Performance Analysis of Parallel N-body Codes

P. Spinnato G.D. van Albada P.M.A. Sloot *

Faculty of Science
Section Computational Science
Universiteit van Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

piero@science.uva.nl dick@science.uva.nl sloot@science.uva.nl

Keywords: Performance, N-body codes, Hybrid Systems

Abstract

N-body codes are routinely exploited 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 such 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 computation of 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 Hybrid Architecture, where a number of Special Purpose Device boards help a general purpose (multi)computer to reach a very high performance.

1 Introduction

N-body codes are a widely used tool for the simulation of 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 which compose the system. Many algorithms have been developed to compute (approximate) gravity interactions between a given particle i and the rest of the system [2, 3, 4]. Our research is concerned with the simplest and most rigorous method [2], 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 which grows as N^2, being N the total number of particles. Consequently the computational cost becomes excessive even with a few thousands of particles, making parallelisation attractive. Recently, parallelisation of N-body codes has become an important research issue [13, 14, 15].

The huge computational requirements of N-body codes make the design and implementation of special hardware worthwhile. The goal of our research is the study of an emergent evolution in this field: Hybrid Computer Architectures. An hybrid architecture is a parallel general purpose computer, connected to a number of Special Purpose Devices (SPDs), which accelerate a given class of computations. An instantiation of this model is presented in [11]. In the light of this, we have evaluated the performance of such a system: two GRAPE boards attached to our local cluster of a distributed multiprocessor system [1]. The GRAPE SPD [9], is specialised in the computation of the inverse square law, governing both gravitational and electrostatic interactions:

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

(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. Gravothermal Oscillations of Globular Clusters cores, and other remarkable results obtained by using GRAPE, are reported in [9]. Though some fundamental differences, like electro-

* Paper appeared as: P.F. Spinnato; G.D. van Albada and P.M.A. Sloot: Performance Analysis of Parallel N-body Codes, in L.J. van Vliet; J.W.J. Heijnsdijk; T. Kielmann and P.M.W. Knijnenburg, editors, ASCI 2000, Proceedings of the sixth annual conference of the Advanced School for Computing and Imaging, pp. 213-220. ASCI, Delft, June 2000. ISBN 90-803086-5-x.

static shielding, exist, this similarity in the force expression allows us in principle to use GRAPE for both classes of problems.

Our research aims at understanding how such architectures interact with a given application. For this purpose, we have used NBODY1 [2] as a reference code. It is a widely used code in the field of Computational Astrophysics. It is rather simple, but includes all the relevant functionalities of a generic N-body code. By using NBODY1, we can determine 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 used to study a more general class of hybrid architectures and their interaction with various types of N-body codes.

2 Architecture description

The GRAPE-4 SPD is an extremely powerful tool for the computation of interactions which are a function of r^{-2}. Given a force law like (1), the main function of a GRAPE board is to output 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, by means of an array of pipelines (up to 96 per board). Each pipeline performs, at 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 ([1]). More sophisticated algorithms exist, which reduce the computing cost to $\mathcal{O}(N \cdot \log N)$, at the price of a decreased accuracy, and an increased code complexity [3, 15]. 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, and makes interesting a study of architectures where the host is a high performance parallel machine.

We connected two GRAPE boards to two nodes of our local DAS cluster. The DAS is a wide-area computer resource. It consists of four clusters placed 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 line connects remote clusters. The main technical characteristics of our DAS-GRAPE architecture are summarised in the table below:

local network	**host**
Myrinet	Pentium Pro 200 MHz
150 MB/s peak-perf.	64 MB RAM
40 μs latency	2.5 GB disk
GRAPE	**channel**
2 boards 30 GFlop/s peak	PCI9080
62 and 94 pipes per board	33 MHz clock
on-chip memory for 20.000 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 on the code, in order to parallelise it, and to let it make full use of GRAPE's functionalities. An overview on the code is given in what follows. We made use of MPI to parallelise it.

3.1 The basic: individual time-step

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 [2]. 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 that particle having 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 temporal complexity to $\mathcal{O}(N^{1/3})$, whereas the global time step approach is $\mathcal{O}(N^{2/3})$ [7] ([2]).

