A&A 531, A120 (2011)
DOI: 10.1051/0004-6361/201015933

Astronomy & Astrophysics

Comparisons of different codes for galactic *N*-body simulations

P. Fortin[1,2,3], E. Athanassoula[2], and J.-C. Lambert[2]

[1] Université Pierre et Marie Curie, LIP6 UMR 7606, 4 place Jussieu, 75252 Paris Cedex 05, France
e-mail: Pierre.Fortin@lip6.fr
[2] Laboratoire d'Astrophysique de Marseille (LAM), UMR6110, CNRS/Université de Provence, Technopôle de Marseille-Etoile, 38 Rue Frédéric Joliot Curie, 13388 Marseille Cedex 20, France
[3] HiePACS Team (former ScAlApplix project), INRIA Bordeaux – Sud-Ouest/CNRS – LaBRI UMR 5800/PRES de Bordeaux, 351 cours de la Libération, 33405 Talence Cedex, France

Received 14 October 2010 / Accepted 2 May 2011

ABSTRACT

Context. *N*-body simulations are widely used for galactic dynamics studies. Several different algorithms have been developed and a number of authors have made their codes public.
Aims. We wish to help potential users of these codes, particularly less experienced ones, to select the appropriate code by giving them relevant information.
Methods. We consider different implementations of three different algorithms, namely the Barnes-Hut tree-code algorithm (implemented both in GADGET-2 and as a NEMO program), the fast multipole method (implemented in the FMB code – Fast Multipole with BLAS) and W. Dehnen's algorithm (implemented in *falcON* – force algorithm with complexity $O(N)$). We compare extensively the timing and memory comparisons of these different implementations and discuss other advantages and disadvantages of these codes.
Results. In terms of serial executions, *falcON* is clearly the fastest code, in many cases by a wide margin. In addition, it is much less sensitive to concentration than other codes, it conserves linear momentum, its computational costs depends roughly linearly on the number of particles, and it allows for manipulators. GADGET-2 is the second fastest and, being parallel, can outperform *falcON* if a sufficient number of processors is used, the number of which, in our tests is between three and ten. Its public version includes an SPH representation of the gas component. FMB is faster than the implementation of the Barnes-Hut algorithm in NEMO, as long as the particle distribution is not too concentrated, but remains considerably slower than *falcON* and GADGET-2. It has, nevertheless, the highest parallel efficiency and memory scalability.
Conclusions. We recommend *falcON* for pure *N*-body simulations with a relatively restricted number of particles (up to a few million). In cases of a larger number of particles, a number of processors being available, or studies including gas and its physics, we recommend GADGET-2. FMB provides the highest parallel efficiency and memory scalability, but is considerably slower in execution time and is ill-adapted to the concentrated distributions encountered in galactic work.

Key words. methods: numerical

1. Introduction

Galaxies are complex objects, with many components and subcomponents and often with a complicated geometry. Thus, although analytical work and orbital structure theory have led to a considerable understanding of galaxy dynamics and evolution (Binney & Tremaine 2008), they are often faced with problems beyond their reach. At this point, the contribution of *N*-body simulations becomes essential. These have led to considerable progress, since they are fully non-linear, allow us to study time evolution, and, for many codes, are capable of handling very complex geometries. Thus, problems such as bar formation, bar properties, galaxy interactions, long-term evolution, or the study of the dynamics of dark matter and its mass distribution have relied heavily on these approaches.

In *N*-body simulations, a number of massive particles are used to represent the object under study, be it an isolated galaxy, or two or more interacting galaxies. However the number of particles in the simulation is by several orders of magnitude smaller than the number of even only the stars in the galaxy, so that the *N* bodies should be considered as Monte-Carlo realisations of the mass distribution in the system. The gravitational forces produced by this ensemble of particles is calculated and then used to update the particle positions and velocities. Forces are calculated anew from the new positions and the whole process re-iterated until the desired evolution is reached. It is the gravity part, i.e. the calculation of the forces, that is the most CPU intensive and that, therefore, limits the number of particles and time-steps that can be used in a simulation.

In this paper, we consider three different algorithms, namely the Barnes-Hut tree-code algorithm, the fast multipole method (FMM), and an algorithm developed by W. Dehnen, and we compare different implementations of these algorithms. In Sect. 2, we briefly describe the techniques and the specific codes that we use. One of these (the FMM) has been very little used so far in astronomical applications, so we describe it in somewhat more detail. None of the descriptions, however, is complete and the reader who wishes to become more acquainted with a given code will have to read the cited literature. In Sect. 3, we describe the different particle distributions to which the codes will be applied. The performance of the different codes is compared in serial execution in Sect. 4 and in

parallel execution in Sect. 5. We discuss our results and certain advantages and disadvantages of the codes in Sect. 6 and we conclude in Sect. 7.

2. The different algorithms

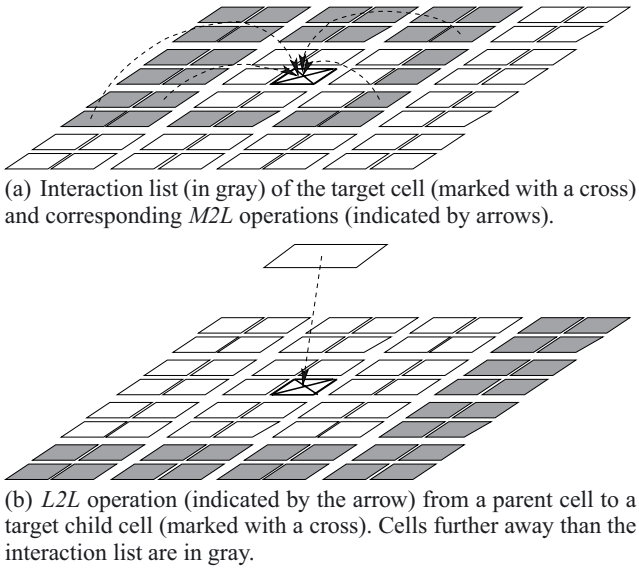
We compare three different algorithms, namely the Barnes-Hut tree-code algorithm, the FMM, and W. Dehnen's algorithm, in specific implementations.

We restrict ourselves to galactic simulations and do not consider cosmological simulations. We also focus only on hierarchical N-body algorithms and do not consider other N-body algorithms such as direct summation or Particle-Mesh (PM) methods and their variants (Hockney & Eastwood 1988; Knebe et al. 2001; Springel 2005), such as (adaptive) Particle-Particle/Particle-Mesh, Tree Particle-Mesh, and multigrid codes... The error behavior is indeed significantly different, and, in contrast to hierarchical methods, the CPU time required for particle mesh methods does not depend mainly on the number of bodies.

The Barnes-Hut algorithm (Barnes & Hut 1986) calculates the forces from a distribution of N bodies in a $O(N \ln N)$ operation count thanks to monopole (and possibly quadrupole) moments and to an octree data structure. The octree is constructed by inserting bodies one by one and by subdividing octree leafs containing more than a given maximum number of bodies. The octree is then recursively traversed for each target body, and "body-cell" or "body-body" interactions are evaluated depending on the acceptance criterion $D/r < \theta$, where D denotes the octree cell side length, r is the distance from the target body to the cell center of mass, and θ is *the opening angle*, which is an input parameter that controls the accuracy of the force computation.

The FMM (Cheng et al. 1999) solves the N-body problem in a $O(N)$ operation count, with a theoretically proven error bound for any given precision. As in the Barnes-Hut algorithm, the force field is decomposed into a near field part, directly computed, and a far field part, approximated in the FMM thanks to multipole and local expansions. For Poisson's equation in astrophysics as well as in molecular dynamics, the required multipole and local expansions are based on spherical harmonics. In the traditional FMM presented here, the maximum degree in these expansions, denoted by P, determines the accuracy of the computation. Greater P values imply more precise approximations of the far field, but the classic FMM operation count grows as $O(P^4 N)$. When considering a cubic cell c_{source}, the multipole expansion based on the center of c_{source} represents the impact of source particles inside c_{source} on target particles sufficiently far away from c_{source}. In contrast, the local expansion based on the center of a cubic cell c_{target} represents the impact of source particles sufficiently distant from c_{target} on target particles inside c_{target}. To convert the multipole expansion of c_{source} into a local expansion for c_{target}, the two cells must be *well-separated*, i.e. they cannot share a boundary point, which guarantees the error introduced by both expansions and by the multipole-to-local operation.

