A Versatile Simulation Model for Hierarchical Treecodes

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

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

Abstract. We present here a performance model which simulates different versions of the hierarchical treecode on different computer architectures, including hybrid architectures, where a parallel distributed general purpose host is connected to special purpose devices that accelerate specific compute-intense tasks. We focus on the inverse square force computation task, and study the interaction of the treecode with hybrid architectures including the GRAPE boards specialised in the gravity force computation. We validate the accuracy and versatility of our model by simulating existing configurations reported in the literature, and use it to forecast the performance of other architectures, in order to assess the optimal hardware-software configuration.

1 Introduction

The problem of computing the force interactions in a system of N mutually interacting particles is one of the important challenges to Computational Science. The gravitational interactions among stars in a stellar globular cluster, or the electrostatic interactions among ions in a chemical solution are typical instances of such a problem, generally known as the N-body problem. As interactions are long ranged, the N-body problem scales as $\mathcal{O}(N^2)$, leading to an unsustainable computational load for realistic values of N. Special software and hardware tools have been developed in order to make the simulation of realistic N-body systems feasible.

Two of the most important contributions to this research have been the introduction of the hierarchical treecodes [1], and the realisation of the GRAPE class of Special Purpose Devices (SPDs) [2]. Such approaches aim at reducing the time spent in computing the forces. The treecode reaches this goal by computing partial forces on a given particle from a truncated multipole expansion of groups of particles, instead of adding each single force contribution. Groups become larger and larger as their distance from the given particle increases. This technique allows to decrease the computing time of the force evaluation to

$\mathcal{O}(N \log N)$, at the cost of a reduced accuracy, due to the truncated multipole expansion. The highest multipole order is usually the quadrupole.

The GRAPE is a very fast SPD, containing an array of pipelines, each one hardwiring the operations needed to compute a gravitational (or electrostatic) interaction. The latest development in the GRAPE series, GRAPE-6, will contain more than 18 000 hierarchically organised pipelines in its final configuration, for an aggregated peak performance of about 100 TeraFlop/s [3]. A computing environment able to take advantage of both approaches could provide a further boost to N-body computation. Research in the direction of merging these two approaches has been carried out for about a decade [4, 5], but the attained performance has been limited by several factors, the most important being related to the difficulty of using GRAPEs efficiently in a distributed architecture. When selected nodes of a distributed architecture are GRAPE hosts, and GRAPEs are used to compute forces also for particles stored in remote nodes, a huge amount of communication is necessary to perform this task, leading to a very high communication overhead.

In this paper, we explore the possibility of using a parallel host in order to improve the performance of the system. Efficient parallelisation of the treecode is not straightforward, and much efforts have been devoted to this task [6–8]. Coupling a parallel host with a set of GRAPE boards introduces a further complication, as said above, which makes it difficult to estimate the optimal configuration of the resulting hybrid architecture. In order to understand the complex interplay of the parallel algorithm, the parallel general purpose host, and the multiple board SPD, we developed a performance model, by means of which we can predict the performance of real systems [9, 10]. Using this tool, the task of designing the optimal computing environment for the integration of N-body systems can be made less difficult. A detailed discussion on our performance modelling approach is given in [9].

In the following sections, we present our model, the hardware, and the software components of the modelled environment. Then we show the modelling of configurations reported in the literature, which validate our model. Finally we show the results concerning the model of a configuration that integrate GRAPE boards in a distributed parallel architecture.

2 The performance model

2.1 Modelling approach

Our modelling approach can be sketched as a so-called Y-chart, represented in fig. 1. We define an *application model*, where each task of the software application is described in terms of the operations performed, and the workload that such operations produce. Workload is expressed as a function of the application parameters $\mathcal{P} = \{\pi_1, \pi_2, \dots\}$. Our application is here the treecode; we will give below in table 1 the modelling expressions of each task of the treecode, as a function of its application parameters.

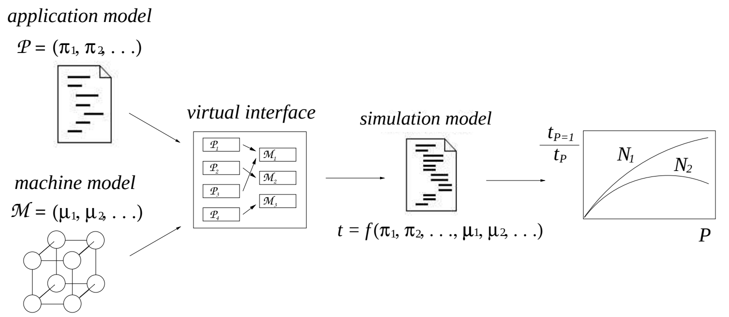

Fig. 1. Performance modelling process. The output of the simulation model, t, is the modelled execution time. The rightmost sketch represents a speedup graph (P is the number of processors), showing the behaviour produced by two different values assigned to a certain parameter N. The meaning of the other symbols is explained in the text.

