N-Body Simulation on Hybrid Architectures

P.M.A. Sloot, P.F. Spinnato, and G.D. van Albada

Section Computational Science
Universiteit van Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{sloot, piero, dick}@science.uva.nl
http://www.science.uva.nl/research/scs/index.html

Abstract. N-body codes are routinely used for simulation studies of physical systems, e.g. in the fields of computational astrophysics and molecular dynamics. Typically, they require only a moderate amount of run-time memory, but are very demanding in computational power. A detailed analysis of an 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 optimisations, is essential in improving such codes. The approach of developing a dedicated device, GRAPE [9], able to provide a very high performance for the most expensive computational task of this code, has resulted in a dramatic performance leap. We explore on the performance of different versions of parallel N-body codes, where both software and hardware improvements are introduced. The use of GRAPE as a 'force computation accelerator' in a parallel computer architecture, can be seen as an example of a Hybrid Architecture, where Special Purpose Device boards help a general purpose (multi)computer to reach a very high performance[1].

1 Introduction

N-body codes are a widely used tool in various fields of computational science, from astrophysics to molecular dynamics. One important application is the simulation of the dynamics of astrophysical systems, such as globular clusters, and galactic clusters [10]. The core of an N-body code is the computation of the (gravitational) interactions between all pairs of particles that constitute the system. Many algorithms have been developed to compute (approximate) gravity interactions between a given particle i and the rest of the system. Our research is concerned with the simplest and most rigorous method [1], which computes the exact value of the gravity force that every other particle exerts on i. Unlike the well-known hierarchical methods [3,4], this method retains full accuracy, but it implies a computational load that grows as N^2, where N is the total number of particles. Consequently, the computational cost becomes excessive even for a few thousand particles, making parallelisation attractive [15,16].

[1] Parts of this work will be reported in the FGCS journal issue on HPCN 2000 [13].

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 883–892, 2001.

The huge computational requirements of N-body codes also make the design and implementation of special hardware worthwhile. The goal of our research is the study of an emergent approach in this field: the use of Hybrid Computer Architectures. A hybrid architecture is a parallel general purpose computer, connected to a number of Special Purpose Devices (SPDs), that accelerate a given class of computations. An example of this model is presented in [11]. We have evaluated the performance of such a system: two GRAPE boards attached to our local cluster of a distributed multiprocessor system [2]. The GRAPE SPD [9] is specialised in the computation of the inverse square force law, governing both gravitational and electrostatic interactions:

$$\mathbf{F}_i = G\frac{m_j m_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}(\mathbf{r}_j - \mathbf{r}_i) \tag{1}$$

(where m_i and m_j are star masses in the gravity force case, and charge values, in the Coulomb force case). The performance of a single GRAPE board can reach 30 GigaFlop/s. Though some fundamental differences, like electrostatic shielding, exist, this similarity in the force expression allows us in principle to use GRAPE for both classes of problems. Simulated gravothermal oscillations of globular clusters cores, and other remarkable results obtained by using GRAPE, are reported in [9].

Our research aims at understanding how hybrid architectures interact with a given application. For this purpose, we have used NBODY1 [1] as a reference code. It is a widely used code in the field of Computational Astrophysics. It includes all the relevant functionalities of a generic N-body code, without becoming overly complex. We have determined the scaling properties of various parallel versions of the code, with and without use of GRAPE boards. The data obtained are used for the realisation of a performance simulation model that will be applied to study a more general class of hybrid architectures and their interaction with various types of N-body codes [12].

2 Architecture Description

The GRAPE-4 SPD is an extremely powerful tool for the computation of interactions that are a function of r^{-2}. Given a force law like (1), the main task of a GRAPE board is to evaluate the force that a given set of particles, the j-particles, exerts on the so called i-particles. This is done in a fully hardwired way, using an array of pipelines (up to 96 per board). Each pipeline performs, for each clock-cycle, the computation of the interaction between a pair of particles.

A GRAPE-4 system consisting of 36 boards was the first computer to reach the TeraFlop/s peak-speed [9]. GRAPE-4 is suitable for systems of up to $10^4 - 10^5$ particles, when running an N-body code whose computational complexity scales as N^2 ([2]). More sophisticated algorithms, such as the Barnes and Hut tree-code,

[2] Besides the $O(N^2)$ complexity due to force computation, another term due to the relaxation time of the system must be accounted for. This makes the total time complexity of a simulation run $\sim O(N^{8/3})$ for homogeneous systems.

reduce the computing cost to $O(N \cdot \log N)$, at the price of a decreased accuracy, and an increased code complexity [3,16]. The latter codes change the work distribution between the GRAPE and the host, since many more computations not related to mere particle-particle force interactions must be done by the host. This can make the host become the system's bottleneck. This problem may be solved by using a high performance parallel machine as the host, leading to hybrid architectures.