In practice, the particle space is hierarchically decomposed by means of a 3D octree and the algorithm requires first an upward pass of this octree to build the multipole expansions of all cells in the octree. Multipole expansions of the octree leafs are built directly from the particles within the leaf, whereas multipole expansions of internal cells result from the translation (multipole-to-multipole, or *M2M* operation) of the multipole expansions of their eight child cells. During a downward pass, the local expansion of each cell c is then computed from the conversion of the multipole expansions of all cells in its *interaction list* (multipole-to-local, or *M2L* operation): where c_{p} denotes the parent cell of c in the octree, the interaction list of c is defined as the set of all children of the nearest neighbors of c_{p} that are not themselves nearest neighbors of c (see Fig. 1a). All cells in the interaction list of c are indeed *well-separated* from c. The interactions due to cells further than the interaction list of c are taken into account thanks to the local expansion translation of c_{p} (local-to-local, or *L2L* operation, see Fig. 1b). In each octree leaf c_{l}, the far field forces are finally deduced from the evaluation of the local expansion of c_{l}. The near field forces for particles within c_{l} result from the direct computation with particles from the nearest neighbors of c_{l}, as well as from other particles within c_{l}. In the end, far field forces and near field forces are added to consider all interactions for all particles. More information can be found in Cheng et al. (1999). Compared to the Barnes-Hut algorithm, the local expansions enable "cell-cell" interactions and, along with this local expansion inheritance via *L2L* operation, result in a $O(N)$ algorithm.



(a) Interaction list (in gray) of the target cell (marked with a cross) and corresponding *M2L* operations (indicated by arrows).

(b) *L2L* operation (indicated by the arrow) from a parent cell to a target child cell (marked with a cross). Cells further away than the interaction list are in gray.

Fig. 1. Downward pass steps of the 2D FMM (with a quadtree). The translation to the 3D FMM (with an octree) is straightforward.

Two algorithms have so far provided an adaptive version of FMM (used with a non-uniform distributions of particles). Like the Barnes-Hut algorithm, the algorithm of Cheng et al. (1999) fixes the maximum number of particles per leaf in the octree, which implies an octree with leaves at different levels and without any empty cells. In contrast, the algorithm of Nabors et al. (1994) does not limit the number of particles per leaf, but merely fixes the height of the octree: the possible numerous empty cells are simply skipped. This results in a modified adaptive algorithm with proven linear operation count without any assumption about the particle distribution (see Nabors et al. 1994). In the context of parallel computing in multi-process mode, this second algorithm also yields more predictable communication patterns (Coulaud et al. 2010).

Finally, Dehnen's hybrid algorithm (Dehnen 2000, 2002) combines the advantages of both the Barnes-Hut method and the FMM, namely specific, relatively low precision expansions and "cell-cell" interactions, to obtain a $O(N)$ algorithm. This algorithm can be considered as a nontraditional FMM, specific to the relatively low precisions required for the softened gravity

in astrophysics. This requires local expansions based on cartesian Taylor expansions of the Greens functions, and a specific multipole acceptance criterion that can control (along with the expansion order) the accuracy and the cost of the computation. Moreover, the downward pass is recursively formulated via a dual-traversal.

To compare these algorithms, we use the following implementations: the *NEMOtree1*, the GADGET-2 code, the *falcON* code, and the FMB code.

NEMOtree1 is an implementation of the Barnes-Hut algorithm (Barnes & Hut 1986), and publicly available in the NEMO toolbox[1] (Teuben 1995). It uses the classical Barnes-Hut acceptance criterion (hereafter BH), applied here with $\theta = 0.7$ and with monopole moments only.

The GADGET-2 cosmological code of V. Springel (public version 2.0 Springel 2005) is also an implementation of the Barnes-Hut algorithm, which is widely used for galactic dynamics studies. We present results with the classical Barnes-Hut acceptance criterion (with $\theta = 0.7$), as well as with the GADGET-2 special enhanced "relative criterion" (hereafter RC). The RC criterion has indeed been shown in Springel (2005) to deliver, at the same computational cost, somewhat more accurate forces. The corresponding tolerance parameter is set to $\alpha = 0.01$ according to Fig. 1 in Springel (2005), where the relative criterion with $\alpha = 0.01$ has indeed been shown to be more precise, while requiring less computation, than the Barnes-Hut acceptance criterion with $\theta = 0.7$. We do not use the TreePM computation and for the memory allocation parameters we set the communication buffer size to 30 MBytes, a particle allocation factor of 1.6 (so that each process allocates space for 1.6 times the average number of particles per process) and a tree allocation factor of 0.8 (to set the number of internal tree-nodes allocated in units of the particle number).

The FMB code (Fast Multipole with BLAS[2]) is an implementation of the traditional fast multipole method (Coulaud et al. 2007, 2008, 2010). This code is based on the algorithm of Nabors et al. (1994): the octree height H is optimally set by the user to minimize the computation time. Usually, the H value must roughly balance the near field and far field computations, which depends on the particle distribution and the P value used. An *octree with indirection* is used as the octree data structure to manage a highly unbalanced octree in non-uniform particle distributions (Coulaud et al. 2010). For the relatively low precisions required in galactic dynamics, we use $P = 4$ in this paper, and the degrees of the *M2L* operator terms are limited to P as in multipole and local expansions; as presented in Coulaud et al. (2008), this approach is indeed more efficient for low P values. In addition, FMB can compute efficiently some expansion operations thanks to a matrix formulation with BLAS routines (Coulaud et al. 2008). However no BLAS computation will be performed here since, as shown in Coulaud et al. (2010), the BLAS routines are mainly efficient for greater P values or within large uniform areas, which are not present in particle distributions representing galaxies or groups of galaxies.

The *falcON* (force algorithm with complexity O(N)) code is W. Dehnen's own implementation of its algorithm. We use the public version 3.0.9I of the *gyrfalcON* full-fledged N-body code (GalaxY simulatoR using falcON). *falcON* uses a specific mass-dependent tolerance parameter (Dehnen 2002): at the same θ value, the standard Barnes-Hut acceptance criterion (used in *NEMOtree1* and in GADGET-2) being more accurate, we therefore set $\theta = 0.6$ for *falcON*. The maximum number of bodies in un-split cells is set to 16. We note that *falcON* allows for various softening kernels, although we use here the standard Plummer softening to match the other codes.

The *falcON* and *NEMOtree1* codes are only serial, whereas GADGET-2 and FMB are parallel. GADGET-2 relies indeed on the MPI standard for message-passing communications[3], and FMB is a hybrid MPI-thread code (Coulaud et al. 2007) that exploits both multi-process and multi-thread modes.

Both GADGET-2 and *falcON* feature individual adaptive time-steps, whereas *NEMOtree1* and FMB do not offer this feature. Although it is possible to implement individual time-steps in these codes, this is well beyond the scope of this paper. We therefore do not consider individual time-steps in our comparisons.

3. Computing miscellanea

We now compare the above described algorithms for the following distributions of N particles. The number of particles N varies between 10^5 and 5×10^7, except for the two last distributions (*group* and *galaxy*) where N is fixed.

- An artificial uniform distribution of particles inside a 3D cube, generated by the NEMO `mkcube` command.
- A classical astrophysical model, the Plummer model (Binney & Tremaine 2008), with a scalelength equal to 1.0 and a truncated radius chosen such that it contains 0.999 of the mass of the Plummer sphere extended to infinity, the mass within the truncation radius being set to 1.0.
- Truncated power-law density profiles whose density is proportional to $r^{-\gamma}$, with cut-off radius set to 1.0.
- A model of a group of 50 identical galaxies, each modelled as a Plummer sphere, composed of 1.5×10^6 particles in total and referred to hereafter as *group*.
- The three remaining models are realisations of disc galaxies with a halo and a disc component, described in Athanassoula (2003). The halo has a small core with a radius equal to half the disc scale-length, i.e. it is quite centrally concentrated. *galaxy0* is the snapshot taken directly from the initial conditions, i.e. before any evolution has started. *galaxy1* and *galaxy2* are snapshots from evolved stages of these simulations, which both contain a bar. They are thus more centrally concentrated than *galaxy0*, since the evolution have pushed both disc and halo material inwards to the central region. All three configurations have about 1.2×10^6 particles.