In the *machine model*, each architecture resource is specified in terms of the time spent in accomplishing the task it is designed to perform, as a function of the machine parameters $\mathcal{M} = \{\mu_1, \mu_2, \ldots\}$. Such quantities are basic performance parameters, such as clock frequency and communication bandwidth.

A *virtual interface* maps each task of the application model to the appropriate machine resource; possible resource contention is managed and resolved by the performance modelling environment. The result of the task mapping is the *simulation model*, which is the actual simulator of the whole system. It simulates an instantiation of the treecode, running on an instantiation of a computer architecture. According to the input parameters, it can simulate different treecode implementations, running on different architectures, including SPDs. The simulation model outputs the execution time of the application, the utilisation of the various hardware components, and other performance measures.

Our performance model is implemented in Pamela (PerformAnce ModEling LAnguage) [11]. Pamela is a C-style procedure-oriented simulation language in which a number of operators model the basic features of a set of concurrent processes. A detailed overview on Pamela is given in [11].

2.2 Model components

Software application The hierarchical treecode [1] 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 with lower accuracy. We give now a short description of the main treecode procedures. A simple pseudo-code sketching the basic tasks of the treecode is given in fig. 2.

The treecode approach for computing forces on a given particle ξ is to group particles in larger and larger cells as their distance from ξ increases, and compute force contributions from such 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 into the same cube, such cube is divided into eight 'child' cubes, whose linear size in one half of their parent's. This procedure is repeated until the two particles find themselves into different cubes. Hierarchically connecting such cubical cells according to their parental relation leads to a hierarchical tree data structure. When force on particle ξ is computed, the tree is traversed looking for cells that satisfy an appropriate criterion (see [12] for a detailed overview on acceptance criterions). By applying this procedure recursively starting from the tree root, i.e. the cell containing the whole system, all the cells satisfying the acceptance criterion are found.

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

Fig. 2. Pseudocode sketching the basic treecode tasks.

The original treecode algorithm has been modified in several ways, in order to improve performance. The tree traversal phase has been optimised by performing a single traversal for a group of nearby particles [13], whereas the original algorithm performs a tree traversal for each particle. This drastically reduces the number of tree traversals, and allows concurrent force computation on vector machines. It is also ideal when the treecode is used with GRAPE, because each pipeline of the array contained in a GRAPE board can compute force on a different particle simultaneously. The drawback of this technique is an increase in memory usage, because for each group an interaction list containing all the cells interacting with the group must be written and stored in memory.

The use of interaction lists is also useful for parallelisation on distributed systems. The possibility of decoupling tree traversal and force computation through interaction list compilation, allows the implementation of latency hiding algorithms for the retrieval of cell information stored in a remote processor memory [8]. We will refer to this version of the parallel treecode as HOT (HOT is the acronym of Hashed Oct-Tree, which is the name given to the code by its authors).

Another modification consists in computing force only on a small fraction of the N particles at each code iteration, a criterion originally introduced in the direct N-body code (*cf.* [14]). In this case, each particle is assigned an individual time-step, and at each iteration only those particles having an update time below a certain time are selected for force evaluation [7]. In this code, a different approach for remote interactions computation is also implemented: data of the selected particles are sent to the remote processors, interactions are computed remotely, and results are received back. A further modification consists in rebuilding the local tree less frequently than at every iteration. This version

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

Fig. 3. Pseudocode sketching the generic parallel treecode tasks. HOT and GDT are the two versions of the treecode modelled in this work (*cf.* sec. 2.2). Tasks involving communication are highlighted.

will be referred as GDT, which is a short for GADGET, the name given to the code by its authors. In fig. 3 we give a pseudo-code representation of the generic algorithm that our model simulates.

Application model Many other versions of the treecode have been proposed, implementing different tools and techniques. A recent report on that is given in [7]. Our performance model has been designed to reproduce the behaviour of state-of-the-art parallel treecodes, running on distributed architectures, and able to make use of dedicated hardware. Table 1 shows a synopsis of the modelling expressions of the application tasks described above, given as functions of the relevant application parameters.

