RAA 2016 Vol. **16** No. **1**, 11 (12pp) doi: 10.1088/1674–4527/16/1/011
http://www.raa-journal.org *http://iopscience.iop.org/raa*

Research in Astronomy and Astrophysics

Performance analysis of parallel gravitational *N*-body codes on large GPU clusters

Si-Yi Huang[1], Rainer Spurzem[1,2,4] and Peter Berczik[1,3,4]

[1] National Astronomical Observatories and Key Laboratory of Computational Astrophysics, Chinese Academy of Sciences, Beijing 100012, China; *huang41@nao.cas.cn*

[2] Kavli Institute for Astronomy and Astrophysics, Peking University, Beijing 100871, China

[3] Main Astronomical Observatory, Ukrainian National Academy of Sciences, 03680 Kiev, Ukraine

[4] Astronomisches Rechen-Institut, Zentrum für Astronomie, Universität Heidelberg, 69120 Heidelberg, Germany

Received 2015 April 23; accepted 2015 August 7

Abstract We compare the performance of two very different parallel gravitational *N*-body codes for astrophysical simulations on large Graphics Processing Unit (GPU) clusters, both of which are pioneers in their own fields as well as on certain mutual scales - `NBODY6++` and `Bonsai`. We carry out benchmarks of the two codes by analyzing their performance, accuracy and efficiency through the modeling of structure decomposition and timing measurements. We find that both codes are heavily optimized to leverage the computational potential of GPUs as their performance has approached half of the maximum single precision performance of the underlying GPU cards. With such performance we predict that a speed-up of $200-300$ can be achieved when up to 1k processors and GPUs are employed simultaneously. We discuss the quantitative information about comparisons of the two codes, finding that in the same cases `Bonsai` adopts larger time steps as well as larger relative energy errors than `NBODY6++`, typically ranging from $10-50$ times larger, depending on the chosen parameters of the codes. Although the two codes are built for different astrophysical applications, in specified conditions they may overlap in performance at certain physical scales, thus allowing the user to choose either one by fine-tuning parameters accordingly.

Key words: methods: analytical — methods: data analysis — methods: numerical

1 INTRODUCTION

Algorithms for gravitational *N*-body simulations, which are widely used tools in astrophysics nowadays, have mainly evolved into two categories over previous decades. Traditionally, by computing the pairwise force among particles, the direct summation method has been employed as the core idea of the so-called "direct N-body code." High accuracy can be archived by choosing smaller time steps, with higher computational costs. Some of the best-known examples are the `NBODY` series of codes developed by Aarseth (1999) and the `Starlab` environment developed by Hut, McMillan, Makino, Portegies Zwart, et al. (Hut 2003). Alternatively, the force calculation can be approximated with certain assumptions. In the late 1960s, some approximation algorithms such as tree-code or mesh-code were developed in an attempt to reduce the computational complexity so that larger simulations could be scaled on limited hardware with acceptable time. One of the most prominent approximation algorithms, namely the Barnes-Hut tree (Barnes & Hut 1986), also has many implementations such as `GADGET` developed by Springel (2005) and `PEPC` developed by Gibbon, Winkel and collaborators (Winkel et al. 2012) .

Since direct summation methods use an all-to-all particle force direct summation method, they have a raw computational complexity of $\mathcal{O}(N^2)$. Additional algorithms have been developed to cut the absolute wall-clock time down in spite of the prohibitive asymptotic complexity. On the other hand, approximation schemes reduce complexities to $\mathcal{O}(N \log N)$ or even $\mathcal{O}(N)$ thanks to the approximate treatments of force computations and some special structures such as octrees or grids. However, such intelligent approximation algorithms may not be very suitable for the simulation of certain astronomical systems such as dense star clusters, e.g. globular clusters or nuclear star clusters with or without central massive black holes. This is because in these systems two-body relaxation is important, which can only be correctly modeled by following pairwise particle interactions with high precision at large distances. They require the use of direct summation methods, which have so far experienced great difficulties reaching even one million particles (but see Wang et al. 2015). As such, the only practical approach at present to handle simu-

lations with ultra-high particle numbers (e.g. cosmological structure formation) is through the employment of approximation methods, despite their lack of resolution at small scales (Shin et al. 2014; Genel et al. 2014; Vogelsberger et al. 2014).

