3

Computational Physics Introduction

Following the evolution of a star cluster is among the most compute-intensive and delicate problems in science, let alone stellar dynamics. The main challenges are to deal with the extreme discrepancy of length- and time-scales, the need to resolve the very small deviations from thermal equilibrium that drive the evolution of the system, and the shear number of computations involved. Though numerical algorithms of many kinds are used, this is not an exercise in numerical analysis: the choice of algorithm and accuracy are dictated by the need to simulate the physics faithfully rather than to solve the equations of motion as exactly as possible.

Length/time-scale problem

Simultaneous close encounters between three or more stars have to be modeled accurately, since they determine the exchange of energy and angular momentum between internal and external degrees of freedom (Chapter 23). Especially the energy flow is important, since the generation of energy by double stars provides the heat input needed to drive the evolution of the whole system, at least in its later stages (Chapter 27f). Unfortunately, the size of the stars is a factor 10^9 smaller than the size of a typical star cluster. If neutron stars are taken into account, the problem is worse, and we have a factor of 10^{14} instead, for the discrepancy in length scales.

The time scales involved are even worse, a close passage between two stars taking place on a time scale of hours for normal stars, milliseconds for neutron stars (Table 1). In contrast, the time scale on which star clusters evolve can be as long as the age of the universe, of order ten

Table 1. Time Scales of the Million-Body Problem

Time Scale	Stellar Dynamics	Stellar Evolution	Human Evolution
Seconds	White dwarf collision	Formation of neutron star	Heartbeat
Years	Hard/soft binary period[1]	Long-period variable	Malt whisky
Myrs	Crossing time[2]	Shortest stellar lifetime	Human evolution
Gyrs	Relaxation time[3]	Lifetime of the sun	Life on Earth

[1] Chapter 19; [2] Chapter 1; [3] Chapter 14

billion years, giving a discrepancy of time scales of a factor 10^{14} for normal stars, and 10^{20} for neutron stars.

indextime scales

Sophisticated algorithms have been developed over the years to deal with these problems, using individual time step schemes, local coordinate patches, and even the introduction of mappings into four dimensions in order to regularize the 3-D Kepler problem (through a Hopf map to a 4-D harmonic oscillator, cf. Chapter 15). While these algorithms have been crucial to make the problem tractable, they are still very time-consuming.

Near-equilibrium problem

In the central regions of a star cluster, the two-body relaxation time scale, which determines the rate at which energy can be conducted through the system, can be far shorter than the time scale of evolution for the system as a whole, by several orders of magnitude. For example, in globular clusters, density contrasts between the centre and the half-mass radius can easily be as large as 10^4, which implies a similar discrepancy in relaxation time scales.

As a consequence, thermal equilibrium is maintained to a very high degree. Since it is precisely the deviation from thermal equilibrium that drives the evolution of the system (Chapter 2), it is extremely difficult to cut corners in the calculation of close encounters. If any systematic type of error would slip in here, even at the level of, say, 10^{-6}, the result could easily invalidate the whole calculation. It is for this reason that none of the recently developed fast methods for approximate force calculations has been adopted in this area, e.g. tree codes, P^3M codes, etc (Barnes & Hut 1986, Greengard 1990, Efstathiou et al. 1985). All such methods gain speed at the expense of relatively large errors in the force computation. The result is that the N-body simulations of stellar dynamics can boast far fewer particles than in, say cosmological simulations. This is a pity, because N is often used as a crude "figure of merit" in the art of simulation, whereas what really matters is the value of the science that comes out.

Computational requirements

The cpu cost of a direct N-body calculation scales $\propto N^3$, where the inter-particle forces contribute two powers in N (Problem 1) and the increased time scale for heat conduction contributes the third factor of N. For this reason the progress of N-body simulations of star clusters has been painfully slow, from the earliest published work of S. von Hoerner in 1960 (Fig.1).