Computer architecture The parallel system simulated in our machine model is a generic distributed multicomputer, where given nodes can be connected to one or more SPDs. When SPDs are present, the appropriate task is executed on them. The application model needs no modification in this case. According to an input parameter which tells whether SPDs are present, the mapping interface chooses the routine that maps the task to the SPD, or to the general purpose processor. Since we are interested in SPDs dedicated to the gravity force computation, the machine model of the SPD reproduces the GRAPE activity, and its communication with the host. The modelling of the fairly complicated data exchange machinery between GRAPE and its host is discussed in [10].

Table 1. Synopsis of the modelling expressions for each task of the application model. All *cp* terms are constant factors proportional to the operations per particle performed. *cm* terms are proportional to the amount of bytes per particle transmitted. The index of a *cp* or *cm* term refers to the task performed.

task	**parameter description**	**modelling expression**
build local tree	n: number of particles per processor n_c: number of cells per processor n_{mp}: operations per cells to compute multipoles	$cp_{bt} \cdot n \cdot \log n +$ $cp_{mp} \cdot n_c \cdot n_{mp}$
data exchange for HOT global tree	N_P: number of processors	$cm_{gt} \cdot N_P$
build local lists	m: fraction of particles selected for force computation (= 1 if code is HOT) n_g: number of groups per processor n_c: number of cells per processor	$cp_{ll} \cdot m \cdot n_g \cdot n_c$
data exchange for HOT global lists	j_{rmt}: number of sources from remote processors n_g: number of groups per processor	$cm_{gl} \cdot j_{rmt} \cdot n_g$
compute forces	m: fraction of particles selected for force computation n: number of particles per processor j: total number of force sources	$cp_{fc} \cdot m \cdot n \cdot j$
data exchange for GDT remote force	m: fraction of particles selected for force computation N: total number of particles N_P: number of processors	$cm_{rf} \cdot m \cdot N \cdot \log N_P$
integrate orbits	n: number of particles per processor	$cp_{or} \cdot n$

The hardware characteristics of the simulated multicomputer are parameterised by two constants, τ_p and τ_n. The first quantity, τ_p, accounts for the processor speed, its value being the amount of floating points operations per nanoseconds; τ_n accounts for the network speed, and its value is the transfer rate in $\mu s/B$. In the execution model, each computation-related function (those containing a *cp* constant in the modelling expressions reported in table 1) will be multiplied by τ_p, each communication-related function (those containing a *cm* constant) will be multiplied by τ_n.

3 Results

3.1 Model validation

We present here the result of running our simulation model with modelling parameters such that performance measurements reported in the literature are

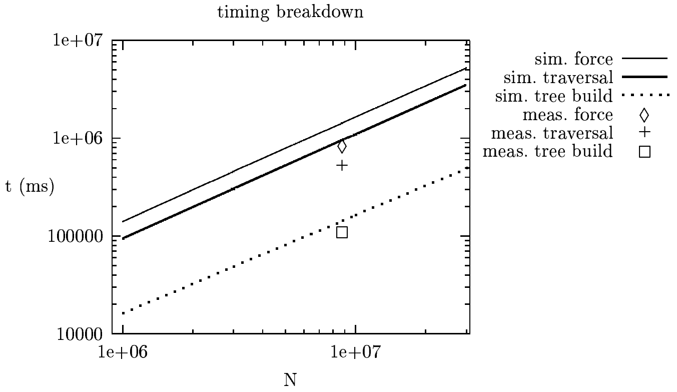

Fig. 4. Timings of the HOT tasks. The real system timings are also reported. The hardware architecture is the Intel Touchstone Delta.

reproduced. We show for each case the scaling with the total particle number N of each task of the code, compared with the corresponding real system timings, as reported by the measurements authors. Finally we present a plot comparing the total compute time for a code iteration of each configuration.

HOT on Touchstone Delta The Touchstone Delta was a one-of-a-kind machine installed at Caltech in the early nineties. It consisted of 512 i860 computing nodes running at 40 MHz, and connected by a 20 MB/s network. The performance measurements reported in [8] are based on a run using the whole 512 nodes system, and consist in a timing breakdown of a code iteration taken during the early stage of evolution of a cosmological simulation, when the particle distribution is close to uniform. The total number of particles is $N = 8.8 \cdot 10^6$. Implementation limitations prevented our performance model to simulate 512 concurrent processes, so that we limited our simulation to 32 processes, and scaled down 16-fold the measured compute time reported in [8]. Since the communication overhead for that run was just $\simeq 6\%$, we assumed a linear scalability of the code. The timing breakdown of our simulation is presented in fig. 4. The real system measurements are reported for comparison.