We connected two GRAPE boards to two nodes of our local DAS (Distributed ASCI Supercomputer [2], unrelated to, and predating the American 'ASCI' machines) cluster. The DAS is a wide-area computer resource. It consists of four clusters in various locations across the Netherlands (one cluster is in Delft, one in Leiden, and two in Amsterdam). The entire system includes 200 computing nodes. A 6 Mbit/s ATM link connects remote clusters. The main technical characteristics of our DAS-GRAPE architecture are summarised in the table below:

local network	**host**	**GRAPE**	**channel**
Myrinet	PentiumPro 200 MHz	300 MFlop/s/pipe peak	PCI9080
150 MB/s peak-perf.	64 MB RAM	62, resp. 94 pipes per board	33 MHz clock
40 μs latency	2.5 GB disk	on-board memory for 44 000 j-particles	133 MB/s

3 Code Description

We chose NBODY1 as the application code for our performance analysis work because it is a rather simple code, but includes all the main tasks which GRAPE has been designed to service. 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 GRAPE's functionalities. We have built and tested the following versions of the code:

- BLK - a basic parallel version of NBODY1, enhanced by adding a block time-step scheme. This code does not make use of GRAPE.
- GRP - like BLK, but now using the GRAPE for the force calculations.

The codes were parallelised using MPI. They are described in more detail below. The basic program flow for an N-body code is given in fig. 1.

3.1 BLK: The BLOCK Code

The original version of NBODY1 uses individual time-steps. Each particle is assigned a different time at which force will be computed. The time-step value Δt depends on the particle's dynamics [1]. 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). At each iteration,

the code selects the particle that has the smallest $t + \Delta t$ value, and integrates only the orbit of that particle. This reduces the computational complexity, with respect to a code where a unique global time step is used. The individual time step approach reduces the overall time complexity to $O(N^{7/3})$, from the $O(N^{8/3})$ for global time step approach [7]. ([3]). 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 its last orbit integration. This means that an extrapolation of the other particles' positions to time t_i is needed, before the force on i is computed.

```
t = 0
while (t < t_end)
   find new i-particles
   t = t_i + Δt_i
   extrapolate particle positions
   compute forces
   integrate i-particle orbits
```

Fig. 1. Pseudocode sketching the basic NBODY1 tasks.

Since their introduction, N-body codes have evolved to newer versions, that include several refinements and improvements (*cf.* [15]). In the version of NBODY1 used in our study we implemented the so called *hierarchical block time step* scheme [8]. In this case, after computing the new Δt_i, the value actually used is the value of the largest power of 2 smaller than Δt_i. This allows more than one particle to have the same Δt, which makes it possible to have many i-particles per time step, 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 force-exerting particles, hereafter called j-particles. Moreover, 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 particle concurrently.

Parallelisation. We let every PE have a local copy of all particle data (see fig. 2 for a pseudocode sketch). Each PE computes force contributions only from its own subset of j-particles, assigned to it during initialisation. A global reduction operation adds 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. To select the i-particles, each PE searches among only its j-particles, to determine a set of i-particle candidates. 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.

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

```
t = 0
while (t < t_end)
   find i-particle candidates among my j-particles
  global reduction to determine actual i-particles
  global communication to scatter i-particles
   t = t_i + Δt_i
   extrapolate particle positions
   compute partial forces from my j-particles
  global sum of partial force values
   integrate i-particle orbits
```

Fig. 2. Pseudocode sketching the tasks of a process of the parallel NBODY1 code.

3.2 GRP: The GRAPE Code

The API 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. These communication operations include: sending j-particle data to GRAPE, sending i-particle data to GRAPE, receiving results from GRAPE.

Parallelisation. The presence of the GRAPE boards introduces a certain degree of complexity in view of code parallelisation. The GRAPE-hosts obviously play a special role within the PE 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 longer symmetric. The force computation by exploiting GRAPE boards is done, similarly to the non-GRAPE code, by assigning an equal number of j-particles to each GRAPE, which will compute the partial force on the i-particle set, exerted by its own j-particles. After that, a global sum on the partial results, done by the parallel host machine will finally give the total force. The GRAPE does not automatically update the j-particles' values, when they change according to the system evolution. The GRAPE-host must take care of this task. Each GRAPE-host holds an 'image' of the j-particles set of the GRAPE board linked to it, in order to keep track of such update. Since all force computations and j-particles positions extrapolations are done on the GRAPE, the only relevant work to do in parallel by the PE set, is the search for i-particles candidates, which is accomplished exactly as in the code described in the previous sub-section, the position extrapolation of the $i$particles, and the orbit integration.