Consequently, parallel technologies are applied for *N*-body simulations as a proper solution. With the help of parallel hardware, simulations could theoretically be accelerated multifold in accordance with the number of processors that are invoked. In practice parallel schemes have been implemented, and performance analysis of parallel *N*-body codes on supercomputers or distributed systems have been conducted, such as the work by Gualandris et al. (2007). Supercomputer clusters have been used for the parallelization of *N*-body simulations, and special devices were also added in order to process the computationally intensive sections. In the beginning, special-purpose architectures called GRAPE series (Makino et al. 2003) were exclusively designed for *N*-body simulations, which achieved speed-ups by putting the whole force calculations into hardware that placed many pipelines on one chip; detailed performance of GRAPE was measured by Harfst et al. (2007). In recent years, with the rapid development of innovative hardware manufacturing techniques, Graphics Processing Units (GPUs) as general-purpose devices are used more and more and play the same role as GRAPE. Now architectures consisting of many processors that are equipped with corresponding GPUs are prevalent in studies that use *N*-body simulations.

Parallel *N*-body simulation software running partially or even entirely on GPUs was subsequently developed (Berczik et al. 2011, 2013; Spurzem et al. 2012; Bédorf et al. 2012a,b), but in practical applications the performance would not measure up to ideal speed-up because of some inevitable serial code in the code structure. The actual speed-up is limited by sequential fractions in codes and is not directly proportional to the number of processor cores; the theoretical maximum value can be predicted by *Amdahl's law* (Amdahl 1967). Moreover, this peak value is unapproachable on account of communication overhead between multiple processors. The effectiveness of either parallelization or GPU acceleration introduced in *N*-body software is not intuitive but is still interesting.

In this paper, we focus on the performance analysis of two kinds of *N*-body software, the direct *N*-body code NBODY6++ and the tree-code Bonsai, which can both be executed in parallel and accelerated by GPUs. Section 2 describes an overview of the software and hardware we used. Section 3 describes the performance models used to analyze complicated *N*-body codes, and provides detailed measurements, performance results and reasonable predictions. Section 4 describes the performance comparisons and analysis, then makes a conclusion which gives us a better reference regarding the choice of an opportune scheme for software type and hardware scale in *N*-body simulations.

2 SOFTWARE AND HARDWARE

2.1 Direct *N*-body Implementation: NBODY6++

In this subsection we provide a brief description of NBODY6++, which we used for the performance analysis of direct *N*-body code.

NBODY6++, developed by Rainer Spurzem, is a parallel version of NBODY6 (Spurzem 1999; Khalisi et al. 2003; Spurzem et al. 2008). The standard NBODY6 is the $6^{\rm th}$ generation of NBODY code initiated by Sverre Aarseth, who has a lifelong dedication to the development of the family of NBODY series of codes (Aarseth 1999). The first code NBODY1 was a basic direct *N*-body code with individual time steps. The Ahmad-Cohen neighbor scheme (Ahmad & Cohen 1973) that was used in NBODY2 and NBODY5 made it possible to treat larger systems. Kustaanheimo & Stiefel (1965) two-body regularization and chain regularization were applied in NBODY3 and NBODY5 to deal with close encounters. By the time NBODY6 had been developed, the code included both the neighbor scheme and regularizations, as well as applying the Hermite scheme integration method combined with hierarchical block time steps.

NBODY6++ is a descendant of the standard NBODY6. It kept those features of its predecessor mentioned above as well as increased the efficiency by redesigning the algorithms to be suitable for parallel hardware. NBODY6++ used the Single Program Multiple Data (SPMD) scheme to achieve parallelism. In this mode multiple autonomous processors simultaneously start with chunked local data and then communicate with each other through the *copy algorithm* (Makino 2002; Dorband et al. 2003), which is a parallelized algorithm that assumes each processor has a local copy of the whole system and every processor handles the subgroup of data itself then immediately broadcasts the new data to all the other processors. The parallelization scheme of NBODY6++ is implemented with the standard MPI library package.