The figure shows that the force computation task and the tree traversal are the most expensive tasks. The relative computational weight of each task is qualitatively well reproduced by our model. Quantitatively, a certain discrepancy between our model and the real system timings originates from an over-estimation of the tree traversal and the force computation tasks, which also results in overestimating the total time, as shown in fig. 7. Since the main goal of our present model is versatility with respect to both hardware and software modelling, and qualitative accuracy, a quantitative discrepancy in a single case does not limit the general validity of the model.

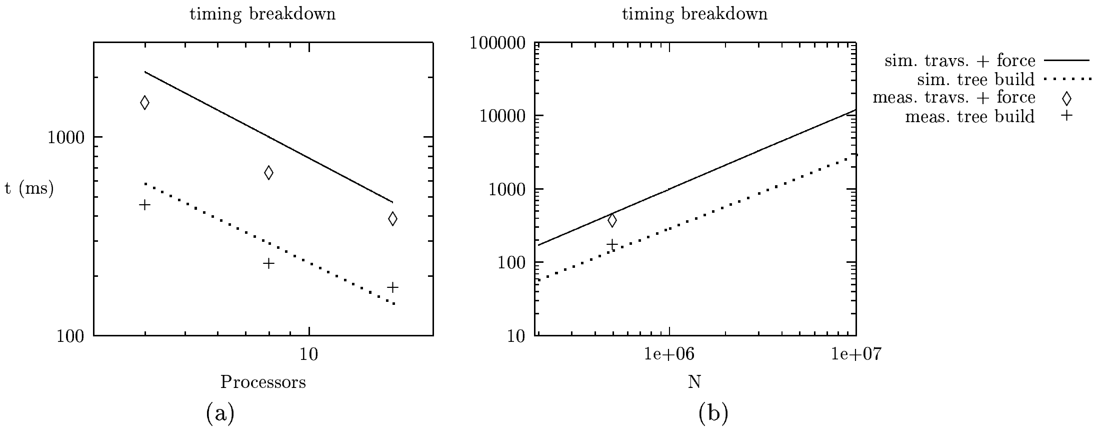

Fig. 5. Timimings of the GDT tasks. The real system timings are also reported. The hardware architecture is a Cray T3E. (a): performance scaling with the number of processors, being $N = 500\ 000$. (b): scaling with N, in a run with 16 processors. Note the change of scale in the two plots.

GDT on T3E This case reproduces the configuration described in [7], where the GADGET code is run on the T3E hosted at the supercomputing centre in Garching, Germany. Each computing node has a frequency of 300 MHz, and the communication network has a throughput of 500 MB/s. Three cases are reported in [7], each running the same cosmological simulation, where a system of 500 000 particles is evolved for 3350 time steps. The difference among the three cases is in the number of processors used. Since in this case measurements from three different hardware configuration are reported, we could compare our model results with a larger set of timing values. We also simulated the fastest configuration, scaling the value of N. As reported in [7], we assumed that only 5% of the particles are selected on average at each time step for force computation. Similarly, we assumed that the local tree is rebuilt each 10 time steps. Timing breakdowns for both cases are shown in fig. 5.

Fig. 5(a) shows the performance gain as the number of processors increases. The trend in the measurements suggests a saturation in the attained performance, arguably due to an increasing load imbalance. This trend is not visible in our model results, because load imbalance is not modelled. The overall pattern of fig. 5(b) resembles the HOT case pattern. Now the tree build task is more relevant, despite of the fact that it is performed only every ten iterations. This is to be expected, since the tree build task is performed for all particles, while the tree traversal and force computation tasks are performed only for a small fraction (5%) of the selected particles. Also in this case our model results match the real system timings. The latter did not provide separate values for the tree traversal and the force computation tasks, so that only the aggregate value can be reported on the plot.

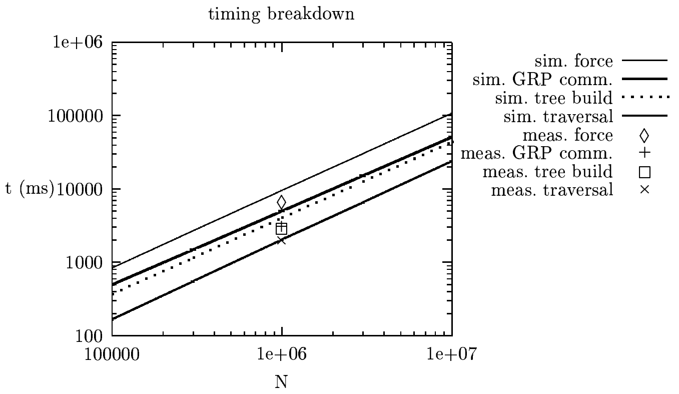

Fig. 6. Timings of the sequential treecode tasks. The real system timings are also reported. The hardware architecture consists of a Compaq 500 MHz workstation connected to a GRAPE-5.