The time necessary to calculate the gravitational forces in a given particle distribution depends of course on the precision of the calculations, which in turn depends on the numerical values of specific parameters of the code, such as e.g. for the tree code the opening angle θ. Thus, for a perfect comparison of the timings the precision should also be exactly equal. Unfortunately this is not possible for two reasons. One is that the values necessary for even rough equality change from one particle distribution to another. The second one is that there are several possible ways of measuring the precision with which the forces are calculated and if two precisions are exactly equal for one definition they can only be roughly similar for another. For example, using as an accuracy measure the L_2 norm difference from a direct summation force evaluation, we find that GADGET-2 RC is somewhat

[1] Namely *treecode1* in NEMO – A Stellar Dynamics Toolbox (1986–2009): `http://bima.astro.umd.edu/nemo`

[2] BLAS (Basic Linear Algebra Subprograms) routines are highly optimized implementations of linear algebra operations.

[3] MPI Forum home page: `http://www.mpi-forum.org/`

less accurate than GADGET-2 BH, while the opposite is true if we use the 99.9 per cent percentile of the force error distribution computed as the individual relative error on each particle (Springel 2005). It is thus necessary to be content with similar, or only roughly equal precisions.

We measured the accuracy using the relative error with respect to the forces obtained by direct summation, and used the L_2 norm as measure, as e.g. in Cheng et al. (1999) and Coulaud et al. (2008). In doing this we ensured that, for the parameter values given in the previous section, errors are roughly concordant for all codes and all particle distributions.

These tests, as well as all the timings for serial execution were carried out on one processor of a computer with 4 dual-core 1.8 Ghz AMD Opteron processors and a 16 GB memory. The operating system is Linux and the compiler is gcc-3.4.6 (g++-3.4.6 for *falcON*). The results for parallel execution given in Sect. 5 necessitated a larger number of processors and were therefore run on a different computer, as described in that section.

4. Serial execution comparison

We now compare the force computation time for the different codes presented in Sect. 2 and ignore times such as the I/O time required to load the particles from the files into the memory, the time required to update the particle positions at the end of the time-step and the time used for writing time-step snapshots. Since, in GADGET-2 for example, some specific computations are performed during the first time-step only, we consider the second time-step for GADGET-2, *NEMOtree1* and *falcON*. All velocities in all distributions are therefore set to zero in order to minimize the particle displacements between the first and the second time-steps. Similarly, in FMB the tree data structure construction is performed only once and is treated as a precomputation; we therefore do not consider the tree construction time for all codes, but the FMM upward pass time is of course included in the FMB force computation time. Moreover, all times correspond to wall-clock times and all of these serial executions are performed in single precision (32 bits) floating point arithmetic.

Since the potential computation is required only at some time-steps (to check the total potential energy conservation), only the forces (or accelerations) are computed, not the potential.

Finally, we also measure the memory consumption of all codes. For this purpose, we intercept all dynamic[4] memory system calls and retain the maximum amount of memory required by the process over its execution.

4.1. Various distributions of particles

We present in Fig. 2 the comparisons of serial force computation times for various distributions of 1.2×10^6 particles. Several conclusions can be drawn from this figure.

First of all, *falcON* is much faster than all other codes, for all distributions. Depending on the distribution and the particle number, *falcON* is 7 to 21 times faster than *NEMOtree1*, 3 to 11 times faster than GADGET-2 (with or without the relative criterion), and 4 to 21 times faster than FMB.

While *NEMOtree1* is almost always the slowest one, FMB is as fast as GADGET-2 for uniform distributions, but slower for Plummer and other galactic distributions. We note that GADGET-2 BH is faster than *NEMOtree1* with the same Barnes-Hut criterion (both with $\theta = 0.7$).

[4] Static memory requirements are here insignificant in all codes.

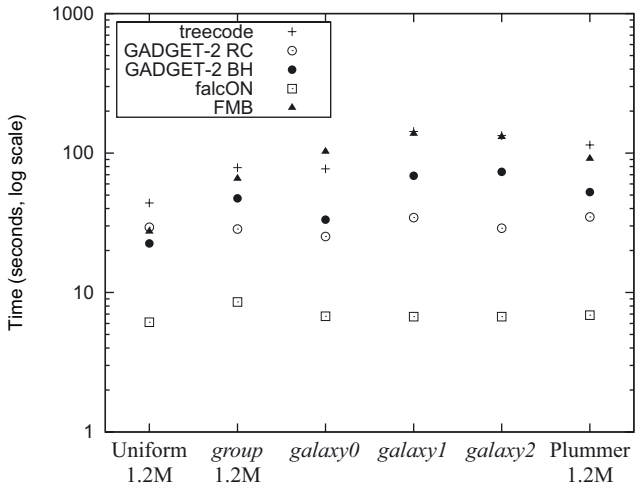


Fig. 2. Force computation time (in logarithmic scale) for various distributions. The *group* 1.2 M distribution is derived from the original *group* by randomly removing 3×10^5 particles.

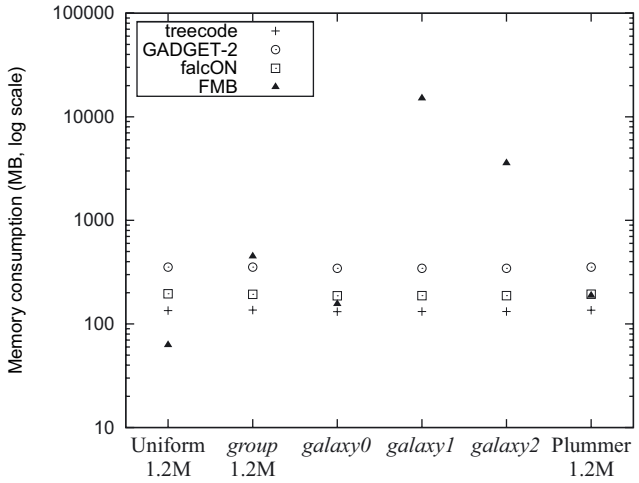


Fig. 3. Memory consumption (in logarithmic scale) for various distributions. The *group* 1.2 M distribution is derived from the original *group* by randomly removing 3×10^5 particles.

One can discern an increase in the force computation time for *NEMOtree1*, FMB, and GADGET-2 (lower for GADGET-2 RC than for GADGET-2 BH) between the *galaxy0* and *galaxy1* distributions. As discussed in Sect. 3, *galaxy1* is indeed more concentrated than *galaxy0*. However, for *falcON* there is no increase at all. In the next section, we thus study more precisely the effect of central concentration, using the power-law density profiles.

As far as memory consumption is concerned, Fig. 3 shows that for *NEMOtree1*, GADGET-2[5] and *falcON*, the memory consumption does not depend on the distribution, and is constant for a given number of particles. In contrast, FMB is far more sensitive to the particle distribution, especially for *galaxy1* and *galaxy2*, which are more concentrated and hence require a deeper octree.

[5] Differences between GADGET-2 RC and GADGET-2 BH memory requirements are indistinguishable. We therefore do not differentiate GADGET-2 RC and GADGET-2 BH for the memory consumption comparison.

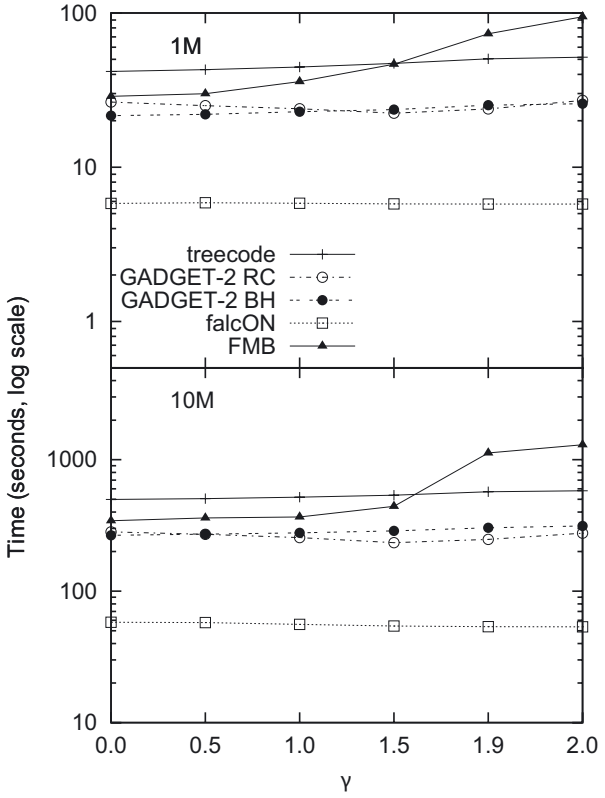


Fig. 4. Force computation time (in logarithmic scale) for power-law density profiles.