The most major improvement of NBODY6++ is the parallelization of regular and irregular force computations, which were special concepts introduced from the Ahmad-Cohen neighbor scheme that divided the full force of each particle involved by all the other particles into two parts: one part called irregular force that has frequent but short time steps for interactions with adjacent particles, and another one called regular force which has longer time steps for full interactions. By assigning the sections with the most expensive overhead to multiple processors, NBODY6++ achieved the expected efficiency. Moreover, the computationally heavier regular force calculation part was adapted for GPU acceleration using CUDA. The performance of parallel accelerations is described in Section 3.1.

2.2 *N*-body Tree-code Implementation: `Bonsai`

In this subsection we provide a brief description of `Bonsai`, which we used for the performance analysis of *N*-body tree-code.

The tree-code algorithm, a widely used method nowadays for *N*-body simulations, was originally introduced by Barnes & Hut (1986). This algorithm reduces the computational complexity of an *N*-body simulation from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$, therefore improving the simulation scale compared to brute force methods. Here `Bonsai`, a parallel GPU tree-code implementation developed by Jeroen Bédorf, Evghenii Gaburov and Simon Portegies Zwart (Bédorf et al. 2012a,b), is a suitable representative of the gravitational *N*-body tree-code that was developed in recent years.

Certain schemes are introduced in `Bonsai` to ensure the high efficiency of the code (Bédorf et al. 2012a). A sparse octree is used as the data structure, which means the structure is the three-dimensional extension of a binary tree where tree-cells are not complete and equal, and it is based on the underlying particle distribution. The tree is constructed layer-by-layer from top to bottom, and inverted with respect to the direction in the traversal process. Tree-cell properties are updated during the steps, and the integration into the simulation process is advanced. The depth of tree traversal crucially affects both accuracy and time consumption, which is determined by the multipole acceptance criterion (MAC) in the tree-code. The criterion is described as follows,

$$d > \frac{l}{\theta} + \delta, \tag{1}$$

where d is the smallest distance between a group and the cell's center-of-mass, l is the length of the cell, δ is the distance between the cell's center-of-geometry and the center-of-mass, and θ is an opening angle parameter to control the accuracy. If the inequality is satisfied then the traversal process will be interrupted and the multipole moment will be used.

Like other existing *N*-body codes, `Bonsai` uses a parallel technique to reach large scale or high resolution simulations, and applies GPUs to speed up force computations. In contrast to those GPU tree-codes, `Bonsai` executes all parts of the algorithm on GPUs, to avoid the bottlenecks generated from CPU-GPU communications. In Section 3.2 the performance of the main parts of the code is presented.

2.3 Hardware Environments and Initial Conditions

The supercomputer we mainly used for the tests of both software presented above is an IBM iDataPlex Cluster JUDGE named "The Milky Way System" partition provided and maintained by the Jülich Supercomputing Centre in Germany, which is a dedicated GPU cluster using two Intel Xeon X5650 6-core processors and two NVIDIA Tesla M2050/M2070 GPU cards in every node, with 206 computing nodes and a peak performance of 239 Teraflops.

Our performance measurements involved two different kinds of parallel gravitational GPU-accelerated *N*-body codes: `NBODY6++` and `Bonsai`. The initial conditions of all tests of both codes are consistent with each other, starting with the Plummer model and running over one standard *N*-body Time Unit. The number of particles ranges from $N = 2^{13}(8\mathrm{k})$ to $2^{20}(1\mathrm{M})$, doubling the number over successive intervals. There are additional tests using larger numbers of particles, up to $N = 2^{24}(16\mathrm{M})$ in `Bonsai` code runs. The number of processors we chose is a series of increasing numbers $N_\mathrm{p} = 1, 2, 4, 8, 16, 32$. Other parameters which are necessary but specific only in each code, such as the time step factor for regular/irregular force polynomial and desired optimal neighbor number in `NBODY6++`, or the accuracy control parameter θ and softening value ϵ in `Bonsai`, will be described in detail in Section 3.

3 PERFORMANCE

In this section we evaluated the performance of these two GPU-based parallel *N*-body simulation codes (i.e. `NBODY6++` and `Bonsai`) which we tested mainly on the Jülich Dedicated GPU Environment described in Section 2.3.

