Hybrid Systems for N-body Simulations

Spinnato, P.F.

Publication date
2003

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`

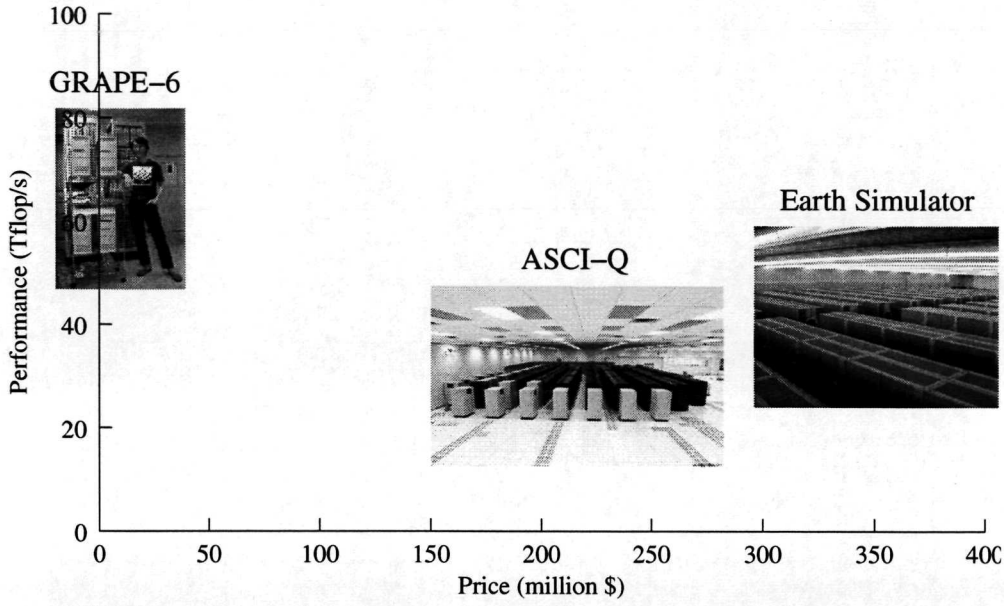

Figure 1.2: Price versus peak performance for the GRAPE-6 and the two fastest general purpose computers in the world. The physical size of the different systems can also be appreciated from the figure. The GRAPE-6 picture shows both the system and (on the right hand side) its main developer, Jun Makino.

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

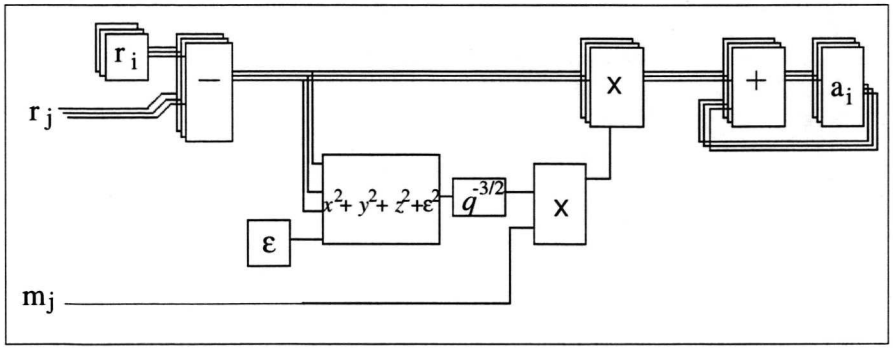

Figure 1.3: The pipeline for the force computation on the GRAPE. Figure adapted from (Makino, 2001b, fig. 4), reproduced here with author's permission. Here $\mathbf{r}_i$ is the position vector of the particle on which force is computed, stored in three pipeline registers. $\mathbf{r}_j$ and m_j are 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).