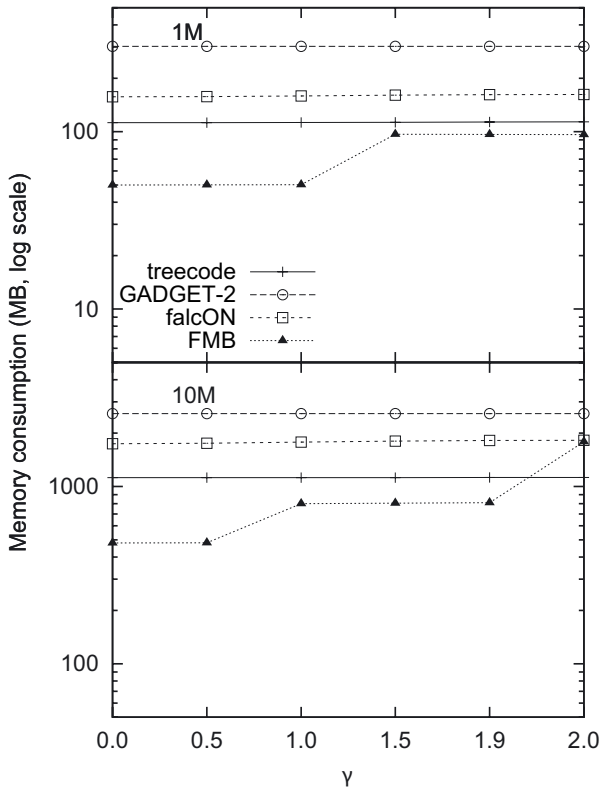


Fig. 5. Memory consumption (in logarithmic scale) for power-law density profiles.

4.2. The effect of central concentration for power-law density profiles

We now compare power-law density profiles with different total particle numbers and different concentration indices. We performed these comparisons for particle numbers ranging between 100 K and 50 M, but show in Fig. 4 illustratively only results for 1 M and 10 M. Our discussion, however, is based on all the comparisons we made.

For all distributions, *falcON* is once again much faster than all other codes. Depending on the distribution profile, *falcON* is 5 to 12 times faster than *NEMOtree1*, 3 to 7 times faster than GADGET-2 (with or without the relative criterion), and 4 to 28.5 times faster than FMB. These results also show that the performance of *falcON* hardly depends at all on the particle distribution profile. On the other hand, the computation time of *NEMOtree1* and GADGET-2 BH increases somewhat with γ, and FMB computation times are strongly affected by the increase in γ. As for untruncated Plummer and power-law distributions, which would then be more concentrated, *falcON* computation times would have been much less affected.

Moreover, GADGET-2 (with or without the relative criterion) is always faster than *NEMOtree1*. GADGET-2 RC appears to be faster than GADGET-2 BH for sufficiently large ($N \geq 10$ M) and concentrated distributions ($1.5 \leq \gamma \leq 2$), mainly because GADGET-2 RC is less sensitive than GADGET-2 BH to the increase in γ.

For low values of γ ($\gamma = 0$ and $\gamma = 0.5$) corresponding to weakly concentrated distributions, FMB is generally faster than *NEMOtree1* and comparable in speed to GADGET-2. In contrast, for strongly concentrated distributions (high values of γ like $\gamma = 1.9$ and $\gamma = 2$), FMB becomes slower than both GADGET-2 and *NEMOtree1*, especially for distributions with 50 million particles. This shows that the FMM algorithm implemented in FMB is not very suitable for strongly concentrated distributions.

Figure 5 confirms our first conclusions about memory consumption. *NEMOtree1*, GADGET-2, and *falcON* are insensitive to particle concentration, whereas FMB memory consumption increases strongly with it: each step for FMB in Fig. 5 corresponds to an increment in the octree height.

4.3. The effect of particle number N

Figure 6 shows the dependence of the timing on the number of particles N. *falcON* is the only one of the four codes we compare with whose computation time is linearly dependent on N. The cost of the two gadget codes only weakly depends on N, particularly GADGET-2 RC, or GADGET-2 BH with non-concentrated distributions. *NEMOtree1* has a much stronger dependence with N. The FMB dependency is again strongly affected by the concentration of the distribution.

Finally, Fig. 7 presents the dependence of the memory consumption on the number of particles N. GADGET-2 memory requirements decrease for increasing N values since GADGET-2 is a massively parallel code whose data structures and communication buffers are most well suited to high N values. *NEMOtree1* is the only code whose memory requirements are strictly linearly dependent on N. The *falcON* memory requirements have a stronger dependence on N, while FMB memory requirements are linearly dependent on N for uniform distributions ($\gamma = 0$) but have a strong dependence on N for concentrated distributions ($\gamma = 2$).

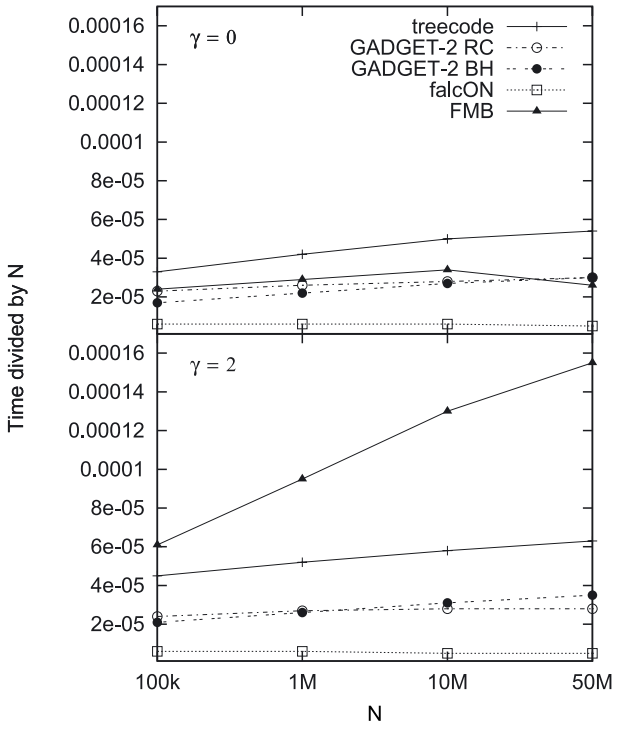


Fig. 6. Force computation time divided by N for two power-law density profiles.

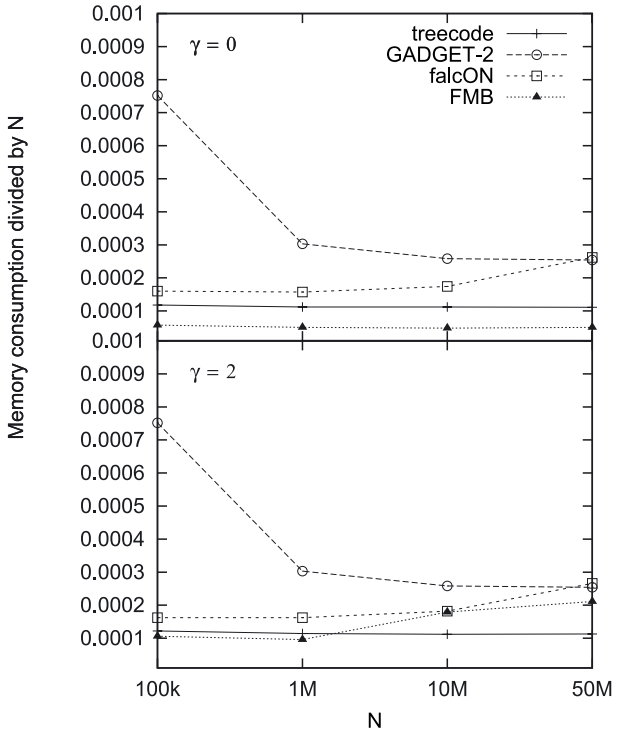


Fig. 7. Memory consumption divided by N for two power-law density profiles.

