, S., & Nagashima, U. (2000). Performance evaluation model for scheduling in global computing systems. *International Journal of High Performance Computing Applications* **14**, 268.

Aluru, S. (1996). Greengard's N-body algorithm is not order N. *SIAM Journal of Scientific Computing* **17**, 773–776.

Anderson, C. R. (1992). An implementation of the Fast Multipole Method without multipoles. *SIAM Journal of Scientific and Statistical Computing* **13**, 923–947.

Aoki, S., Burkhalter, R., Kanaya, T., Yoshié, T., Boku, T., Nakamura, H., & Yamashita, Y. (1999). Performance of lattice QCD programs on CP-PACS. *Parallel Computing* **25**, 1243–1255.

Athanassoula, E., Bosma, A., Lambert, J.-C., & Makino, J. (1998). Performance and accuracy of a GRAPE-3 system for collisionless N-body simulations. *Monthly Notices of the Royal Astronomical Society* **293**, 369–380.

Athanassoula, E., Fady, E., Lambert, J.-C., & Bosma, A. (2000). Optimal softening for force calculations in collisionless N-body simulations. *Monthly Notices of the Royal Astronomical Society* **314**, 475–488.

Bagrodia, R., Meyer, R., Takai, M., Chen, Y., Zeng, X., Martin, J., & Ha Yoon Song, H. Y. (1998). Parsec: A parallel simulation environment for complex systems. *Computer* **31 (10)**, 77.

Bal, H. *et al.* (2000). The distributed ASCI supercomputer project. *ACM Computing Surveys* **34**, 76–96.

Barnes, J. E. (1990). A modified tree code: Don't laugh, it runs. *Journal of Computational Physics* **87**, 161.

Barnes, J. E. & Hut, P. (1986). A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature* **324**, 446–449.

Bhattacharya, D. & van den Heuvel, E. P. J. (1991). Formation and evolution of binary and millisecond radio pulsars. *Physics Reports* **203**, 1–124.

Binney, J. & Tremaine, S. (1987). *Galactic Dynamics.* Princeton, NJ (USA): Princeton University Press.

Capuzzo Dolcetta, R. & Miocchi, P. (1998). A comparison between the Fast Multipole Algorithm and the tree-code to evaluate gravitational forces in 3-D. *Journal of Computational Physics* **143**, 29–48.

Capuzzo Dolcetta, R., Pucello, N., Rosato, V., & Saraceni, F. (2001). On the use of a heterogeneous MIMD-SIMD platform to simulate the dynamics of globular clusters with a central massive object. *Journal of Computational Physics* **174**, 208–225.

Carrier, J., Greengard, L., & Rokhlin, V. (1988). A fast adaptive multipole algorithm for particle simulations. *SIAM Journal of Scientific and Statistical Computing* **9**, 669–686.

Chandrasekhar, S. (1943). Dynamical friction I. general considerations: the coefficient of dynamical friction. *Astrophysical Journal* **97**, 255–262.

Cheng, H., Greengard, L., & Rokhlin, V. (1999). A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics* **155**, 468–498.

Cionco, R. & Brunini, A. (2002). Orbital migrations in planetesimal discs: N-body simulations and the resonant dynamical friction. *Monthly Notices of the Royal Astronomical Society* **334**, 77–86.

Colpi, M. (1998). Accretion of a satellite onto a spherical galaxy: Binary evolution and orbital decay. *Astrophysical Journal* **502**, 167–176.

Colpi, M., Mayer, L., & Governato, F. (1999). Dynamical friction and the evolution of satellites in virialized halos: the theory of linear response. *Astrophysical Journal* **525**, 720–733.

Colpi, M. & Pallavicini, A. (1998). Drag on a satellite moving across a spherical galaxy: Tidal and frictional forces in short-lived encounters. *Astrophysical Journal* **502**, 150–166.

Cora, S. A., Muzzio, J. C., & Vergne, M. M. (1997). Orbital decay of galactic satellites as a result of dynamical friction. *Monthly Notices of the Royal Astronomical Society* **289**, 253–262.