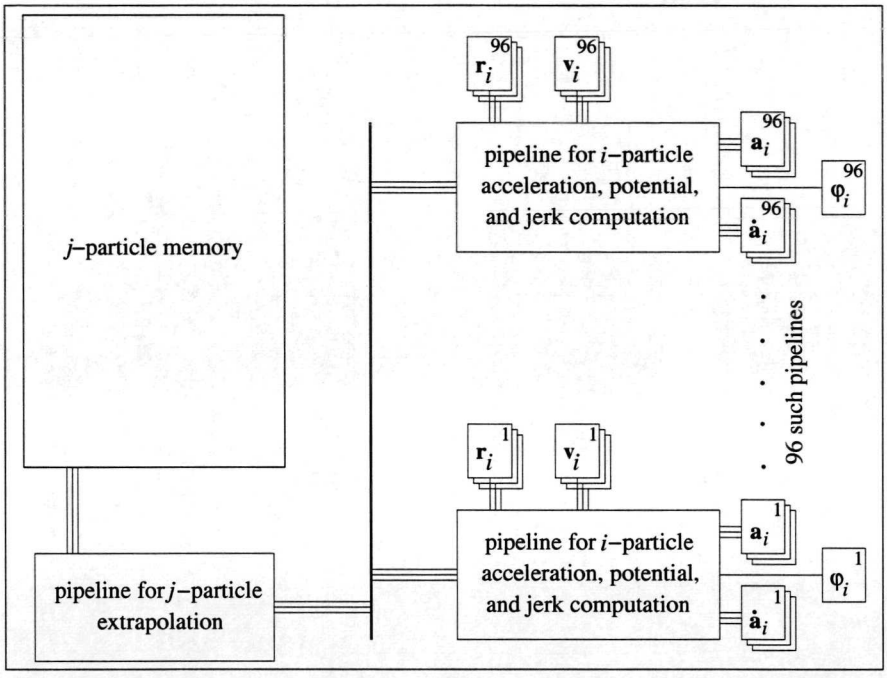

Figure 1.4: Sketch of a GRAPE-4 board. Force is computed on up to 96 i-particles simultaneously. At each cycle, a j-particle is loaded from the board memory, then its position and velocity are extrapolated to the i-particle time, these data are then fed into the acceleration and jerk (i.e. acceleration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

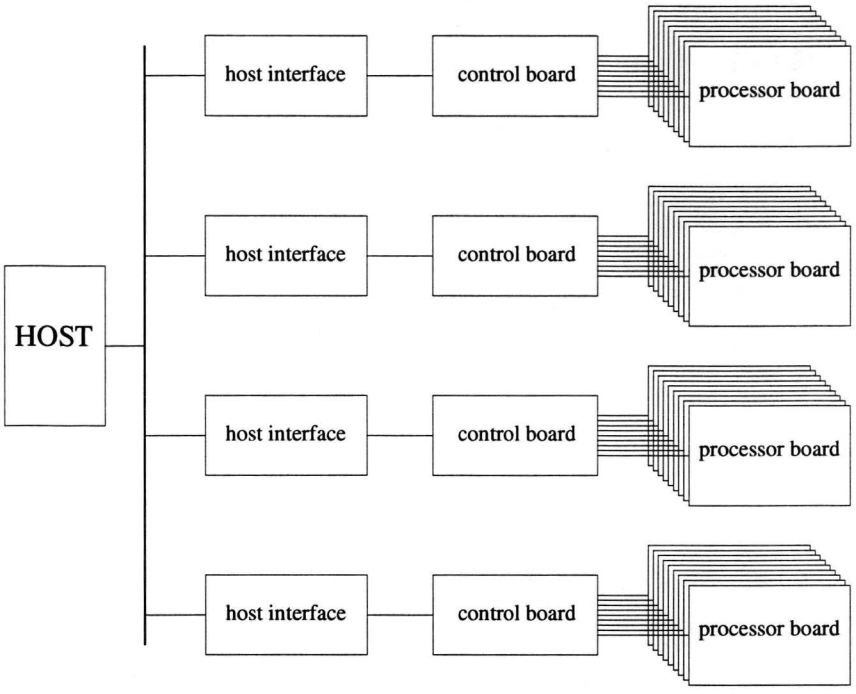

Figure 1.5: Sketch of the GRAPE-4 system at the University of Tokyo. The system consists of 36 processor boards as the one sketched in fig. 1.4, grouped in four clusters of nine boards each. The i-particle set is identical for each of the 36 boards. Each of the four control boards receives a different 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,