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 force on i is computed.

3.1.1 parallelisation

Since contributions to the gravity force on a given particle i are computed from all the other particles using eq. (1), regardless of their distances from i, an

[1] Besides the $\mathcal{O}(N^2)$ complexity due to force computation, another term due to temporal integration has to be accounted for. See discussion in next section.

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

uniform distribution of particles to each processing element (PE) suffices to assure load balancing. The force computation is done by broadcasting the coordinates of the currently selected particle i. Then each PE computes the partial component to the force on i, by accumulating contributions from its own particles. Finally such components are sent back to the PE which hosts i, where the force resultant is computed, the particle's orbit is integrated, and the new values are stored.

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

3.2 Toward a GRAPE code: block time-step

Since its introduction, NBODY1 has evolved to newer versions, which include several refinements and improvements (*cf.* [13]). 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 assigned 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 such hardware, since each pipeline can compute the force on a different particle concurrently.

3.2.1 parallelisation

Having many i-particles, instead of only one, makes attractive to use a somewhat different parallel code structure. If the i-particles reside on different processors, distributing the particles as in the individual time-step case could cause too convoluted communication patterns, with consequential increase of code complexity. Therefore, we chose to let every PE have a local copy of all particle data. The force computation is done in parallel by making each PE compute force contributions only from its own set of j-particles, assigned to it during initialisation. A global reduction operation 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. For what concerns the search for i-particles, each PE searches among only its j-particles,

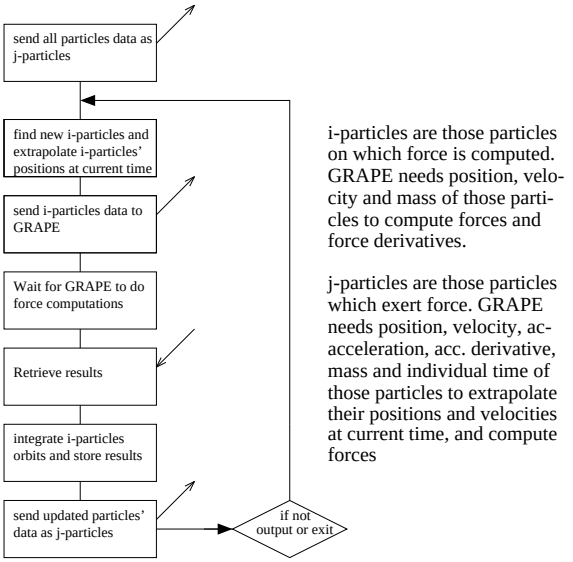


Figure 1: Basic sketch of NBODY1 tasks. Diagonal arrows symbolise communication with GRAPE.

to determine a set of i-particles candidates. Then a global reduction operation is performed on the union of such 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.

3.3 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. Such communication operations are: sending j-particle data to GRAPE, sending i-particle data to GRAPE, receiving results from GRAPE. A sketch of the program flow for an N-body code which uses GRAPE is given in fig. 1.

3.3.1 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 PEs set. This asymmetry somehow breaks the SPMD paradigm which parallel MPI programs are expected to comply with. Besides the asymmetry in the code structure, also the data distribution among PEs is no more symmetric. The force computation 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 PEs set, is the search for i-particles candidates, which is accomplished exactly as in the code described in the previous subsection.

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

Our runs were started having 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, which is 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 Individual Time-step Code

The essential tasks of this version of the code (hereafter called IND) are basically the same as in the code-flow depicted in figure 1. That case refers to the code which makes use of GRAPE; in the present case no communications with the GRAPE device are done.