Couchman, H., Thomas, P., & Pearce, F. (1996). Hydra: an adaptive-mesh implementation of P^3M-SPH. *Astrophysical Journal* **452**, 797–813.

Cremonesi, P. & Gennaro, C. (2002). Integrated performance models for SPMD applications and MIMD architectures. *IEEE Transactions on Parallel and Distributed Systems* **13**, 745.

Dehnen, W. (1993). A family of potential-density pairs for spherical galaxies and bulges. *Monthly Notices of the Royal Astronomical Society* **265**, 250–256.

Dikaiakos, M. D., Rogers, A., & Steiglitz, K. (1996). Functional algorithm simulation of the Fast Multipole Method: Architectural implications. *Parallel Processing Letters* **6**, 55.

Ewald, P. P. (1921). Die berechnung optischer und elektrostatischer gitterpotentiale. *Annalen der Physik* **64**, 253–287.

Fellhauer, M., Kroupa, P., Baumgardt, H., Bien, R., Boily, C., Spurzem, R., & Wassmer, N. (2000). SUPERBOX – an efficient code for collisionless galactic dynamics. *New Astronomy* **5**, 305–326.

Figer, D. F., Kim, S. S., Morris, M., Serabyn, E., Rich, R. M., & McLean, I. S. (1999). *Hubble Space Telescope*/NICMOS observations of massive stellar clusters near the galactic center. *Astrophysical Journal* **525**, 750–758.

Freund, R. F. & Siegel, H. J. (1996). Heterogeneous processing. *Computer* **26 (6)**, 13.

Fukushige, T. & Makino, J. (1996). N-body simulation of galaxy formation on the GRAPE-4 special purpose computer. In *Proc. of Supercomputing '96 Conference*. ACM press.

Fukushige, T., Taiji, M., Makino, J., Ebisuzaki, T., & Sugimoto, D. (1996). A highly parallelized special-purpose computer for many-body simulations with an arbitrary central force: MD-GRAPE. *Astrophysical Journal* **468**, 51–61.

Funato, Y., Hut, P., McMillan, S. L. W., & Makino, J. (1996). Time-symmetrized Kustaanheimo-Stiefel regularization. *Astronomical Journal* **112**, 1697.

Gerhard, O. (2001). The galactic center He I stars: Remains of a dissolved young cluster. *Astrophysical Journal* **546**, L39–L42.

Goldreich, P. & Tremaine, S. (1980). Disk-satellite interactions. *Astrophysical Journal* **241**, 425.

Greengard, L. (1988). *The Rapid Evaluation of Potential Fields in Particle Systems.* Cambridge, MA (USA): MIT Press.

Greengard, L. & Rokhlin, V. (1997). A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica* **6**, 229–269.

Gunther, N. J. (2000). The dynamics of performance collapse in large-scale networks and computers. *International Journal of High Performance Computing Applications* **14**, 367.

Hashimoto, Y., Funato, Y., & Makino, J. (2003). To circularize or not to circularize? – orbital evolution of satellite galaxies. *Astrophysical Journal* **582**, 196–201.

Heggie, D. C. & Hut, P. (2003). *The Gravitational Million Body Problem.* Cambridge, England (UK): Cambridge University Press.

Heggie, D. C. & Mathieu, R. D. (1985). Standardised units and time scales. In S. L. W. McMillan & P. Hut (Eds.), *The Use of Supercomputers in Stellar Dynamics*, Berlin, pp. 233–235. Springer Verlag.

Hernquist, L. (1990). An analytical model for spherical galaxies and bulges. *Astrophysical Journal* **356**, 359–364.

Hockney, R. W. (1965). A fast direct solution of Poisson's equation using Furier analysis. *Journal of the ACM* **12**, 95–113.

Hockney, R. W. & Eastwood, J. W. (1988). *Computer Simulation Using Particles.* Bristol, England (UK): IOP Publishing.

Hoisie, A., Lubeck, O., & Wassermann, H. (2000). Performance and scalability analysis of teraflop-scale parallel architectures using multidimentional wavefront applications. *International Journal of High Performance Computing Applications* **14**, 330.