5. Parallel execution comparison

We now present results of parallel executions. GADGET-2 and FMB are indeed parallel codes that can use several processors to compute one single time-step. Serial computation times for *falcON* and *NEMOtree1* are also presented for comparison. These results were obtained on the IBM p575 cluster of the M3PEC center (University of Bordeaux 1). We use up to 8 SMP (Symmetric MultiProcessors) nodes linked with a 2×12 Gb/s Federation network. Each node has a 28 GB memory and 8 dual-core 1.5 Ghz Power5 processors. The total number of available cores is thus 128. Since the GADGET-2 code is a pure MPI code, we use one MPI process per core, namely up to 128 MPI processes, and one node for up to 16 MPI processes since intra-node MPI communications are faster than inter-node ones. In contrast, FMB is a hybrid MPI-thread code that is most efficient with one MPI process per node and 16 computation threads (one for each core available on the node) inside each MPI process: we refer to Coulaud et al. (2007) for more details. We therefore use up to 8 MPI processes for FMB (with a total of $8 \times 16 = 128$ threads). Since the hybrid MPI-thread parallelization of FMB requires a thread compliant MPI implementation, we here use the IBM vendor MPI implementation (version 1.2) on AIX 5.3. The same MPI implementation is also used for GADGET-2. *NEMOtree1*, FMB, and GADGET-2 are compiled with the IBM `xlc` compiler (version 7.0) whereas the *falcON* code is compiled with the `g++` compiler (version 4.0.2) since the *falcON* code uses advanced C++ features that are not supported by the `xlc++` compiler (IBM `C++` compiler). Before making the comparisons, we verified with the Opteron computer described in Sect. 3 that the compiler difference is an acceptable bias, similar for all distributions, and hence that our comparison can be made.

We use single precision computations for all codes, since (depending on the code) this is either faster than or as fast as double precision on this architecture. The parallel times presented here are computed in a similar way to serial times (see Sect. 4). In particular, we do not consider here the computation times required for the octree's construction and the parallel decomposition (for FMB and GADGET-2), as well as other precomputation times. Moreover, since other nodes in the cluster can be used by other users during our tests, the network bandwidth may not be entirely available to us. Therefore, we performed each parallel execution three times and present here the minimum measured time. As in Sect. 4, only the forces are computed. We finally note that GADGET-2 is able to balance the work-load among the MPI processes thanks to an estimation of the computation cost within each MPI process at the previous time-step. This however requires a new domain decomposition that is too costly to be performed at every time-step. We therefore present results with and without this work balancing (WB). We also measured the corresponding memory consumptions of these parallel executions. We proceed as for the serial executions, and retain the sum, over all processes, of the maximum amount of memory required by each process over its execution.

Figure 8 presents the computation times for different particle distributions. Figure 9 presents the parallel efficiencies of both GADGET-2 and FMB codes for the computation times presented in Fig. 8. These parallel efficiencies are used to measure the quality and the scalability of the parallel code (the closer to 1, the better) and are defined as

$$\text{Parallel\_efficiency} = \frac{\text{Serial\_Time}}{\text{Parallel\_Time} \times \text{Number\_of\_Cores}}. \quad (1)$$

Several conclusions can be drawn from these figures. One can look at the number of cores required by GADGET-2 and FMB to reach or outperform the faster but serial *falcON* code. For uniform particle distributions (not presented here), 4 cores are thus required by FMB to obtain computation times equal to *falcON* ones, and with 128 cores the parallel FMB is 25 times faster than the serial *falcON*. Both GADGET-2 RC and GADGET-2 BH also require 4 cores to reach *falcON*, and with 128 cores

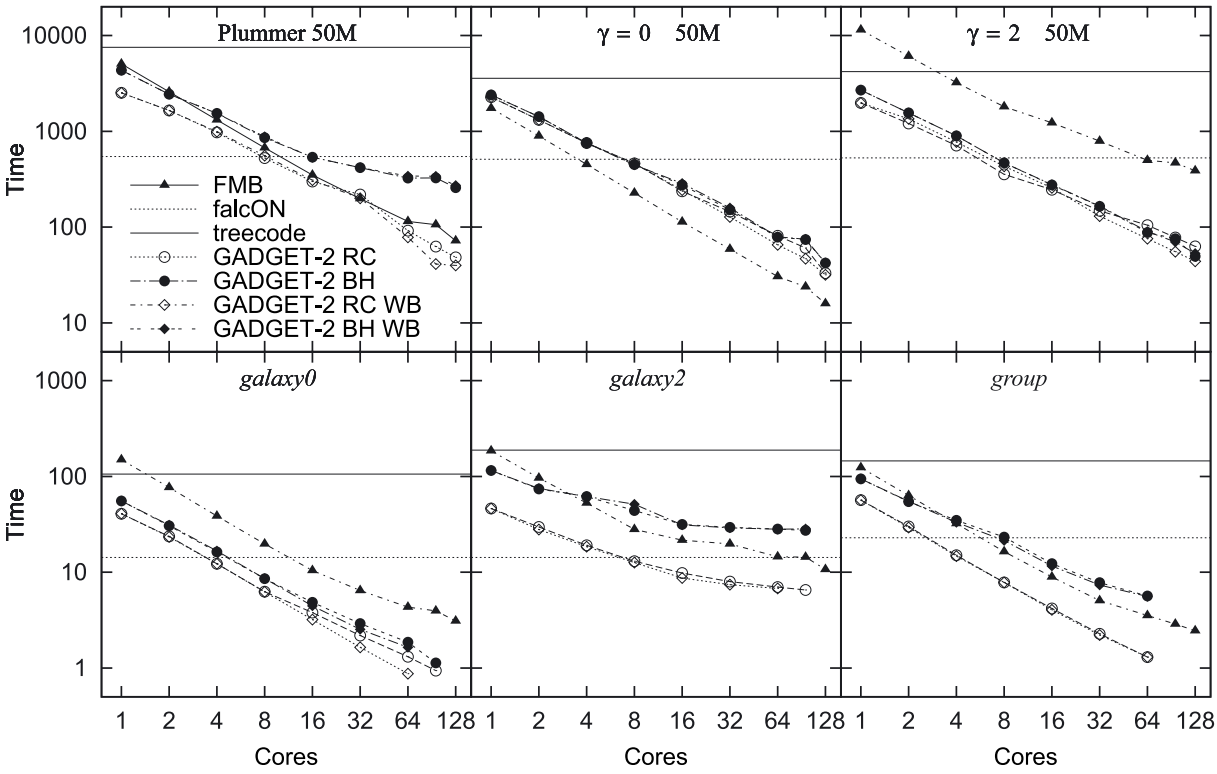


Fig. 8. Computation times for power-law density profiles and various other distributions as a function of the number of cores used.

GADGET-2 RC (respectively GADGET-2 BH) is more than 15 (resp. 27) times faster than the serial *falcON* code. For distributions representing galaxies or groups of galaxies, such as *group* and the Plummer model, the number of cores required to reach the performance of *falcON* increases to 8 for FMB. For more concentrated distributions such as the power-law density profiles with increasing values of γ, the FMB code is less and less adapted. While the parallel FMB with 128 cores is almost 32 times faster than *falcON* for $\gamma = 0$ with 50 million particles, it is only around 25% faster than *falcON* for $\gamma = 2$. In contrast, GADGET-2 is less sensitive to the particle concentration, and the number of cores required to match the execution times of *falcON* is quite stable, in most cases 4 for 10 M power-law density profiles, and 8 for 50 M power-law density profiles. In Sect. 4, we showed that in serial execution GADGET-2 RC is less sensitive than GADGET-2 BH to the distribution concentration. Here we see that in parallel it also fares better with concentrated configurations: this is particularly true for the distributions *galaxy0* and *galaxy1*, while for *group* it takes only around 3 processors to match the performance of *falcON*. This second result is in a way expected, as a tree-code should be well suited to particle distributions representing groups or clusters of galaxies and is important because of the number of studies dealing with more than one galaxy.

One can also see in Fig. 9 that the parallel efficiency of FMB is superior to that of GADGET-2. Furthermore, it usually tends to increase as the number of particles increase (not shown here), since as shown in Coulaud et al. (2007) the hybrid MPI-thread parallelization of FMB makes it possible to gain parallel efficiency over a pure MPI code such as GADGET-2.