As described in the previous section, the parallel version of this code implements communications in the task regarding the i-particle search, and when i-particle's position is broadcast, and partial forces are gathered by the PE that owns the i-particle. Figure 2 shows the timings and the related performance of the parallel version of the IND code. Performance is defined as:

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

where n is the number of PEs used, and t_n the execution time when using n PEs. Timings refer to 1000 iterations of the code. Their dependence is linear with respect to N, since the number of operations to compute the force on a given particle scales linearly with N, and in each run the same number of force computations is performed, i.e. 1000, independently of the total number of particles. An interesting super-linear speedup is visible in fig. $2b$, arguably due to an optimised cache utilisation. This figure also clearly shows how this code suffers of a communication overhead when the computational work-load is light, i.e. for low values of the N/PEs ratio, but performs quite satisfactorily when this ratio is high, thanks to the compute-intense characteristics of the N-body code, and the high performance communication network of our architecture.

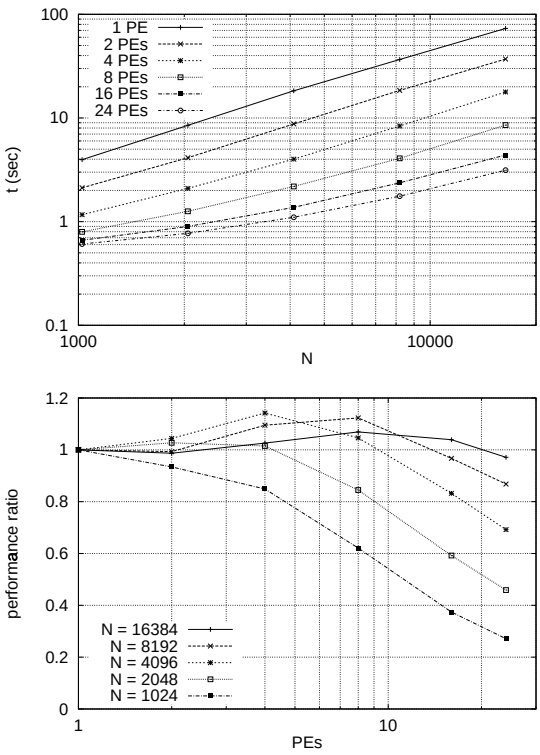


Figure 2: a: Global timings for the parallel individual time-step code, running for 1000 iterations. b: Performance of the code.

4.2 Block Time-step Code

The basic tasks of this version of the code (BLOCK hereafter) are the same as those described

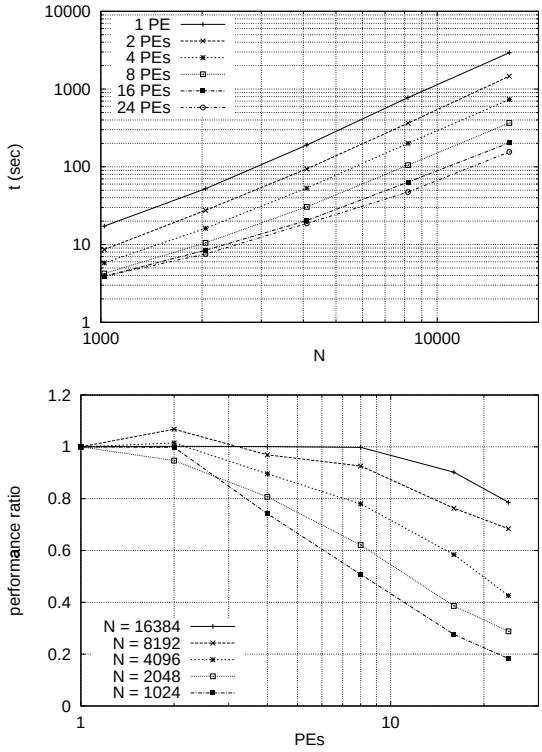
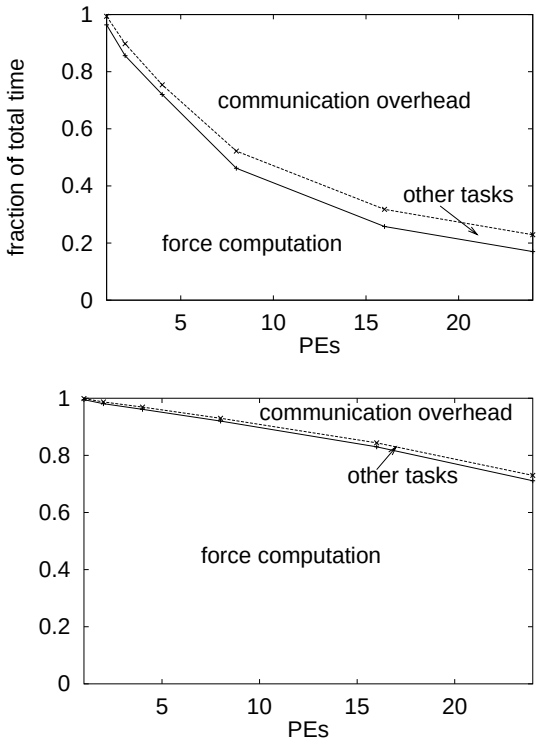