4 Results

Measurements for the evaluation of performance of the codes described in the previous section were carried out. They were intended to explore the scaling

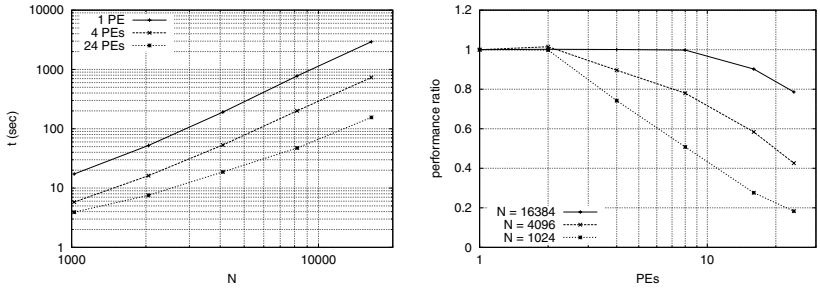

Fig. 3. *a*: Global timings for the parallel block time-step code. *b*: Parallel efficiency of the code.

behaviour of parallel N-body codes. Sample runs were made scaling both N, and the number of PEs n_{PE}; the former from 1024 to 16384, the latter from 1 to 24. NBODY1 does not need much run-time memory, just about 200 bytes per particle, but is heavily compute-bound [5]. Our timings were carried out in order to show the relative computational relevance of the various code tasks as a function of N and n_{PE}.

Our runs were started using a Plummer model distribution as initial condition (density of particles decreasing outward as a 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, inserted in the denominator in eq. (1). It reduces the strength of the force in case of close encounters and thus prevents the formation of tightly-bound binaries. In this way very short time-steps and correspondingly long integration times are avoided. The use of a softening parameter is common practice in N-body codes. 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.

4.1 Block Time-Step Code

The essential tasks of this version of the code are depicted in figure 1. As already sayed, the number of i-particles per iteration can be greater than one. 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 each time step. Fig. 3 shows total timings and performance of this code, performance being defined as:

$$P_n = \frac{t_1}{n_{PE} \cdot t_n},$$

with t_n the execution time when using n_{PE} PEs. Timings refer to 300 iterations of the code. The execution time grows as a function of N^2 because the number of

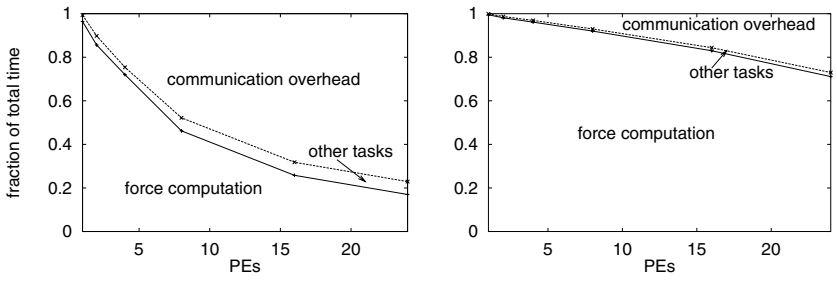

Fig. 4. Evolution of execution time shares for the BLK code. *a*: runs with 1024 particles; *b*: runs with 16384 particles.

i-particles, i.e. the number of force computations, grows approximately linearly with N ([4]). Since the computational cost for the force on each particle also grows linearly with N, the resulting total cost per time-step is $O(N^2)$. Fig. 3*b* shows a good performance gain for this code, affected anyway by a relevant communication overhead. This large overhead can be seen in Fig. 4, which also shows how the execution time shares evolve as a function of n_{PE}. These figures show that for the BLK code, almost all of the computational time is spent in the force computation task; the j-particle extrapolation, that takes roughly $25 \sim 30\%$ of the total time in the original code [13], is now reduced to a fraction of one percent.

4.2 GRAPE Code

The code-flow sketched in fig. 1 represents the actual working of GRP too. The only difference with BLK is now that forces are computed on the GRAPE, instead that on the host. We analysed the relative importance of the time spent in GRAPE computation, host computation, and mutual communication. For the parallel version, network communications overheads also have been analysed. The parallel code runs have been done by using only the DAS nodes connected to the GRAPE boards at our disposal, thus the maximum number of PEs in this case is two. We observed that the parallel performance of the GRP code is very poor. The large communication overhead that dominates the GRP code, as can be seen in fig. 5, can explain this. This figure shows, apart from the large communication overhead, 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. The GRAPE performs its task much faster than its host and the communication link between them. The figure clearly shows that for our

[4] In our runs we found that on average $2.5\% \sim 3\%$ of the particles are updated in every time-step for all values of N. This fraction may change as the system evolves.