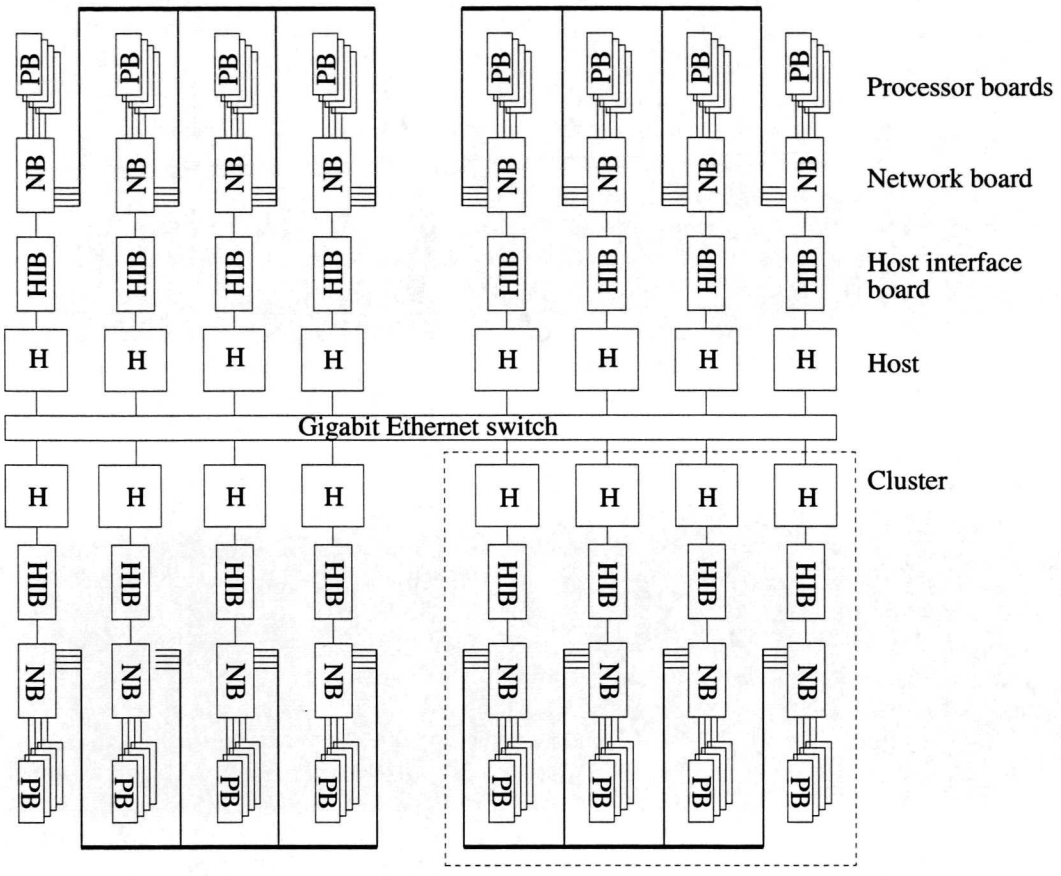

Figure 1.6: Sketch of the GRAPE-6 system at the University of Tokyo. Four clusters including four general purpose hosts and 16 processor boards are each connected by means of a Gigabit Ethernet switch. The processor boards in a cluster are able to communicate directly with each other by means of the 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:

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

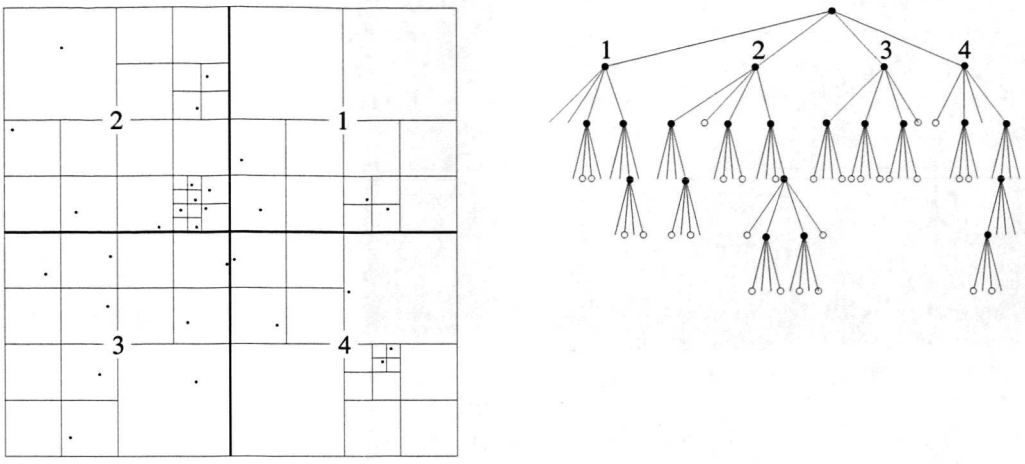

Figure 1.7: Sketch of the treecode space partition, and corresponding hierarchical tree data structure. The root cell, the one that encompasses the particle distribution, is recursively subdivided, until every particle is contained in a different cell. The corresponding tree data structure is shown on 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.

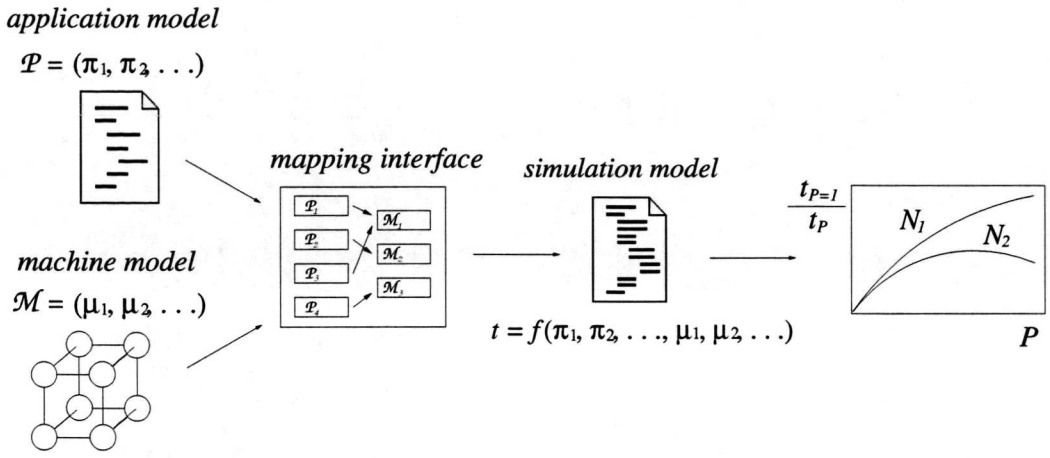

Figure 1.8: Performance modelling process. Figure adapted from the slides of the ASCI† course "Performance Modeling of Parallel Systems", taught by Arjan van Gemund, reproduced here with author's permission. The output of the simulation model, t, is the modelled execution time. The rightmost sketch represents 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.

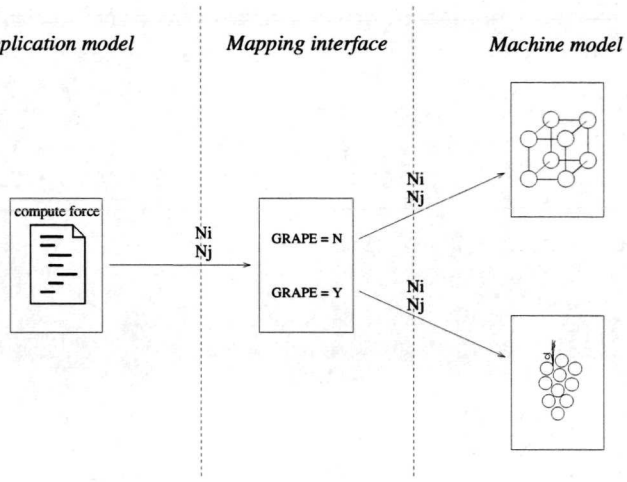

Figure 1.9: Sketch of the force evaluation task, as performed by our model. The application model module passes its parameters to the mapping module, that selects the architecture component that will perform the actual computation.

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.