Figure 3: a: Global timings for the parallel block time-step code. Here force on many particles is computed at each time-step b: Performance of the code.



N	$< N_i >$
1024	35.05
2048	43.91
4096	111.16
8192	207.52
16384	351.14

Figure 4: Evolution of execution time shares. a: runs with 1024 particles; b: runs with 16384 particles; c: Mean number of i-particles (i.e. of force computations) per iteration in the runs of the BLOCK code.

for the IND code. The only difference is that now the number of i-particles per iteration can be greater than 1. As stated, 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.

The effect of this is clearly shown in the figures presented here. Fig. 3 shows total timings and performance of this code; in this case the execution time grows as a function of N^2 because the number of i-particles, i.e. the number of force computations, grows approximately linearly with N. Since the computational cost for the force on each particle also grows linearly with N, the resulting total cost is $\mathcal{O}(N^2)$. A table listing the mean number of force computations per iteration, to show such linear scaling with N, is given in fig. 4*c*. Fig. 3*b* shows how the performance gain of this code is less spectacular than the gain of the IND code, since communication overhead plays a larger role in the total execution time. This large overhead can be seen in figures 4*a*,*b*, which also show how the execution time shares evolve as a function of PEs number. These figures show that for the BLOCK code, almost all the computational part of the execution time is spent in the force computation task; the j-particles extrapolation, that takes roughly $25 \sim 30\%$ of the total time in the IND code (data not shown), here is reduced to a fraction of one percent.

4.3 GRAPE Code

The code version which makes use of GRAPE boards will be called GRP hereafter. A code-flow of the serial version of GRP is sketched in fig. 1. The communication overhead of the parallel version now includes also network communications. 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 2.

It is clear from fig. 5 that the parallel performance is very poor. The large communication overhead, which dominates the GRP code as can be seen in fig. 6, can explain this. Here, GRAPE0 refers to

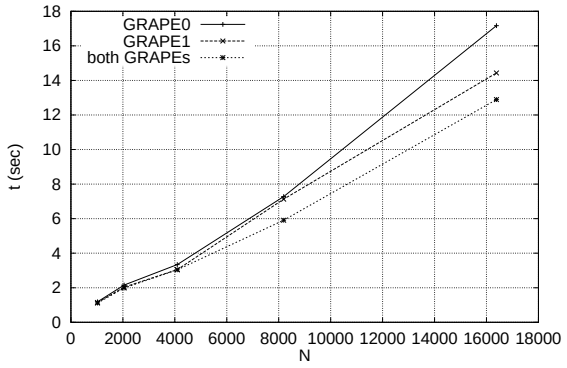


Figure 5: Execution time for the GRP code.

the GRAPE with 62 pipelines, and GRAPE1 to the GRAPE with 94 pipelines. Figure 5 shows that runs on GRAPE1 are a bit faster, thanks to the larger number of pipelines available. The other figure shows that, apart from the large communication overhead, the time share spent in GRAPE computations (i.e. force computations) is quite low, resulting in a low efficiency of this code, in terms of GRAPE exploitation. One reason for that is of course the very high speed of the GRAPE. This device is by far faster in accomplishing its task than its host and the communication link between them. The figures clearly show that for our hardware configuration the capabilities of the GRAPE will only be fully utilised for problems of over 40000 particles (for single GRAPEs) and approximately double than that for the parallel system. This number is, however, limited by the onboard memory for j-particles of GRAPE.