Hut, P. (1996). The role of binaries in the dynamical evolution of globular clusters. In E. Milone & J.-C. Mermilliod (Eds.), *Origins, Evolution, and Destinies of Binary Stars in Clusters. ASP Conference Proceedings Series, vol. 90.* Astronomical Society of the Pacific press. `astro-ph/9602158`

Hut, P. & Makino, J. (1999). Astrophysics on the GRAPE family of special purpose computers. *Science* **283**, 501–505.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis.* New York, NY (USA): John Waley & Sons.

Just, A. & Peñarrubia, J. (2003). Dynamical friction in inhomogeneous systems. *Astronomy and Astrophysics* **submitted.**

Karp, A. H. (1993). Speeding up N-body calculations on machines without hardware square root. *Scientific Programming* **1**, 133–140.

Kawai, A. (1999, December). *Implementation of the Barnes-Hut Treecode on GRAPE Special-Purpose Computers.* Ph. D. thesis, University of Tokyo. Available at: `http://atlas.riken.go.jp/~atsushi/paper/d-thesis.ps`.

Kawai, A., Fukushige, T., & Makino, J. (1999). $7.0/Mflops astrophysical N-body simulation with treecode on GRAPE-5. In *Proc. of Supercomputing99 Conference.* ACM press.

Kawai, A., Fukushige, T., Makino, J., & Makoto, T. (2000). GRAPE-5: A special-purpose computer for N-body simulations. *Publications of the Astronomical Society of Japan* **52**, 659–676.

Kawai, A., Fukushige, T., Taiji, M., Makino, J., & Sugimoto, D. (1997). The PCI interface for GRAPE systems: PCI-HIB. *Publications of the Astronomical Society of Japan* **49**, 607–618.

Kawai, A. & Makino, J. (1999). A simple formulation of the Fast Multipole Method: Pseudo-particle multipole method. In B. Hendrickson *et al.* (Eds.), *Proceedings of the Nineth SIAM Conference on Parallel Processing for Scientific Computing.* SIAM press. `astro-ph/9812431`

Kawai, A. & Makino, J. (2001). Pseudoparticle multipole method: A simple method to implement high-accuracy treecode. *Astrophysical Journal* **550**, L143–L146.

Kim, S. S. & Morris, M. (2002). Dynamical friction on star clusters near the galactic center. *in preparation.*

Kokubo, E. & Ida, S. (1996). On runaway growth of planetesimals. *Icarus* **123**, 180–191.

Kokubo, E. & Ida, S. (1998). Oligarchic growth of protoplanets. *Icarus* **131**, 171–178.

Krabbe, A., Genzel, R., Eckart, A., Najarro, F., Lutz, D., Cameron, M., Kroker, H., Tacconi-Garman, L. E., Thatte, N., Weitzel, L., Drapatz, S., Geballe, T., Sternberg, A., & Kudritzki, R. (1995). The nuclear cluster of the Milky Way: Star formation and velocity dispersion in the central 0.5 parsec. *Astrophysical Journal* **447**, L95.

Kurc, T., Uysal, M., Eom, H., Hollingsworth, J., Saltz, J., & Sussman, A. (2000). Efficient performance prediction for large-scale, data-intensive applications. *International Journal of High Performance Computing Applications* **14**, 216.

Kustaanheimo, P. & Stiefel, E. (1965). Perturbation theory of Kepler motion based on spinor regularization. *Journal für die Reine und Angewandte Mathematik* **218**, 204–219.

Lissauer, J. J. (1993). Planet formation. *Annual Review of Astronomy and Astrophysics* **31**, 129–174.

Makino, J. (1991*a*). A modified Aarseth code for GRAPE and vector processors. *Publications of the Astronomical Society of Japan* **43**, 859–876.

Makino, J. (1991*b*). Treecode with a special-purpose processor. *Publications of the Astronomical Society of Japan* **43**, 621–638.

Makino, J. (1996). Postcollapse evolution of globular clusters. *Astrophysical Journal* **471**, 796–803.

Makino, J. (1997). Merging of galaxies with central black holes. II. evolution of the black hole binary and the structure of the core. *Astrophysical Journal* **478**, 58–65.