Figure 10 indicates the memory consumptions of the various particle distributions. Parallel codes such as GADGET-2 and FMB tend to require more memory than serial codes because parts of the octree are replicated among the processes and because of the communication data structures. As the number of processes increases, the total memory requirements also increase. However, the FMB memory consumption is constant up to 16 cores, since FMB uses only one MPI process in these cases and the (up to) 16 threads share the single memory of this process. In contrast, the GADGET-2 memory consumption always increases when more cores are used, and always ends up exceeding the FMB one for all distributions. The important memory consumption of GADGET-2 can be reduced by decreasing the memory allocation values chosen in Sect. 2 (e.g. from (30, 1.6, 0.8) to (15, 1.1, 0.7)). But when the number of processes increases, these memory allocation values also have to be increased (e.g. to (100, 2.0, 1.0)), otherwise GADGET-2 fails some of our tests at too high process numbers. These greater memory allocation values imply naturally a greater total memory consumption.

6. Discussion

6.1. Serial execution comparison

Among the four codes that we test, *falcON* is the one that is least sensitive to the particle configuration and roughly linearly dependent on N. GADGET-2 RC depends also very little on the distribution profile. GADGET-2 BH and *NEMOtree1* are somewhat more sensitive to the distribution profile and FMB is even more sensitive, being ill-suited to too concentrated distribution profiles. Moreover, *falcON* is clearly faster than all other codes.

The FMM algorithm implemented in FMB is not well adapted to strongly concentrated distributions because of its iterative ("level by level") nature, as well as its octree data structure. For very concentrated distributions of particles, the overhead (in terms of computation time, but also in terms of memory requirement) of adding smaller boxes at a deeper height in order

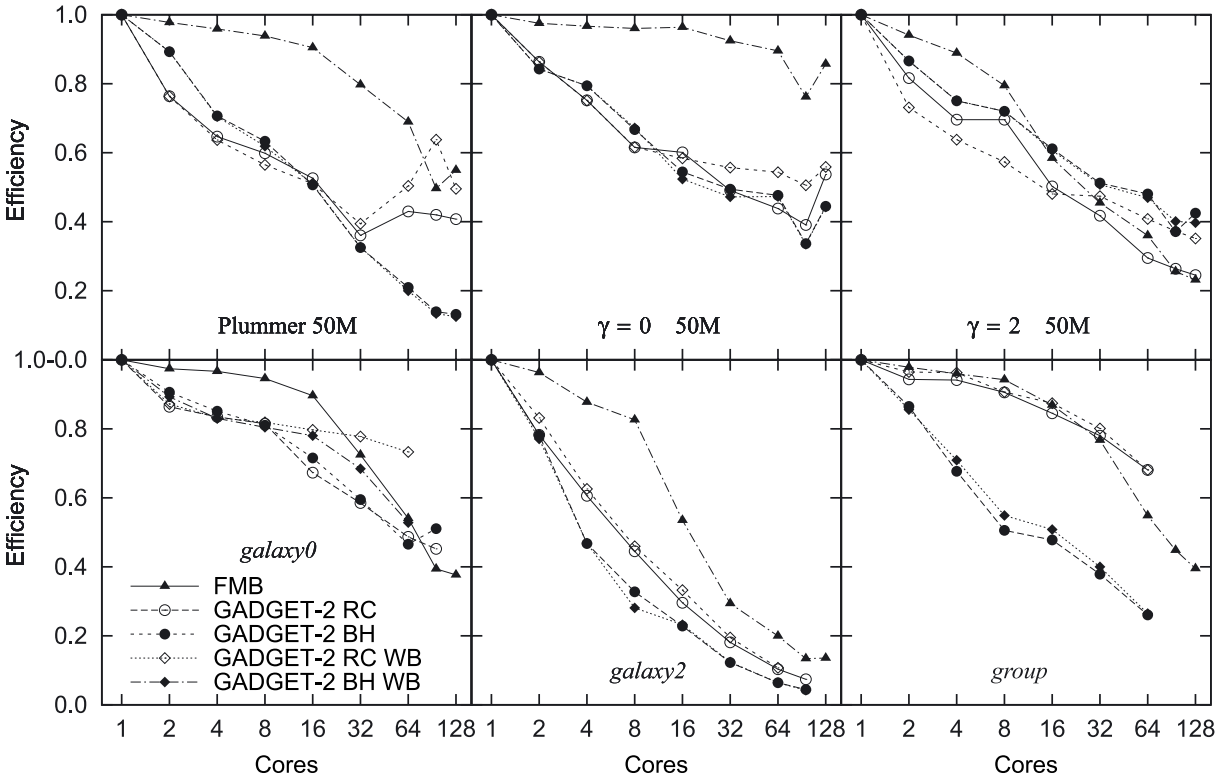


Fig. 9. Parallel efficiencies as a function of the number of cores used.

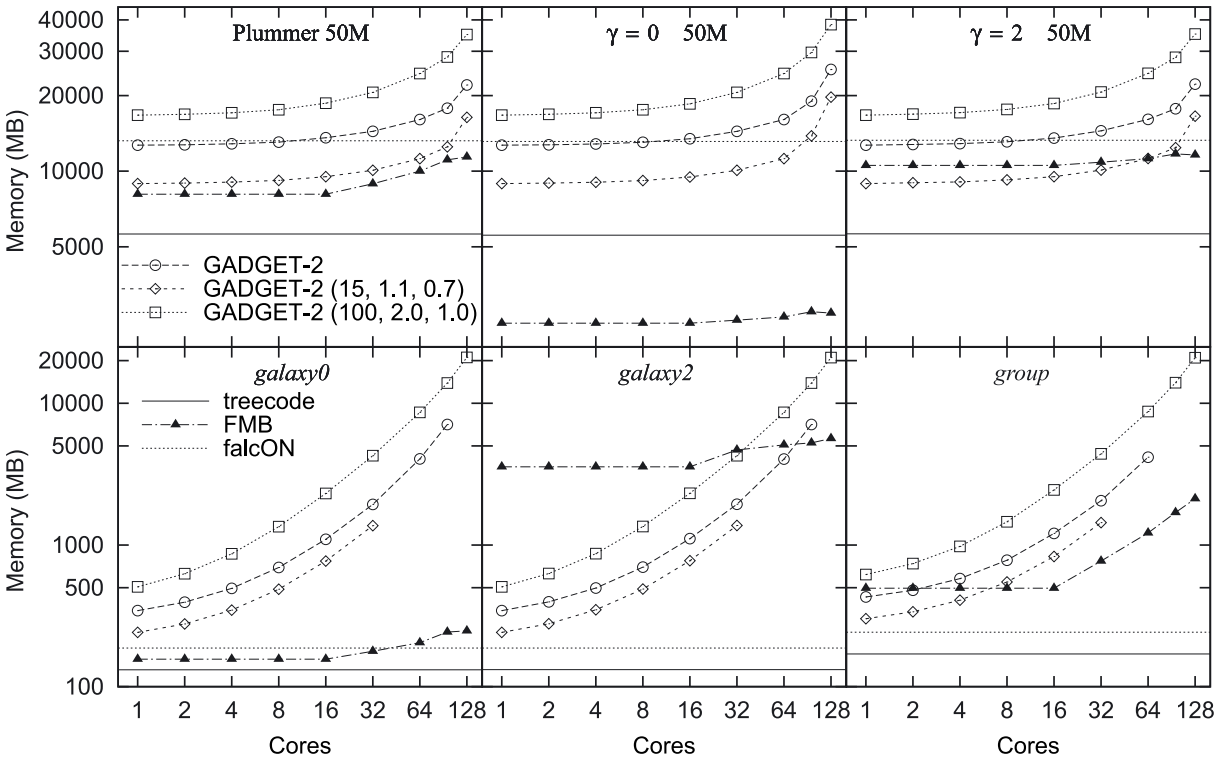


Fig. 10. Memory consumption for power-law density profiles and various other distributions as a function of the number of cores used.

to refine the computation space is much heavier for FMB than for the other codes, whose algorithms and data structure have recursive features.

The excellent performance of *falcON* can be explained in several ways. As argued in Dehnen (2002), Dehnen's algorithm is faster than the Barnes-Hut algorithm (used in *NEMOtree1* and GADGET-2) because the former contains an improved dual tree walk that enables cell-cell interactions and preserves the mutuality of gravity at the expansion level. These expansions are also fast to compute and specific to the relatively low precisions

Table 1. Number of calls to the square root computation function for *falcON* and FMB codes.

	falcON	FMB
Plummer 10k	$3,37 \times 10^5$	$5,18 \times 10^6$
Uniform 1M	$3,6 \times 10^7$	$3,86 \times 10^8$

required in N-body simulations of galaxies or groups of galaxies. In addition, this improved algorithm is very efficiently implemented in the highly optimized C++ *falcON* code.