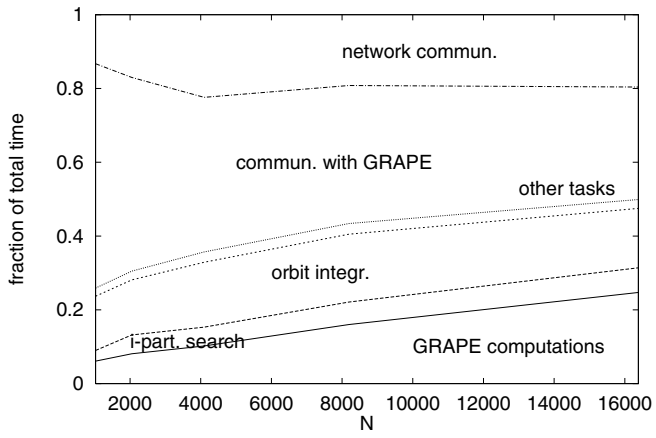

Fig. 5. Scaling of execution time shares for the parallel GRP code.

hardware configuration the capabilities of the GRAPE will only be fully utilised for problems of over 40 000 particles (for single GRAPEs) and approximately double than that for the parallel system. This number is, however, limited by the on-board memory for j-particles of GRAPE.

Measurements [14] show that most of the time spent in communication is due to software overhead in copy operations and format conversions; analogous measurements [6], performed on a faster host, showed a higher communication speed, linearly dependent on the host processor clock speed. Nevertheless, even though GRAPE boards are not exploited optimally, the execution times for the GRP code are much shorter than those for the BLK code. The heaviest run on two GRAPEs is about one order of magnitude faster than the analogous run of the BLK code on 24 PEs. A global comparison of the throughput of all codes studied in this work is given in the next subsection.

4.3 Code Comparison

In order to evaluate the relative performance of the two versions of the N-body code studied in this paper, runs were made, where a 8192 particle system, and a 32768 particle system were simulated for 7200 seconds. Initial conditions and values of numerical parameters were identical to the ones previously specified. The fastest hardware configuration was used in each case, i.e. 24 PEs for the BLK code runs, and 2 PEs (and 2 GRAPEs) for the GRP run. Figs. 6*a,b* show the evolution of the simulated time, as a function of the execution time. The performance of each code is measured as the time it takes before a simulation reaches a certain simulated time. The figures show that the GRP code outperforms the other code by a factor 8, for 8192 particles, and by a factor 20, for 32768 particles.

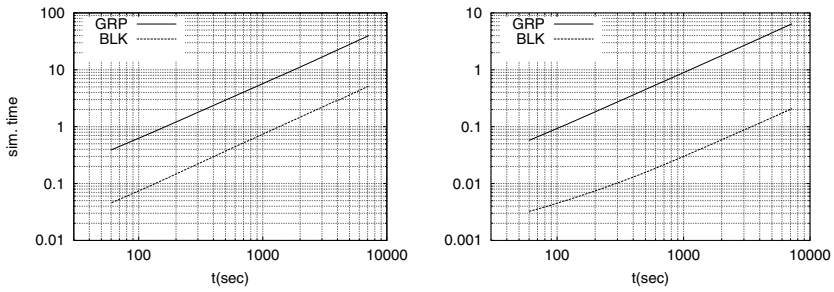

Fig. 6. Performance comparison for the two versions of the N-body code. *a*: runs with 8192 particles; *b*: runs with 32768 particles.

These figures clearly show the large performance gain obtained with GRAPE. Using only two PEs, an order of magnitude better performance was attained compared to the BLK 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. Hence an even larger relative gain may be expected for larger problems, as the relative weight of the communication overhead will become less. The difference in performance between the two cases shown in fig. 6 clearly illustrates this effect.

5 Discussion

The main conclusions from our work are, apart from the very good parallel performance of the BLK code, that the GRP code shows a dramatic performance gain, even at a low efficiency in terms of GRAPE boards utilisation. This low efficiency is mainly due to a very high communication overhead, even for the largest problem studied. This overhead can be strongly reduced with the use of a faster host, and by the development of an interface requiring fewer format conversions. The GRAPE hosts in the system that we studied have a 200 MHz clock speed. Nowadays standard clock speeds are 3 to 4 times faster; the use of a state-of-the-art processor would reduce the host and communication times significantly. An extremely powerful machine as GRAPE, in any case, can be exploited efficiently only when the problem size is large, thus attaining the highest SPD utilisation.