Makino, J. (1999). Yet another Fast Multipole Method without multipoles — pseudoparticle multipole method. *Journal of Computational Physics* **151**, 910–920.

Makino, J. (2001*a*). Direct simulation of dense stellar systems with GRAPE-6. In S. Deiters, B. Fuchs, A. Just, R. Spurzem, & R. Wielen (Eds.), *Dynamics of Star Clusters and the Milky Way. ASP Conference Proceedings Series, vol. 228.* Astronomical Society of the Pacific press. `astro-ph/0007084`

Makino, J. (2001*b*). GRAPE project: special-purpose computers for many-body simulations. *Computer Physics Communications* **139**, 45–54.

Makino, J. (2001*c*). Next-generation massively parallel computers — massively parallel computer for particle-based simulations. In *Proceedings of the Fourth Symposium on Computational Science and Engineering.* Japan Society for the Promotion of Science. `http://grape.c.u-tokyo.ac.jp/~makino/papers/cse2001.ps.gz`

Makino, J. (2003, April). *GRAPE-6 User's Guide - Multi-Cluster version without monolithic configuration. Version 0.71a.* University of Tokyo. `http://grape.astron.s.u-tokyo.ac.jp/~makino/softwares/GRAPE6/`

Makino, J. & Aarseth, S. J. (1992). On a Hermite integrator with Ahmad-Cohen scheme for gravitational many-body problems. *Publications of the Astronomical Society of Japan* **44**, 141–151.

Makino, J. & Ebisuzaki, T. (1996). Merging of galaxies with central black hole I. hierarchical mergings of equal-mass galaxies. *Astrophysical Journal* **465**, 527–533.

Makino, J. & Fukushige, T. (2001). A 11.55 tflops simulation of black holes in a galactic center on GRAPE-6. In *Proc. of Supercomputing 2001 Conference*. ACM press.

Makino, J., Fukushige, T., & Koga, M. (2000). A 1.349 tflops simulation of black holes in a galactic center on GRAPE-6. In *Proc. of Supercomputing 2000 conference*. ACM press.

Makino, J. & Hut, P. (1988). Performance analysis of direct N-body calculations. *Astrophysical Journal Supplement Series* **68**, 833–856.

Makino, J., Kokubo, E., Fukushige, T., & Daisaka, H. (2002). A 22.72 Tflops simulation of planetesimals in Uranus-Neptune region on GRAPE-6. In *Proc. of Supercomputing 2002 Conference*. ACM press.

Makino, J. & Taiji, M. (1995). Astrophysical N-body simulations on GRAPE-4 special-purpose computer. In *Proc. of Supercomputing '95 Conference*. ACM press.

Makino, J. & Taiji, M. (1998). *Scientific Simulations with Special-Purpose Computers*. Chichester, England (UK): Waley.

Makino, J., Taiji, M., Ebisuzaki, T., & Sugimoto, D. (1997). GRAPE-4: a massively parallel special-purpose computer for collisional N-body simulations. *Astrophysical Journal* **480**, 432–446.

Maoz, E. (1993). A fluctuation-dissipation approach to dynamical friction in non-homogeneous backgrounds. *Monthly Notices of the Royal Astronomical Society* **263**, 75–85.

Mawhinney, R. D. (1999). The 1 Teraflops QCDSP computer. *Parallel Computing* **25**, 1281–1296.

McFarland, T., Couchman, H., Pearce, F., & Pichlmeier, J. (1998). A new parallel P^3M code for very large-scale cosmological simulations. *New Astronomy* **3**, 687–705.

McMillan, S. L. W. (1986). The vectorization of small-N integrators. In S. L. W. McMillan & P. Hut (Eds.), *The Use of Supercomputers in Stellar Dynamics*, Berlin, pp. 156–161. Springer Verlag.

McMillan, S. L. W. & Aarseth, S. J. (1993). An $O(N \log N)$ integration scheme for collisional systems. *Astrophysical Journal* **414**, 200–212.