An important point to keep in mind when comparing the traditional FMM to Dehnen's nontraditional FMM is that FMB uses a "rigid" interaction list, whereas *falcON* uses a "dynamic" acceptance criterion that adapts to the layout of the particles inside the source and target cells. *falcON* can thus precisely determine whether it is worthwhile (and allowed according to the error bound) using expansions in order to compute the interaction between the two cells, even if these are nearest neighbors. In FMB, the choice, in contrast, is only based on the octree geometrical factors and always chooses the direct computation for the nearest neighbors of the target cell (see Sect. 2). One could use an acceptance criterion based on the spheres encircling all the particles in each cell (instead of the spheres encircling totally each cell) as in Capuzzo-Dolcetta & Miocchi (1998) or Krishnan & Kale (1995), but the sphere and expansion centers still have to match the cell center in these FMMs. Thus, *falcON* expansions based on the center of mass of all cell particles will always be more appropriate for the type of applications described here. In practice, this difference affects the number of square roots used (one for each interaction computed with the direct method); this is shown in Table 1, where we have at least ten times more square root computations for FMB than for *falcON*. There is, therefore, much less direct computation for *falcON* than for FMB. Moreover, the FMM expansions based on spherical harmonics are not limited to relatively low precisions, and are thus more costly to handle than either the monopole or quadrupole moments of the Barnes-Hut algorithm, or the Taylor expansions used in *falcON*, well matched to the relatively low precisions required for the softened gravity in astrophysics. In the end, the *falcON* overall computation time is thus shorter than the FMB one for N-body simulations of galaxies and groups of galaxies.

6.2. Parallel execution comparison

The parallel efficiencies of FMB and GADGET-2, are higher for uniform distributions than galactic distributions. This is because the distributions of particles representing galaxies or groups of galaxies are highly non-uniform, so that an accurate load balancing over the MPI processes is harder to obtain when distributing the particles or the octree cells over the MPI processes than with more uniform distributions. As discussed e.g. by Springel (2005) or Coulaud et al. (2007), space filling curves such as the Morton or Hilbert orderings are used in practice with estimated costs (for example, roughly equal number of particles per process) or cost functions (for example, number of particles in the cell). These costs based on the particle number give only an approximation of the real computation cost thereby attributed to each process. The resulting load balancings are good in cases of regular data (particle) distribution, where the computation cost associated with each particle vary slightly. But the load balancings degrade in cases of (highly) irregular distributions, where the computation cost is much higher for particles in high density regions than for particles in low density regions. Moreover, in these high density regions there are also more communications, because of the tight coupling involved in galactic applications, that leads to a high degree of data dependency. These numerous communications then worsen the load imbalance because of the small computation grain[6] of galactic simulations. These require relatively low precisions and thus low expansion orders for the far field computation, which imply a fine computation grain. This fine computation grain makes the numerous communications costly (compared to the computation cost) and may also prevent these communications from being fully overlapped with computation. When measuring the computation cost of each process, correction of the particle distribution among the processes may improve the load balancing between two consecutive time-steps, but this costly parallel task cannot be performed at every time-step and this may still not lead to an optimal load balancing for highly concentrated distributions of particles.

As presented in Sect. 5, the hybrid MPI-thread parallelization of FMB makes it possible to gain parallel efficiency over a pure MPI code such as GADGET-2. This hybrid MPI-thread parallelization indeed offers greater load balancing among all the cores that share one given MPI process on one node. Owing to the shared memory, we can have a better load balancing on one node between threads inside one MPI process than between distinct MPI processes. More precisely, we can have several load balancings in shared memory, each load balancing being dedicated to one step in the FMM algorithm (the different steps having different costs). On the other hand, at the MPI level only one single load balancing is possible, which may not be optimal in galactic simulations as mentioned above. The MPI-thread parallelization makes it thus possible to improve the MPI load balancing within each node. An efficient dynamic load balancing implemented at the thread level can also improve locally the static load balancing among the nodes. Moreover, the MPI-thread parallelization also takes greater advantage of the mutuality of gravitational interactions. The multi-thread parallelization can indeed avoid all redundant direct computations, thus fully exploits the mutuality of gravity in the direct computations, whereas the MPI parallelization has to introduce either redundant direct computations or extra communications (Coulaud et al. 2007). There is, however, no such gain in parallel efficiency for FMB over GADGET-2 for too concentrated distributions such as $\gamma = 2$ power-law models and *galaxy1*, since FMB is ill-suited to too concentrated distribution profiles, as shown in Sect. 4.

As shown in Sect. 5, a better memory scalability is also obtained with the MPI-thread parallelization than with a pure MPI parallelization: the octree data structure is indeed shared by all threads inside each MPI process, thus saving memory consumption within each node. Reducing the memory allocation values of GADGET-2, to reduce its memory consumption, also implies a slight computation time increase since there will be less memory available for the work-load balance in the domain decomposition[7]. In contrast, by increasing GADGET-2's memory requirements it is possible to speed up the execution and improve its parallel efficiency, by enabling a more severe particle-load imbalance that may improve the work-load balance. This is presented in Fig. 11, where lower memory allocation values (15,

[6] In parallel computing, the *computation grain* refers to the size (in terms of CPU time) of the elementary task that can be performed without communication.

[7] See *User guide for GADGET-2*, V. Springel, 2005.

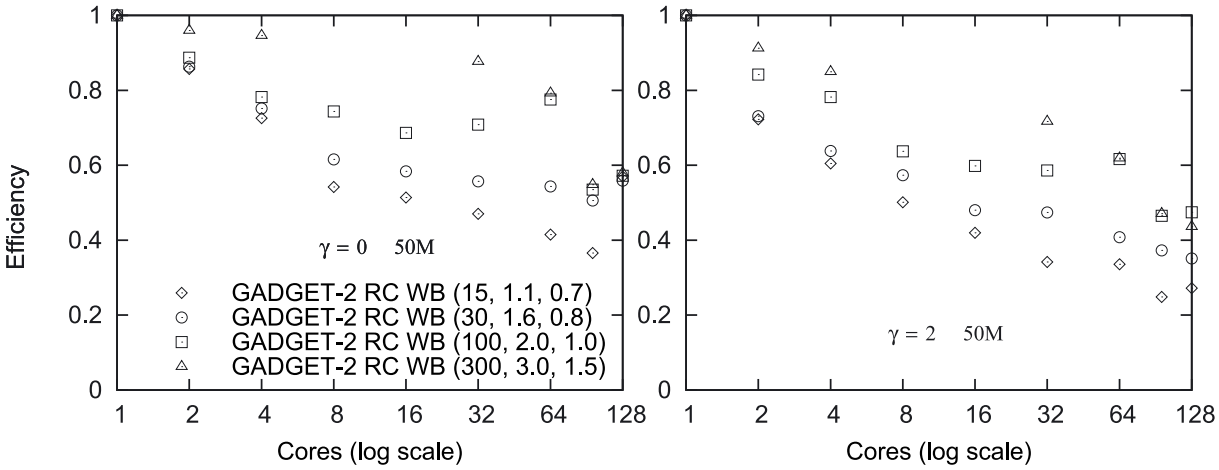


Fig. 11. Parallel efficiencies for power-law density profiles with GADGET-2 with various memory allocation parameters.

1.1, 0.7) produce a slow down of up to 14 points of parallel efficiency, whereas greater memory allocation values (300, 3.0, 1.5) offer up to 32 points of parallel efficiency gain but also sometimes prevent GADGET-2 from running, e.g. when there are too many processes (8 or 16) on one single node.

The best performance would clearly be obtained from a code with both hybrid MPI-thread parallelization and load balancing correction between two time-steps.

6.3. Further advantages of the various codes

The GADGET-2 code has a further, very important advantage, namely its public version includes gas, while a number of semi-public versions include star formation, feedback, and cooling, or even a central black hole. Thus, it can be used for a very large spectrum of problems involving galaxy evolution and interactions.

falcON has a number of many interesting and very useful features, which we now briefly describe.