In spite of a vast difference between the two codes derived from their own fundamental algorithms and specific details, which makes it difficult to give a one-to-one comparison, there are some global values providing sufficient information. Timing variables, speed-up and hardware performance indicators like speed and bandwidth were measured and are described below for performance analysis of the codes.

3.1 Performance of `NBODY6++`

`NBODY6++`, a parallel direct gravitational *N*-body code, is featured with a couple of elegant algorithms and schemes developed and maintained over the past few decades. The procedures that we used enabled more realistic size of simulations running in achievable circumstances while increasing sophistication as well. As a consequence, we present a performance model for analyzing the overall behavior as well as the main components of the code. Through this model we will have a better idea about the performance of a typical direct *N*-body code and predictions about the behavior of the code on larger scales.

3.1.1 Performance model

We measured running time directly for evaluating performance and modeling. In `NBODY6++`, the total wall-clock time T_total required to advance the simulation for a certain integration interval can be written as

$$T_\mathrm{tot} = T_\mathrm{force} + T_\mathrm{comm} + T_\mathrm{host}, \tag{2}$$

where $T_\mathrm{force} = T_\mathrm{reg} + T_\mathrm{irr} + T_\mathrm{pre}$ is time spent on both host and device involving force calculations; here T_reg, T_irr

and $T_{\rm pre}$ are time spent on force computations of regular time steps, irregular time steps and prediction respectively; $T_{\rm comm} = T_{\rm mov} + T_{\rm mci} + T_{\rm mcr} + T_{\rm syn}$ is time spent on data moving for parallel components, MPI communications after regular and irregular blocks and synchronizations of processors; $T_{\rm host}$ is time spent on the host side which is a completely sequential process. All of the time variables are measured directly by standard Fortran functions `ETIME` in sequential mode and `MPI_WTIME` in parallel mode. All of the descriptions are listed in the glossary (Table 1).

According to the decomposition described above we broke down the code structure and measured these main sections which have heavy weights in the code. Owing to a large amount of variables and the high complexities, some insignificant components in `NBODY6++` are not counted. For every part to be analyzed we listed the expected scaling and optimal fitting value in Table 2, which are obtained from the structure of code implementation, chronograph and fitting functions. A python function `scipy.optimize.curve_fit` is used to obtain the optimal fitting value, which is based on non-linear least squares.

The conception of speed-up is used for evaluating the parallelism of the code. There are a couple of definitions of speed-up with different ranges. The ideal maximum speed-up $S_i = N_{\rm p}$ will never be achievable, where $N_{\rm p}$ is the number of processors used. Unreachable as well, but a more reasonable indicator to predict the theoretical maximum speed-up, is the so-called *Amdahl's law*, which is defined as

$$S_{\rm a}(N_{\rm p}) = \frac{T(1)}{T(N_{\rm p})} = \frac{1}{(1-X) + \frac{X}{N_{\rm p}}}\,, \tag{3}$$

where X is the fraction of the algorithm that can benefit from parallelization. In practice there is another experiential speed-up to be measured through timer recording, which is given by

$$S_{\rm e}(N_{\rm p}) = \frac{T_{\rm tot}(1)}{T_{\rm tot}(N_{\rm p})}\,, \tag{4}$$

where $T_{\rm tot}(1)$ and $T_{\rm tot}(N_{\rm p})$ are both the measured values of actual running time. By combining the fitting values into the speed-up formula we will have a general overall behavior of the code, by which we can make a prediction accordingly about the code performance in larger scale simulations.

The speed of force calculation is measured by the extent at which the program reaches the peak of computing devices. Here in our tests the computing device particularly refers to the NVIDIA Tesla M2050/M2070 GPU cards, which feature up to 1030 Gigaflops of single precision floating point performance and 515 Gigaflops of double precision floating point performance per card.