McMillan, S. L. W. & Portegies Zwart, S. F. (2003). The fate of star clusters near the galactic center I: Analytic calculations. *Astrophysical Journal* **accepted**.

Message Passing Interface Forum (1997). *MPI-2: Extensions to the Message-Passing Interface*. Knoxville, TN (USA): University of Tennessee.

Meylan, G. & Heggie, D. C. (1997). Internal dynamics of globular clusters. *Astronomy and Astrophysics Review* **8**, 1–127.

Mezger, P., Zylka, R., Philipp, S., & Launhardt, R. (1999). The nuclear bulge of the galaxy. *Astronomy and Astrophysics* **348**, 457–465.

Mikkola, S. & Aarseth, S. J. (1990). A chain regularization method for the few-body problem. *Celestial Mechanics and Dynamical Astronomy* **47**, 375–390.

Milosavljević, M. & Merritt, D. (2001). Formation of galactic nuclei. *Astrophysical Journal* **563**, 34–62.

Morris, M. (1993). Massive star formation near the galactic center and the fate of the stellar remnants. *Astrophysical Journal* **408**, 496–506.

Nagata, T., Woodward, C. E., Shure, M., & Kobayashi, N. (1995). Object 17: Another cluster of emission-line stars near the galactic center. *Astronomical Journal* **109**, 1676.

Nagata, T., Woodward, C. E., Shure, M., Pipher, J. L., & Okuda, H. (1990). AFGL 2004 - an infrared quintuplet near the galactic center. *Astrophysical Journal* **351**, 83.

Narumi, T., Susukita, R., Ebisuzaki, T., McNiven, G., & Elmegreen, B. (1999). Molecular Dynamics Machine: Special-purpose computer for Molecular Dynamics simulations. *Molecular Simulation* **21**, 401–415.

Narumi, T., Susukita, R., Koishi, T., Yasuoka, K., Furusawa, H., Kawai, A., & Ebisuzaki, T. (2000). 1.34 Tflops Molecular Dynamics simulation for NaCl with a special-purpose computer: MDM. In *Proceedings of Supercomputing 2000 conference*. ACM Press.

Navarro, J. F., Frenk, C. S., & White, S. D. M. (1997). A universal density profile from hierarchical clustering. *Astrophysical Journal* **490**, 493–508.

Okuda, H., Shibai, H., Nakagawa, T., Matsuhara, H., Kobayashi, Y., Kaifu, N., Nagata, T., Gatley, I., & Geballe, T. (1990). An infrared quintuplet near the galactic center. *Astrophysical Journal* **351**, 89.

Palazzari, P., Arcipiani, L., Celino, M., Guadagni, R., Marongiu, A., Mathis, A., Novelli, P., & Rosato, V. (2000). Heterogeneity as key feature of high performance computing: the PQE1 prototype. In *Proceedings of the nineth Heterogeneous Computing Workshop*, pp. 17–30. IEEE Computer Society Press.

Pimentel, A. D., Hertzberger, L. O., Lieverse, P., van der Wolf, P., & Deprettere, E. F. (2001). Exploring embedded-systems architectures with Artemis. *Computer* **34 (11)**, 57.

Plummer, H. C. (1915). The distribution of stars in globular clusters. *Monthly Notices of the Royal Astronomical Society* **76**, 107–121.

Portegies Zwart, S. F., Makino, J., McMillan, S. L. W., & Hut, P. (1999). Star cluster ecology. III. runaway collisions in young compact star clusters. *Astronomy and Astrophysics* **348**, 117–126.

Portegies Zwart, S. F., McMillan, S. L. W., Hut, P., & Makino, J. (2001). Star cluster ecology - IV. dissection of an open star cluster: Photometry. *Monthly Notices of the Royal Astronomical Society* **321**, 199–226.

Salmon, J. K. & Warren, M. S. (1994). Skeletons from the treecode closet. *Journal of Computational Physics* **111**, 136–155.

Salmon, J. K. & Warren, M. S. (1997). Parallel, out-of-core methods for N-body simulation. In *Proc. of the eigth SIAM conference on Parallel Processing for Scientific Computing.* SIAM press.