Almost none of this progress has been made by large general-purpose supercomputers. The use of parallel computers, such as Cray-T3E, has had less impact on this area than on many others, because of the communication bottleneck. The force on each particle necessarily depends on the position of every other, and therefore it is not usually efficient to parallelise the force calculations (which are the main bottleneck in serial codes). Another way of exploiting parallelism is to advance many particles simultaneously (Spurzem & Baumgardt 2001). While this works well in simple cases, the enormous range of time scales can ruin the efficiency of this approach also: individual time steps were introduced precisely so

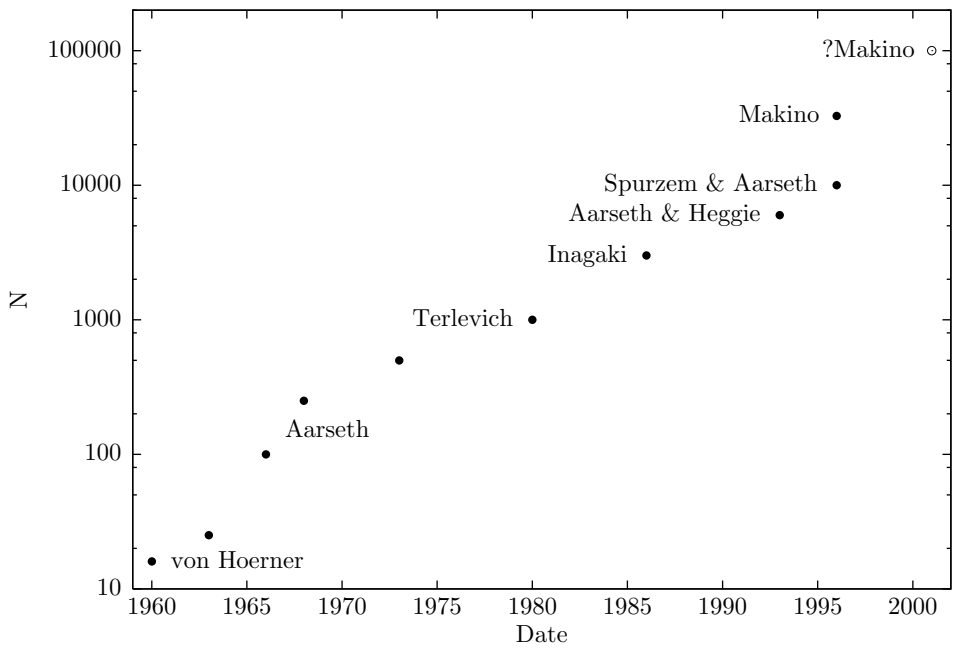

Fig. 1. The slow progress of N-body simulations of star clusters. Models computed well into the late evolution are plotted against publication date. For a human perspective, see Aarseth (1999) and von Hoerner (2001).

that it should not be necessary to advance all particles with the tiny time step required for one close binary!

Currently, with routine calculations, it is only feasible to model the evolution of a globular cluster containing a few thousand stars, since this requires some 10^{15} floating point calculations, equivalent to 10 Gflops-day, or several months to a year on a typical workstation. Therefore, a calculation with half a million stars, resembling a typical globular star cluster, will require ~ 10 Pflops-day.

In contrast, the memory requirements are and will remain very modest. All that is needed is to keep track of $N = 5 \times 10^5$ particles, each with a mass, position, velocity, and a few higher derivatives for higher-order integration algorithms. Adding a few extra diagnostics per particle still will keep the total number of words per particle to about 25 or so. With 200 bytes per particle, the total memory requirement will be a mere 100 Mbytes.

Output requirements will not be severe either. A snapshot of the positions and velocities of all particles will only take 10 Mbytes. With, say, 10^5 snapshot outputs for a run, the total run worth 10 Pflops-day will result in an output of only 1 Tbyte.

Special-purpose Hardware

While general-purpose supercomputers have not yet made much impact in this field, from time to time it has attracted attention as a possible application of *special*-purpose hardware (Fukushige et al. 1999). The earliest idea along these lines was put into practice by Holmberg as long ago as 1941 (Holmberg 1941; see also Tremaine 1981). He arranged a set of light bulbs like the stars in a stellar system, and used photometers to determine the illumination at each. Since light also obeys an inverse square law, this provided an analogue estimate of the gravitational field.