The measurements described in this paper have been used to validate and calibrate a performance simulation model for N-body codes on hybrid computers. The model will be used to study the effects of various software and hardware approaches to the N-body problem.

Acknowledgments. Jun Makino has kindly made two GRAPE-4 boards available to us, without which this work would have been altogether impossible.

Sverre Aarseth and Douglas Heggie are also acknowledged for having made available to us the original serial N-body codes. Discussions with Douglas Heggie were of great value at an early stage of the work related to the parallelisation of the code.

References

1. S.J. Aarseth, Direct Methods for N-body Simulations, in J.U. Brackhill and B.I. Cohen, eds., Multiple Time Scales (Academic Press, 1985)
2. H.E. Bal, R.A.F. Bhoedjang, R.F.H. Hofman, C.J.H. Jacobs, T. Kielmann, J. Maassen, R. van Nieuwpoort, J. Romein, L. Renambot, T. Rühl, R. Veldema, K. Verstoep, A. Baggio, G. Ballintijn, I. Kuz, G. Pierre, M. van Steen, A.S. Tanenbaum, G. Doornbos, D. Germans, H. Spoelder, E.-J. Baerends, S. van Gisbergen, H. Afsarmanesh, G.D. van Albada, A.S. Belloum, D. Dubbeldam, Z.W. Hendrikse, L.O. Hertzberger, A.G. Hoekstra, K.A. Iskra, B.D. Kandhai, D.C. Koelma, F. van der Linden, B.J. Overeinder, P.M.A. Sloot, P.F. Spinnato, D.H.J. Epema, A. van Gemund, P.P. Jonker, A. Radulescu, C. van Reeuwijk, H.J. Sips, P.M. Knijnenburg, M. Lew, F. Sluiter, L. Wolters, H. Blom and A. van der Steen, The Distributed ASCI supercomputer project, Operating Systems Review 34, nr 4 (2000) 76–96.
3. J. Barnes and P. Hut., A Hierarchical $O(N \cdot \log N)$ Force-Calculation Algorithm, Nature 324 (1986) 446
4. H. Cheng, L. Greengard and V. Rokhlin, A Fast Adaptive Multipole Algorithm in Three Dimensions, J. Comput. Phys. 155 (1999) 468
5. P. Hut, The Role of Binaries in the Dynamical Evolution of Globular Clusters, in E.F. Milone and J.C. Mermilliod, eds, Proc. of Int. Symp. on the Origins, Evolution, and Destinies of Binary Stars in Clusters. ASP Conf. Series 90 (1996) 391
6. A. Kawai, T. Fukushige, M. Taiji, J. Makino and D. Sugimoto, The PCI Interface for GRAPE Systems: PCI-HIB, Publ. of Astron. Soc. of Japan 49 (1997) 607
7. J. Makino and P. Hut, Performance Analysis of Direct N-body Calculations. Astrophys. J. Suppl. 68 (1988) 833
8. J. Makino, A Modified Aarseth Code for GRAPE and Vector Processors, Publ. of Astron. Soc. of Japan 43 (1991) 859
9. J. Makino and M. Taiji, Scientific Simulations with Special-Purpose Computers, Wiley, 1998
10. G. Meylan and D. C. Heggie, Internal Dynamics of Globular Clusters, Astron. and Astrophys. Rev. 8 (1997) 1
11. P. Palazzari, L. Arcipiani, M. Celino, R. Guadagni, A. Marongiu, A. Mathis, P. Novelli and V. Rosato, Heterogeneity as Key Feature of High Performance Computing: the PQE1 Prototype, in Proc. of Heterogeneous Computing Workshop 2000, Mexico Cancun (May 2000). IEEE Computer Society Press (2000), http://www.computer.org/proceedings/hcw/0556/0556toc.htm
12. P.F. Spinnato, G.D. van Albada and P.M.A. Sloot, Performance Modelling of Hybrid Architectures, Computer Physcics Communications (2001), accepted.
13. P.F. Spinnato, G.D. van Albada and P.M.A. Sloot, Performance of N-body Codes on Hybrid Machines, Future Generation Computer Systems (2001), in press.
14. P.F. Spinnato, G.D. van Albada and P.M.A. Sloot, Performance Measurements of Parallel Hybrid Architectures. Technical Report, *in preparation*
15. R. Spurzem, Direct N-body Simulations, J. Comput. Appl. Math. 109 (1999) 407
16. M.S. Warren, and J.K. Salmon, A Portable Parallel Particle Program. Comp. Phys. Comm. 87 (1995) 266