Sauer, C. H. & Mani Chandri, K. (1981). *Computer Systems Performance Modeling.* Englewood Cliffs, NJ (USA): Prentice-Hall.

Spitzer, L. (1987). *Dynamical Evolution of Globular Clusters.* Princeton, NJ (USA): Princeton University Press.

Springel, V., Yoshida, N., & White, S. D. M. (2001). GADGET: a code for collisionless and gasdynamical cosmological simulations. *New Astronomy* **6**, 79–117.

Spurzem, R. (1999). Direct N-body simulations. *Journal of Computational and Applied Mathematics* **109**, 407–432.

Sugimoto, D., Chikada, Y., Makino, J., Ito, T., Ebisuzaki, T., & Umemura, M. (1990). A special-purpose computer for gravitational many-body problems. *Nature* **345**, 33–35.

Taffoni, G., Mayer, L., Colpi, M., & Governato, F. (2003). On the life and death of satellite haloes. *Monthly Notices of the Royal Astronomical Society* **341**, 434–448.

Tamblyn, P. & Rieke, G. H. (1993). IRS 16 - the galaxy's central wimp? *Astrophysical Journal* **414**, 573.

Tanenbaum, A. S. (2001). *Modern Operating Systems, 2nd ed.* Upper Saddle River, NJ (USA): Prentice-Hall.

Triendl, R. (2002). Our virtual planet. *Nature* **416**, 579–580.

Tripiccione, R. (1999). APEmille. *Parallel Computing* **25**, 1297–1309.

van den Bosch, F. C., Lewis, G. F., Lake, G., & Stadel, J. (1999). Substructure in dark halos: Orbital eccentricities and dynamical friction. *Astrophysical Journal* **515**, 50–68.

van Gemund, A. (1993). Performance prediction of parallel processing systems: The PAMELA methodology. In *Proceedings of seventh ACM International Conference on Supercomputing.* ACM press.

van Gemund, A. (2003). Symbolic performance modeling of parallel systems. *IEEE Transactions on Parallel and Distributed Systems* **14**, 154–165.

Warren, M. S. & Salmon, J. K. (1993). A parallel hashed oct-tree N-body algorithm. In *Proc. of Supercomputing '93 conference*, pp. 12–21. ACM press.

Warren, M. S. & Salmon, J. K. (1995). A portable parallel particle program. *Computer Physics Communications* **87**, 266–290.

Warren, M. S., Salmon, J. K., Becker, D. J., Goda, M. P., Sterling, T., & Winckelmans, G. S. (1997). Pentium Pro inside: I. A treecode at 430 Gigaflops on ASCI Red, II. Price/performance of $50/Mflop on Loki and Hyglac. In *Proc. of Supercomputing '97 conference*. ACM press.

Samenvatting

In dit proefschrift analyseren we hulpmiddelen die ontwikkeld zijn om de snelheid en de accuratesse van zogenaamde "N-body simulaties" te verhogen. De moleculen in een chemische oplossing, of de sterren in een sterrenhoop zijn voorbeelden van dergelijke N-body systemen. In dit proefschrift richten we ons op N-body systemen die gedreven worden door de zwaartekracht, zoals die worden toepast in de sterrenkunde.

N-body problemen zijn analytisch onoplosbaar. Met behulp van high-performance computing technieken en geavanceerde algoritmen kunnen ze echter wel numeriek worden aangepakt. Deze numerieke oplossingen vereisen zoveel computationeel vermogen, dat er geavanceerde algoritmen nodig zijn om het N-body probleem te versnellen. Hierbij wordt echter op de nauwkeurigheid van de oplossing ingeleverd. Een andere aanpak is het gebruik van gespecialiseerde hardware, die zich leent voor N-body problemen die meer numerieke accuratesse vereisen. Hiermee kan de numerieke oplossing van het N-body probleem met behoud van accuratesse worden versneld.