Measurements [12] 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 by far shorter than those for the BLOCK code. The heaviest run on 2 GRAPEs is about one order of magnitude faster than the analogous run of the BLOCK code on 24 PEs. A global comparison of the throughput of all codes studied in this work is given in the next subsection.

4.4 Codes Comparison

In order to evaluate the relative performance of the three versions of the N-body code studied in this work, a series of runs has been made, where both a 8192 particles system, and a 32768 particles system were simulated for 7200 seconds. This will illustrate our expected better scaling of the GRP code, with

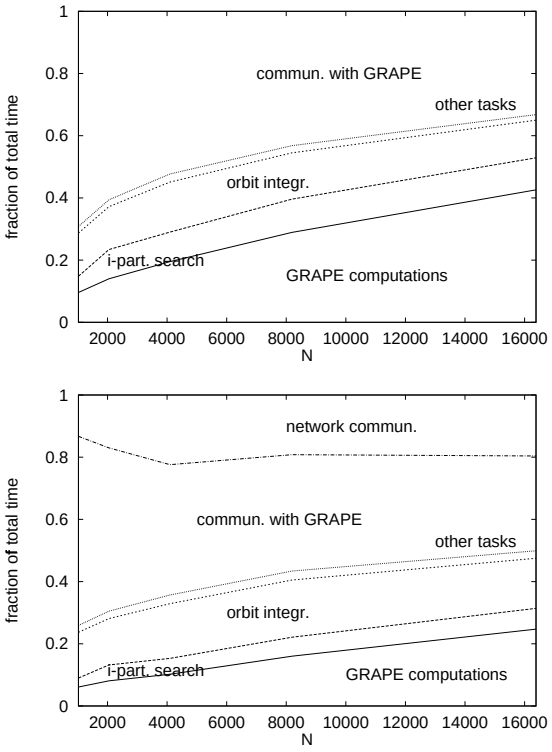


Figure 6: Evolution of execution time shares. a: runs on GRAPE0; b: runs on both GRAPEs. Data for GRAPE1 (not shown) are qualitatively very similar to GRAPE0 data.

respect to an increasing computational load. 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 IND and BLOCK code runs, and 2 PEs (and 2 GRAPEs) for the GRP run. Figs. 7*a*,*b* show the evolution of the simulated time, as a function of the execution time. In such a way, the performance of each code is made clear in terms of how long one should wait before a simulation reaches a certain simulated time. The figures show that the GRP code outperforms the other two codes by a factor 8, when the computational load is lighter, and by a factor 20, with a heavier computational load. In both cases the BLOCK code is 1.5 times faster than the IND code, thanks to the optimisation of the j-particles extrapolation step. Fig. 7*b* shows an initial overlapping of these two codes performance curves, due to a start-up phase, which is not visible in fig. 7*a*, because at the first timing event (after $60\,s$) this system is already stabilised.

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 BLOCK code on 24 PEs. Due to the reduction in the time needed for the force calcula-

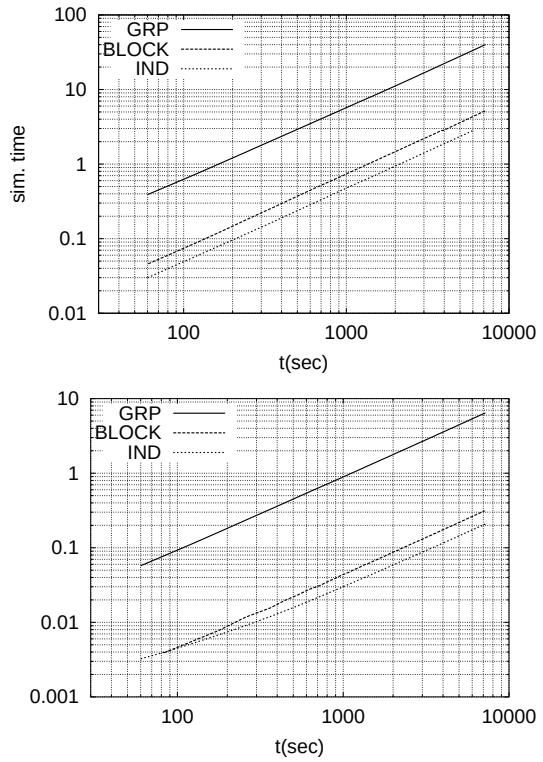


Figure 7: Performance comparison for the three versions of the N-body code. a: runs with 8192 particles; b: runs with 32768 particles.