The next step in astronomy took place not in stellar dynamics but in celestial mechanics, with the building and development of the Digital Orrery (Applegate et al. 1985). For several years it performed groundbreaking calculations on the stability of the solar system, including the discovery of chaos in the motion of Pluto (Sussman & Wisdom 1988), until eventually being regretfully laid to rest in the Smithsonian Museum.

A significant step toward the modeling of globular star clusters was made in 1995 with the completion of a special-purpose piece of hardware, the GRAPE-4, by an ingenious group of astrophysicists at Tokyo University (Makino & Taiji 1998). GRAPE, short for GRAvity PipE, is the name of a family of pipeline processors that contain chips specially designed to calculate the Newtonian gravitational force between particles. A GRAPE processor operates in cooperation with a general-

purpose host computer, typically a normal workstation. Just as a floating point accelerator can improve the floating point speed of a personal computer, without any need to modify the software on that computer, so the GRAPE chips act as a form of Newtonian accelerator (Box 1).

The force integration and particle pushing are all done on the host computer, and only the inter-particle force calculations are done on the GRAPE. Since the latter require a computer power that scales with N^2, while the former only require power $\propto N$, load balance can always be achieved by choosing N values large enough.

For example, the complete GRAPE-4 configuration, with a speed of more than 1 Tflops, could be efficiently driven by a workstation of 100 Mflops. Although such a workstation operates at a speed that is lower than that of the GRAPE by a factor of 10^4, load balance could be achieved for particle numbers of order $N \sim 5 \times 10^5$. In practice, even with this hardware, routine calculations did not greatly exceed a particle number of about 10^4, since much larger simulations could not be completed in less than a few months, and it has been found scientifically more productive to compute large numbers of relatively modest simulations rather than to break records. (Note, by the way, that the extreme parallelism of

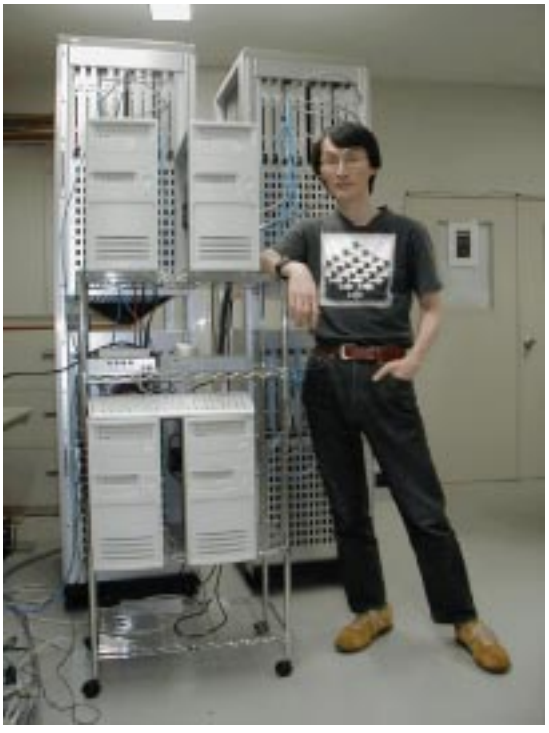

Fig. 2. The Grape-6 at the University of Tokyo, with J. Makino (right). With permission.

Box 1. *GRAPE design*

The design of a typical GRAPE chip (Fig.1) reflects the N-body equations (Eq.(1.1)). The position and mass of an attracting particle (index j) are read from memory, the difference $\mathbf{r} = \mathbf{r}_i - \mathbf{r}_j$ is computed, then r^2, then r^3, and finally one term on the right of the equations of motion. Contributions from all attracting particles are summed. One reason for the efficiency of GRAPE is the fact that, because of the "pipelined" design, one contribution is computed for each clock cycle. On a conventional computer each arithmetic operation usually requires several clock cycles, and each contribution requires about 30 such operations.