In dit proefschrift onderzoeken we de mogelijkheid om deze twee benaderingen te combineren, door de snelle, gespecialiseerde hardware in een conventionele, parallelle computer te integreren. We noemen dergelijke systemen *hybride architecturen.* Veel onderzoekers streven naar generalisatie en zijn meer gecharmeerd van conventionele systemen, die worden opgebouwd uit gewone hardware (PC's), zoals de Beowulf of het Grid. In de andere benadering streeft men ernaar zeer hoge prestaties door middel van hardwarespecialisatie te verkrijgen. Het doel van ons onderzoek is, deze twee benaderingen te overbruggen. We onderzoeken de levensvatbaarheid van hybride architecturen, en evalueren hun potentieel om grootschalige simulatieproblemen op te lossen.

De gespecialiseerde hardware die we bestuderen is de GRAPE. Dit is een zeer krachtige machine die wordt gebruikt voor de simulatie van sterrenkundige N-body systemen. Op GRAPE uitgevoerde simulaties hebben vijf keer in de laatste acht jaar de Gordon Bell prijs gewonnen, die jaarlijks wordt toegekend aan de snelste simulatie ter wereld.

Het is belangrijk om de interactie te begrijpen tussen de parallelle machine en de gespecialiseerde hardware enerzijds, en anderzijds de softwaretoepassing die op de hybride

architectuur wordt uitgevoerd. Zo kunnen knelpunten in het verloop van de simulatie worden gelocaliseerd, en kan de optimale configuratie worden gevonden. Om deze interactie te bestuderen maken we gebruik van *performance modelling*. In deze techniek worden simulaties gebruikt om de prestaties van simulatiesystemen te bestuderen. Deze meta-simulatie is de kern van ons onderzoek, en richt zich op het vinden van de optimale interactie tussen snelle software en hardware die uiteindeljk zal kunnen leiden tot de ontwikkeling van een zeer snelle simulatieomgeving voor N-body systemen.

Een belangrijk doel van dit proefschrift is om de potentie van hybride architecturen als optimale berekeningsomgeving voor de oplossing van specifieke problemen aan te tonen. In dit licht hebben we een numeriek algoritme bestudeerd en aangepast aan de hybride architectuur. Dit algoritme maakt het mogelijk om geavanceerde N-body codes te gebruiken op gespecialiseerde hardware. Dit numerieke algoritme is het softwarematige deel van de hybride architectuur, en vormt de basis voor de ontwikkeling van een efficiënte simulatieomgeving voor het N-body probleem.

Ten slotte bestuderen we de toepassing van de N-body simulaties in sterrenkundig onderzoek. We bestuderen de val van een massief object (een zwart gat) naar het centrum van het melkwegstelsel.

N-body simulaties zijn een effectief hulpmiddel om de tijdschaal van deze val te bestuderen; ze kunnen ondersteuning (of bezwaren) bieden aan theoretische modellen voor de verklaring van astronomische observaties.

Dit proefschrift bestaat uit drie delen. Het eerste deel behandelt de *performance modelling* en de simulatie hiervan, en bestaat uit twee hoofdstukken. In hoofdstuk 2 analyseren we de prestatie van de N-body-code NBODY1 op onze experimentele hybride architectuur, die bestaat uit twee GRAPE-4-borden en een parallelle computer. Het bevat een uitvoerige beschrijving van de taken van NBODY1, en bevat prestatiemetingen en een analyse van een aantal versies van NBODY1 op onze hybride architectuur. Deze metingen vormen de basis voor de performance modelling en simulatie van de architecturen waarop N-body-codes worden uitgevoerd. Dit *performance modelling and simulation* werk wordt geïntroduceerd in hoofdstuk 3.

Deel twee van dit proefschrift bestaat ook uit twee hoofdstukken. Hoofdstuk 4 behandelt de nauwkeurigheidsanalyse en optimalisering van een hybride N-body-code, die voor optimaal gebruik met de GRAPE ontwikkeld is. Wij bestuderen de foutpropagatie in de code bij verschillende deeltjesverdelingen, en verbeteren de nauwkeurigheid van de code wanneer ze gebruikt wordt bij zeer inhomogene distributies.