Sequential treecode on GRAPE-5 Here we simulate the configuration described in [15]. In that case, a modified treecode is used to simulate a system containing about one million particles, and groups of $\simeq$ 2000 particles share the same interaction list. This code is run on a Compaq workstation with a 500 MHz Alpha 21264 processor, connected to a GRAPE-5 board containing 96 virtual pipelines ([1]) , each one able to compute a force interaction in 75 nanoseconds. Estimating a force interaction as 30 flops, the aggregate performance of a GRAPE-5 board is 38.4 Gflop/s. Fig. 6 shows the results of our simulation model, compared with the real system timings, as reported in [15].

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

Cases comparison We compare here the three cases presented above. We show in fig. 7 a plot of the time taken by a code iteration versus N, as obtained from our simulation model, compared with the real system measurements. The value for the HOT code on the Touchstone Delta is 16 times greater than the value reported in [8], in order to scale their 512 processor run to our 32 processor simulation. The converse scaling would have resulted in simulation values overlapping the values for the GRAPE case.

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

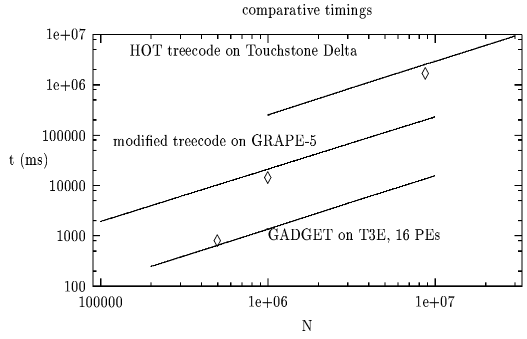

Fig. 7. Timings of a code iteration for the three simulated configurations.

The simulation values match well the real system measurements. This allows us to use our model to forecast the behaviour of other configurations with a good degree of confidence. Results from such simulations are given in the next section.

3.2 Model forecasts

We explore in this section the possibility of using a hybrid architecture consisting of a distributed general purpose system, where single nodes host zero or more GRAPE boards. We span the two-dimensional parameter space defined by the two quantities P, the number of nodes, and G, the number of GRAPEs. We assign to those quantities values as follows: $P \in \{1, 2, 4, 8, 12, 16, 20, 24\}$, $G \in \{0, 1, 2, 4, 8, 12, 16\}$. We simulate the same software configuration as described in the previous section with respect to the case related to the sequential treecode on GRAPE-5. The SPD we simulate in this case is the GRAPE-4 [16], whose performance per board is 30 Gflop/s, comparable to GRAPE-5's. It provides a higher accuracy with respect to GRAPE-5, and is used in fields as Globular Cluster dynamics on Planetesimal evolution [2], where high computing precision is required. The general purpose nodes are assumed to perform a floating point operation in 2 ns, and the communication network is assumed to have a 100 MB/s throughput.

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

Fig. 8 shows our results. It is clear that, as long as all nodes are connected to one (or more) GRAPE, a remarkable performance benefit appears. For com-

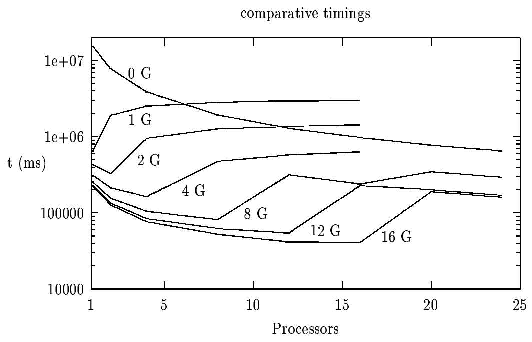

Fig. 8. Timings for a system with P processors and G GRAPEs. Comparison with a system without GRAPES (marked as 0 G) is also provided.

parison, we also provide the timing of a system without GRAPEs. When no all nodes are GRAPE hosts, the very large communication overhead due to sending particle and interaction list data is disruptive for performance. This result suggests that the communication task needs a very careful analysis, in order to design an efficient parallel treecode for hybrid architectures. Here we assumed that a 'non graped' node sends all its data to a single 'graped' node. We discuss this point further in the next section. The plot in fig. 8 also features an oscillatory behaviour, particularly evident in the case with 8 GRAPEs. The local minima (i.e. better performances) correspond to configurations where P is an exact multiple of G. In this case the computational load on the GRAPEs is perfectly balanced, whereas in the other cases some GRAPE bears a higher computational load from remote data.