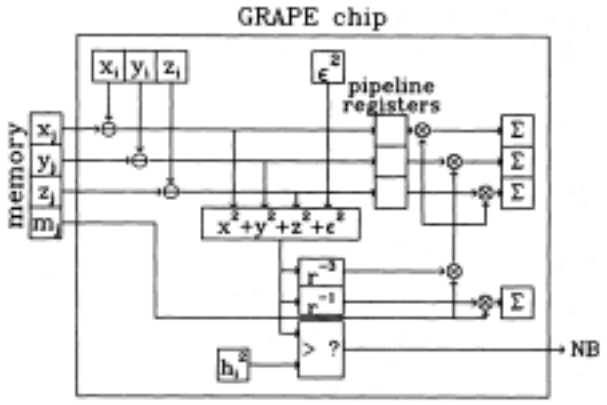

Fig. 1 Schematic of a GRAPE-3 chip (from Okumura et al. 1993).

This and other versions of the GRAPE chips perform a few other intensive calculations at the same time. Also, several chips or pipelines are installed, along with control hardware and memory for the particle data, on one board, rather like the mother board of a conventional PC. This board communicates with the host computer via a cable and an interface board, such as a PCI card. In larger installations, several GRAPE boards are arranged in a tree with suitable communications interfaces to the single host computer.

In order to make use of an installed GRAPE, sections of a simulation program are replaced by calls to software libraries which have been written by the GRAPE team in Tokyo. For example, computation of forces (and potential) on all n particles on the now-obsolete GRAPE-3 with 8 chips was computed as follows, using the library function `g3frc`:

the GRAPE does not allow the most efficient scalar algorithm to be implemented.) In addition, cut-down versions of this computer can be

Box 1 (continued)

```
    do 119 i=1,n,8
       ii = 8
       if(i+ii .gt.  n+1) ii = n- i + 1
       call g3frc(x(1,i),awork,pwork,ii)
       do 1198 j = 1,ii
          do 1197 k = 1,3
          f(k,i+j-1)=awork(k,j)
1197      continue
          pot(i+j-1) = pwork(j) + mass(i+j-1)*epsinv
1198   continue
119 continue
```

The particle data are loaded beforehand with similar instructions.

and have been used for simulations in a wide range of other problems in astrophysics (Hut & Makino 1999) and, indeed, other fields of science, such as plasma physics, molecular dynamics, the study of turbulence, and even protein folding.

There are several reasons for GRAPE's success. In the first place it was developed quickly, always keeping ahead of general-purpose computers. Secondly, the mathematical model of inverse square laws is quite fixed, and can be "hard-wired". Thirdly, the GRAPE group ensured that the devices could be made available to potential users throughout the world, and this maximised the scientific returns.

The introduction of special-purpose hardware has been a truly revolutionary advance, and not just in speed. Before GRAPE and its predecessor, the Digital Orrery, the hardware used by theorists was bought off the shelf from a computer dealer. By contrast observers have always built their own hardware (or have had it built to their own specification), even back to the time of Galileo. From this perspective, GRAPE represents a remarkable culture shift in the way theorists can do science. In retrospect it is not surprising that it is in the area of dynamical astronomy that this has happened, as it is here that the governing equations and the underlying physical model are most stable. And we are not yet at the end of the road: GRAPE 5 is already at work (Kawai et al. 2000) and, as we write, GRAPE 6 is coming on stream (Fig.2). It is about 100 times faster than GRAPE 4.

Software Environments

Generating data is only half the job in any simulation. The other half of the work of a computational theorist parallels that of an observer, and lies in the job of data reduction. As in the observational case, here too a good set of tools is essential. And not only that: unless the tools can be used in a flexible and coherent software environment, their usefulness will still be limited.

Three requirements are central in handling the data flow from a full-scale star cluster simulation: modularity, flexibility, and compatibility. For example, to set up a major simulation, it is very useful to have a set of model building tools that are sufficiently modular, so that they can be combined in many different ways. If the data representation is flexible enough, it will be possible to add new physical variables whose use may not have been foreseen at the time that the software package was first developed. And in order for those new variables not to interfere with existing programs, compatibility is a vital issue as well.