In hoofdstuk 5 introduceren we onze vergelijkende multi-methode N-body simulaties. Met behulp van deze simulaties beogen we kwantitatieve schattingen te doen van de efficiëntie van de spiralisering van een zwart gat naar het centrum van het melkwegstelsel. We proberen het effect van deeltjesgranulariteit en codeonnauwkeurigheid op de valefficiëntie van de zwarte gat te begrijpen. Tot slot, vatten we in deel III ons werk samen en bespreken we de toekomstige ontwikkelingen.

List of Publications

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance modelling of hybrid parallel systems*, Proceedings of the TUG99 meeting, Barcelona, 1999.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Analysis of Parallel N-Body Codes*, in M. Bubak; H. Afsarmanesh; R.D. Williams and L.O. Hertzberger, editors, Proceedings of the HPCN2000 Conference, LNCS vol. **1823**, pp. 249–260. Springer-Verlag, 2000.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance analysis of parallel N-body codes*, in L.J. van Vliet; J.W.J. Heijnsdijk; T. Kielmann and P.M.W. Knijnenburg, editors, Proceedings of the sixth annual conference of the Advanced School for Computing and Imaging, pp. 213–220. ASCI, 2000.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance modelling of hybrid architectures*, in U. Barberis *et al.*, editors, Proceedings of the SIMAI 2000 conference, Sommari - Abstracts, pp. 73–74. Società Italiana di Matematica Applicata e Industriale, Rome, 2000.

Bal, A., *et al.*, *The distributed ASCI supercomputer project*, Operating Systems Review, vol. **34** (4), pp. 76–96. Association for Computing Machinery, Special Interest Group on Operating Systems, 2000.

P.M.A. Sloot; P.F. Spinnato and G.D. van Albada: *N-body simulation on hybrid architectures*, in V.N. Alexandrov; J.J. Dongarra; B.A. Juliano; R.S. Renner and C.J.K. Tan, editors, Proceedings of the ICCS 2001 conference, LNCS vol. **2074**, pp. 883–892. Springer Verlag, 2001.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance of N-body Codes on Hybrid Machines*, Future Generation Computer Systems, **17**, 951–959, 2001.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Prediction of N-body Simulations on a Hybrid Architecture*, Computer Physics Communications, **139**, 34–44, 2001.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *A Simulator for Parallel Hybrid Computer Systems*, in R.L. Lagendijk; J.W.J. Heijnsdijk; A.D. Pimentel and M.H.F. Wilkinson, editors, Proceedings of the seventh annual conference of the Advanced School for Computing and Imaging, pp. 210–219. ASCI, 2001.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *A Versatile Simulation Model for Hierarchical Treecodes*, in P.M.A. Sloot; C.J.K. Tan; J.J. Dongarra and A.G. Hoekstra, editors, Proceedings of the ICCS2002 conference, LNCS vol. **2329**, pp. 176–185. Springer Verlag, 2002.

P.F. Spinnato and S.F. Portegies Zwart, *The Infall of Dense Star Clusters in the Galactic Centre*, Proceedings of the 57th annual conference of the NAC (Dutch Astronomers Club), 2002.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Modelling of Hierarchical N-body Codes Running on Hybrid Architectures*, in E.F. Deprettere; A.S.Z. Belloum; J.W.J. Heijnsdijk and F. van der Stappen, editors, Proceedings of the eighth annual conference of the Advanced School for Computing and Imaging, pp. 211–218. ASCI, 2002.

P.F. Spinnato; S.F. Portegies Zwart; M. Fellhauer; G.D. van Albada and P.M.A. Sloot: *Tools and Techniques for N-body Simulations*, in R. Capuzzo Dolcetta, editor, Proceedings of the 1st workshop on Computational Astrophysics in Italy: Methods and Tools, MemSAIt Suppl. Series vol. **1**, pp. 54–65. Società Astronomica Italiana, 2003.

P.F. Spinnato; M. Fellhauer and S.F. Portegies Zwart: *The Efficiency of the Spiral-in of a Black Hole to the Galactic Centre*, Monthly Notices of the Royal Astronomical Society, in press, 2003.

P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: *Performance Modelling of Distributed Hybrid Architectures*, IEEE Transactions on Parallel and Distributed Systems, in press, 2003.