6.3.1. Drift

In tree-codes in general, the positions of the centers of mass and density can drift with time. Thus, a model galaxy initially centered on the center of the coordinates may, by the end of the simulation, be considerably offset. This does not introduce any further complications, provided one keeps track of the location of the center of mass, or better the center of density, and takes this shift into account before performing any analysis. For *falcON*, however, this drift is very small, because the cell-cell interactions are computed in a completely symmetric fashion, such that, in contrast to the tree code, Newton's third law is satisfied by construction and so momentum is conserved to numerical accuracy.

6.3.2. Manipulators

In all simulations, it is possible to access, or manipulate the data in three different ways:

i) The standard procedure is to dump on disc full or partial information at given time intervals. These snapshots can later be analysed to provide the simulation results. The main disadvantage of this is that for certain tasks – such as for example movies, or the calculation of a bar pattern speed – the output frequency needs to be quite high and this requires considerable disc space per run.

ii) For analyses necessitating very frequent output, it is possible to perform an on-line analysis, introduced initially by Athanassoula et al. (1998). In these cases, a snapshot is produced as output in a specific directory and a flag is raised, so that another processor accesses the data and executes a predetermined task, after which it erases the snapshot. This leads to a very considerable saving of disc space, since full snapshots can be saved at much sparser time-steps. The only disadvantage is that one has to anticipate the analysis that will be necessary.

iii) Manipulators provide a closer link to the simulation data because they permit interaction. During the simulation, data are transferred to a specific program, called a manipulator, whose name and properties are set by the appropriate *falcON* keyword, and which manipulates them and then returns them to the simulation code. During this time, the execution of the simulation is stopped. Manipulators have a very large number of applications. They can, for example, be used to axisymmetrise the disc component of a spiral galaxy every so many time-steps in order to artificially avoid the growth of a bar (e.g. Machado & Athanassoula 2010). In this case, the positions and velocities of the particles are output to the manipulator program, which randomizes the azimuthal coordinates, brings the corresponding changes to the velocities, and then returns the new coordinates to the simulation program to continue the simulation with these new coordinates. This concept is quite powerful since it allows a user to make certain changes to a program (such as add new particles to simulate accretion[8], impose a given symmetry when building initial conditions etc.) without actually going into the code itself. In its simplest form, it cannot change the phase space coordinates and masses of the particles, thus can be used to make the on-line analysis. This most useful feature is an integral part of the *falcON* code.

6.3.3. NEMO

A very considerable part of any simulation research project is taken by the writing and the debugging of the appropriate

[8] It is of course possible to add particles during the simulation when using GADGET-2 as well, but at the cost of a higher programming complexity.

analysis programs. The NEMO toolbox was devised so as to focus all of these efforts, thus avoiding everybody having to write his/her own version of the basic programs. NEMO (Teuben 1995) is an extendible stellar dynamics toolbox, which follows an Open-Source Software model. It includes a large variety of programs to create, integrate, analyze, and visualize *N*-body and SPH-like systems, as well as operate on images, tables, and orbits. Since the output of *falcON* is in NEMO format, a user of *falcON* also has available to him/her this considerable library of programs. This is particularly important for users starting with simulations.

7. Conclusions

N-body simulations are now commonly used for galactic dynamics studies, by experienced, but also by less experienced users. To help in particular the latter group of users in choosing amongst a number of possible codes, we have performed timing and memory comparisons of four different codes, three of which are publicly available, while the fourth one (FMB) has been widely used by one of us (PF) in different contexts. These are implementations of three different algorithms, namely the Barnes-Hut tree-code, the FMM, and W. Dehnen's algorithm. We made the comparisons using simple particle distributions often used in modelling galaxies. We thus use Plummer and power-law spheres, disc galaxies with or without a bar component, and groups of Plummer spheres, which were intended to represent a zeroth order model of a galaxy group. In our comparisons, we used the publicly available versions of all codes. For both *falcON* and Gadget, private versions with superior performances are also available, but we did not consider them since they are not available to all users.

As far as serial executions are concerned, Dehnen's algorithm implemented in the *falcON* code is clearly the fastest and in many cases by a wide margin. It is, furthermore, much less sensitive to the particle distribution concentration than the other codes and its computational costs depend roughly linearly on the number of particles.

The second fastest is the GADGET-2 code, which implements the Barnes-Hut algorithm with specific acceptance criteria. For concentrated configurations, it is the RC criterion that provides the fastest execution, but the difference diminishes with decreasing concentration and the trend is reversed for uniform distributions. However, GADGET-2 is a parallel code, thus outperforms *falcON* if a sufficient number of processors is available. This number depends on the configuration and, if the appropriate acceptance criterion is used, it varies between three and ten in all examples presented here. Parallel codes such as GADGET-2 and FMB, also enable us to simulate larger distributions since more memory is available.

A first MPI parallelization of Dehnen's algorithm has been presented in Londrillo et al. (2002), but is based on a complete rewriting of the algorithm in Fortran 90, not on the highly optimized C++ *falcON* code. Dehnen's algorithm is indeed hard to parallelize, especially on distributed memory architecture where the communication scheme is hardly predictable because of the dual recursive tree walk formulation and the dynamic acceptance criterion used. Although limited to the number of cores on one single node, a multi-thread parallelization in shared memory may be more successful and would be very useful, because multi-core machines are now standard, even as desktop computers.

The implementation of the fast multipole method in FMB is faster than the implementation of the Barnes-Hut algorithm in NEMO, as long as the particle distribution is not too concentrated. For a sufficient number of processors, the parallel FMB code also outperforms the serial *falcON* code. The number of processors necessary can, however, be very large and reach as high as 64 for the most highly concentrated distributions considered here. The hybrid MPI-thread parallelization of FMB enables a gain in parallel efficiency as well as in memory scalability over a pure MPI code such as GADGET-2. As the number of cores increases in present-day processor architectures, this characteristic will become even more important in the future.

To summarize, we recommend the use of *falcON* for purely *N*-body simulations with a particle number such that a single processor can perform the simulation in reasonable time. *falcON* has a number of additional advantages, namely that it conserves linear momentum, includes the possibility of using manipulators, and allows the user to rely on the NEMO toolkit for the analysis of the simulations. For a larger number of particles, or if a number of processors are available, or for studies including gas and its physics, we recommend GADGET-2. FMB provides the highest parallel efficiencies, but is considerably slower in terms of execution time. Furthermore, it is ill-suited to the concentrated distributions often encountered in galactic work.

Acknowledgements. We thank A. Bosma, O. Coulaud, W. Dehnen, S. Rodionov and V. Springel for useful disussions. This work was partially supported by grant ANR-06-BLAN-0172.

References

Athanassoula, E. 2003, MNRAS, 341, 1179
Athanassoula, E., Bosma, A., Lambert, J. C., & Makino, J. 1998, MNRAS, 293, 369
Barnes, J. E., & Hut, P. 1986, Nature, 324, 446
Binney, J., & Tremaine, S. 2008 (Princeton Univ. Press)
Capuzzo-Dolcetta, R., & Miocchi, P. 1998, J. Comput. Phys., 143, 29
Cheng, H., Greengard, L., & Rokhlin, V. 1999, J. Comput. Phys., 155, 468
Coulaud, O., Fortin, P., & Roman, J. 2007, ISPDC, 391
Coulaud, O., Fortin, P., & Roman, J. 2008, J. Comput. Phys., 227, 1836
Coulaud, O., Fortin, P., & Roman, J. 2010, Math. Comput. Model., 51, 177
Dehnen, W. 2000, ApJ, 536, L39
Dehnen, W. 2002, J. Comput. Phys., 179, 27
Hockney, R. W., & Eastwood, J. W. 1988 (Institute of Physics Publishing)
Knebe, A., Green, A., & Binney, J. 2001, MNRAS, 325, 845
Krishnan, S., & Kale, L. V. 1995, Int. Conf. on Parallel Processing, 46
Londrillo, P., Nipoti, C., & Ciotti, L. 2002, Computational astrophysics in Italy: methods and tools, Mem. della Soc. Astron. It. Suppl., 1, 18
Machado, R., & Athanassoula, E. 2010, MNRAS, 406, 2386
Nabors, K., Korsmeyer, F. T., Leighton, F. T., & White, J. 1994, SIAM J. Sci. Comput., 15, 713
Springel, V. 2005, MNRAS, 364, 1105
Teuben, P. J. 1995, Astronomical Data Analysis Software and Systems IV, PASP Conf Ser., 77, 398