The two main specially constructed environments in use are called *Nemo* and *starlab* (Box 2). Both consist of large collections of software and tools satisfying the above requirements. In addition a great deal of N-body work is carried out in the unix-type environments used universally by computational scientists. The principal codes used in this way are the suite of N-body programmes written by S.J. Aarseth (1985). An extremely simplified N-body code is provided in Appendix A.

Problems

1) Use either N-body code in Appendix A to investigate how the CPU time depends on the number of particles. Try to explain the dependence you find.
2) Code the N-body equations (Eq.(1.1)) using a Runge-Kutta solver, either one specially prepared for the purpose, or one drawn from any available numerical library, such as Press et al. (1992). Try to ensure that the accuracy (judged, for example, by energy conservation) is comparable with that in Appendix A. Compare the timing with that of Problem 1, and explain the difference.
3) Code the N-body equations in a symbolic computation package, such as Maple or Mathematica. Compare the timing with that in Problem 2, again with comparable accuracy, and explain the difference.

Box 2. *Starlab*

Starlab is a software package for simulating the evolution of dense stellar systems and analyzing the resultant data. It is a collection of loosely coupled programs ("tools") linked at the level of the UNIX operating system. The tools share a common data structure and can be combined in arbitrarily complex ways to study the dynamics of star clusters and galactic nuclei.

Starlab features the following basic modules:

- Three- and four-body automated scattering packages, constructed around a time-symmetrized Hermite integration scheme.
- A collection of initialization and analysis routines for use with general N-body systems.
- A general Kepler package for manipulation of two-body orbits.
- N-body integrators incorporating both 2nd-order leapfrog and 4th-order Hermite integration algorithms.
- *Kira*, a general N-body integrator incorporating recursive coordinate transformations, allowing uniform treatment of hierarchical systems of arbitrary complexity within a general N-body framework.

In addition, starlab enables the use of stellar evolution packages such as *SeBa*, which models the evolution of any star or binary from arbitrary starting conditions.

A novel aspect of Starlab is its very flexible external data representation, which guarantees that tools can be combined in arbitrary ways, without loss of data or internally-generated comments. Thus, two tools connected by UNIX pipes may operate on different portions of the same data set, even though neither understands the data structures, or even the physical variables, used by the other. Unknown data are simply passed through unchanged to the next tool in the chain.

Individual Starlab modules may be linked in the "traditional" way, as function calls to C++ (the language in which most of the package is written), C, or FORTRAN routines, or at a much higher level–as individual programs connected by UNIX pipes. The former linkage is more efficient, and allows finer control of the package's capabilities; however, the latter provides a quick and compact way of running test simulations and managing production runs. The combination affords great flexibility to Starlab, allowing it to be used by both the novice and the expert programmer with equal ease.

Box 2 (continued)

To some extent, Starlab is modeled on NEMO, a stellar dynamics software environment developed during the 1980s at the Institute for Advanced Study, Princeton, in large part by Josh Barnes, with input from Peter Teuben and Piet Hut (and subsequently maintained and extended by Peter Teuben). Starlab differs from NEMO mainly in its use of UNIX pipes, rather than temporary files, its use of tree structures rather than arrays to represent N-body systems, and its guarantee of data conservation– data which are not understood by a given module are simply passed on to the next rather than filtered out and lost. The original version of Starlab was written by Piet Hut in 1989, while on sabbatical at Tokyo University. From 1993 onwards, Steve McMillan has extended starlab, with help from Piet Hut, Jun Makino, Simon Portegies Zwart and Peter Teuben. Visualisation tools, such as *partiview*, have been added by Stuart Levy. Nemo, starlab and partiview are all available at the web site <http://www.manybody.org>.

Recently, the concept of a 'virtual observatory' has been the topic of several workshops and conferences. The idea is to connect the major observational archives, from radio to optical to X-ray observations, to make available a 'digital sky' online. Archives of large-scale simulations, such as those provided by Starlab, will be connected with those virtual observatories as well, facilitating comparisons between observations and simulations (Teuben et al. 2001).