tion, the communication overhead for the GRP code accounts for approximately 50% of the total execution time (*cf.* fig. 6). 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. 7 clearly illustrates this effect.

5 Discussion

The main conclusions from our work are, apart from the very good parallel performance of the BLOCK and especially the IND code, that the GRP code shows a dramatic performance gain, even at a low efficiency in terms of GRAPE boards exploitation. Such low efficiency is mainly due to a very high communication overhead, even for the largest problem studied. This overhead can be 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 efficiently exploited only when the problem size remarkably increases, hence 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.

Acknowledgements

The work related to the parallelisation of the IND code was done at the Edinburgh Parallel Computing Centre (EPCC) under the TRACS programme, supported by the EU. EPCC and TRACS are warmly acknowledged for their support. 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 unvaluable for the work related to the parallelisation of the IND code. Jun Makino has kindly made two GRAPE-4 boards available to us, without which this work would have been altogether impossible.

References

[1] `http://www.cs.vu.nl/das/`

[2] Aarseth, S.J.: Direct Methods for N-body Simulations. In Brackhill, J.U., & Cohen, B.I. (eds.): Multiple Time Scales. Academic Press (1985)

[3] Barnes, J. & Hut., P.: A Hierarchical $\mathcal{O}(N \cdot \log N)$ Force-Calculation Algorithm. Nature **324** (1986) 446

[4] Cheng, H., Greengard, L., & Rokhlin, V.: A Fast Adaptive Multipole Algorithm in Three Dimensions. J. Comput. Phys. **155** (1999) 468

[5] Hut, P.: The Role of Binaries in the Dynamical Evolution of Globular Clusters. In Milone, E.F., & Mermilliod, J.-C. (eds.): proc. of Int. Symp. on the Origins, Evolution, and Destinies of Binary Stars in Clusters. ASP Conf. Series **90** (1996) 391 (*also available at:* `astro-ph/9602158`)

[6] Kawai, A., *et alii*: The PCI Interface for GRAPE Systems: PCI-HIB. Publ. of Astron. Soc. of Japan **49** (1997) 607

[7] Makino, J., & Hut., P.: Performance Analysis of Direct N-body Calculations. Astrophys. J. Suppl. **68** (1988) 833

[8] Makino, J.: A Modified Aarseth Code for GRAPE and Vector Processors. Publ. of Astron. Soc. of Japan **43** (1991) 859

[9] Makino, J., & Taiji, M.: Scientific Simulations with Special-Purpose Computers. Wiley (1998)

[10] Meylan, G. & Heggie, D. C.: Internal Dynamics of Globular Clusters. Astron. and Astrophys. Rev. **8** (1997) 1

[11] Palazzari, P., *et alii*: Heterogeneity as Key Feature of High Performance Computing: the PQE1 Prototype. To appear in Proc. of Heterogeneous Computing Workshop 2000, Cancun, Mexico (May 2000). IEEE Computer Society Press (2000)

[12] Spinnato, P., van Albada, G.D., and Sloot, P.M.A.: Performance Measurements of Parallel Hybrid Architectures. Technical Report, *in preparation*

[13] Spurzem, R.: Direct N-body Simulations, J. Comput. Appl. Math. **109** (1999) 407

[14] Sweatman, W. L.: The Development of a Parallel N-body Code for the Edinburgh Concurrent Supercomputer. J. Comput. Phys. **111** (1994) 110

[15] Warren, M.S., & Salmon, J.K.: A Portable Parallel Particle Program. Comp. Phys. Comm. **87** (1995) 266