4 Conclusions and future work

The efficient integration of hierarchical treecodes and hybrid architectures, consisting of distributed parallel systems powered by GRAPE SPDs, could lead to a very high performance environment for the solution of the N-body problem. In order to assess the optimal configuration, we realised a performance model, with which we can explore the parameter space of this problem. We validated our model by simulating existing configurations and comparing our results to real system measurements. We used our model to evaluate the performance of a hybrid architecture, and highlighted that an efficient implementation of such architecture is made difficult by an intrinsic high communication overhead. Issues like latency hiding, or partial redistribution of work to remove load imbalance, could help to solve this problem, and will be the object of further research. The model would also benefit from an accurate parameterisation of load imbalance. An experimental configuration realised according to those guidelines will allow us to fine-tune our model, which in turn will give a feedback to improve the actual configuration, in the direction of finding the optimal interaction of software, parallel host, and SPD.

Acknowledgements We acknowledge Arjan van Gemund for having made the PAMELA package available to us. We also acknowledge Jun Makino for the GRAPE boards that he kindly made at our disposal.

References

1. Barnes, J.E., Hut, P.: A hierarchical $O(N \log N)$ force-calculation algorithm. Nature **324** (1986) 446–449
2. Hut, P., Makino, J.: Astrophysics on the GRAPE family of special purpose computers. Science **283** (1999) 501–505
3. Makino, J., Fukushige, T., Koga, M.: A 1.349 tflops simulation of black holes in a galactic center on GRAPE-6. In: Proceedings of the Supercomputing 2000 conference, ACM press (2000)
4. Athanassoula, E., Bosma, A., Lambert, J.C., Makino, J.: Performance and accuracy of a GRAPE-3 system for collisionless N-body simulations. Montly Notices of the Astronomical Society **293** (1998) 369–380
5. Makino, J.: Treecode with a special-purpose processor. Publications of the Astronomical Society of Japan **43** (1991) 621–638
6. Becciani, U., Antonuccio-Delogu, V.: Are you ready to FLY in the Universe? a multi-platform N-body tree code for parallel supercomputers. Computer Physics Communications **136** (2001) 54–63
7. Springel, V., Yoshida, N., White, S.D.M.: GADGET: a code for collisionless and gasdynamical cosmological simulations. New Astronomy **6** (2001) 79–117
8. Warren, M.S., Salmon, J.K.: A parallel hashed oct-tree N-body algorithm. In: Proceedings of the Supercomputing '93 conference, ACM press (1993) 12–21
9. Spinnato, P.F., van Albada, G.D., Sloot, P.M.A.: Performance modelling of distributed hybrid architectures. IEEE Transactions on Parallel and Distributed Systems **submitted** (2001) Also available at: `http://www.science.uva.nl/~piero`.
10. Spinnato, P.F., van Albada, G.D., Sloot, P.M.A.: Performance prediction of N-body simulations on a hybrid architecture. Computer Physics Communications **139** (2001) 34–44
11. van Gemund, A.: Performance prediction of parallel processing systems: The PAMELA methodology. In: Proceedings of seventh ACM International Conference on Supercomputing, ACM press (1993)
12. Salmon, J.K., Warren, M.S.: Skeletons from the treecode closet. Journal of Computational Physics **111** (1994) 136–155
13. Barnes, J.E.: A modified tree code: Don't laugh, it runs. Journal of Computational Physics **87** (1990) 161
14. Aarseth, S.J.: From NBODY1 to NBODY6: The growth of an industry. Publications of the Astronomical Society of the Pacific **111** (1999) 1333–1346
15. Kawai, A., Fukushige, T., Makino, J., Makoto, T.: GRAPE-5: A special-purpose computer for N-body simulations. Publications of the Astronomical Society of Japan **52** (2000) 659–676
16. Makino, J., Taiji, M., Ebisuzaki, T., Sugimoto, D.: GRAPE-4: a massively parallel special-purpose computer for collisional N-body simulations. Astrophysical Journal **480** (1997) 432–446