In `NBODY6++`, as the total force is divided into two parts, we used two speed variables $P_{\rm reg}$ and $P_{\rm irr}$ to represent regular and irregular force calculating speed respectively, which are written as

$$\begin{aligned} P_{\rm reg} &= \frac{N_{\rm reg\_tot}}{T_{\rm reg}} = \frac{N_{\rm reg} \times N \times \gamma_{\rm h4}}{T_{\rm reg}}\,, \\ P_{\rm irr} &= \frac{N_{\rm irr\_tot}}{T_{\rm irr}} = \frac{N_{\rm irr} \times \langle N_{\rm nb} \rangle \times \gamma_{\rm h4}}{T_{\rm irr}}\,, \end{aligned} \tag{5}$$

where $N_{\rm [reg|irr]\_tot}$ is the total floating point operations of regular/irregular force computations, $N_{\rm [reg|irr]}$ is the cumulative number of regular/irregular time steps, $\langle N_{\rm nb} \rangle$ is the average number of integrated "neighbor" particles, and $\gamma_{\rm h4}$ defines the floating point operation counts of the fourth-order Hermite scheme per particle per interaction per step, from the work Nitadori & Makino (2008) which is a constant value of $\gamma_{\rm h4} = 60$.

Bandwidth is measured as a part of hardware performance along with computing speed (P). In `NBODY6++`, we defined bandwidth ($B_{\rm reg},\ B_{\rm irr}$) as

$$\begin{aligned} B_{\rm reg} &= \frac{N_{\rm mcr}}{T_{\rm mcr}} = \frac{8 \times (41 + l_{\rm max}) \times N/N_{\rm p}}{T_{\rm mcr}}\,, \\ B_{\rm irr} &= \frac{N_{\rm mci}}{T_{\rm mci}} = \frac{8 \times 19 \times \langle N_{\rm act} \rangle / N_{\rm p}}{T_{\rm mci}}\,, \end{aligned} \tag{6}$$

where $N_{\rm [mcr|mci]}$ is the number of bytes transferred during MPI communication after regular/irregular blocks, constant terms (in Eq. (6), i.e. 8, 41, 19) derived from the size of datasets transferred, in which $l_{\rm max}$ is the maximum size of neighbor lists set manually. Detailed results of all these performance indicators are presented in the next subsection.

3.1.2 Performance results

The measured total wall-clock time of `NBODY6++` is shown in Figure 1.

On the whole, the result shows a good extensibility and acceleration when using more numbers of processors. To be specific, we assign a different weight to each part of the code. Among all of the parts, the time spent for force computations always has the highest value, therefore both regular and irregular force computations and prediction have been implemented with the parallel algorithm and decrease rapidly when code is run using multi-processors. Here in the heaviest part $T_{\rm force}$, the regular force computation $T_{\rm reg}$, which takes the highest fraction of computing time in the former versions of the code, has been accelerated and implemented on the specific device (GPU), as a consequence of causing a significant reduction of the whole running time costs. Other parts are currently executed on the CPU side.

Table 2 shows the main components of `NBODY6++` as a function of N and $N_{\rm p}$. As described above, for every part the expected scaling is evaluated by the code structure, and the fitting value is based on experimental data. The fitting process includes two steps by fitting N and $N_{\rm p}$ successively but independently. Firstly we used a minimum of fixed $N_{\rm p}$ to avoid the influence of processor number and

Table 1 Glossary

Variable	Description
GENERAL	
N	total number of particles
$N_{\rm p}$	number of processors
$S_{\rm a}$	theoretical maximum speed-up defined by Amdahl's law
S_i	ideal maximum speed-up equal to $N_{\rm p}$
$S_{\rm e}$	experiential speed-up equal to the ratio between measured time of single and multiple processor numbers
P	force computation speed of floating point operations per second
B	bandwidth of bytes of data transfer per second
$T_{\rm tot}$	total wall-clock time
kn_x, kp_x	quantitative factors for fitting the result of certain parts; $k[n\|p]$ implies the factor only depends on $N\|N_{\rm p}$, subscript x indicates different parts
ΔE	relative energy error
Δt	time step interval of integration
NBODY6++	
$\langle N_{\rm act}\rangle$	average number of integrated active particles
$\langle N_{\rm nb}\rangle$	average number of neighbors
$N_{\rm irr}$	cumulative number of irregular time steps
$N_{\rm reg}$	cumulative number of regular time steps
$\gamma_{\rm h4}$	floating point operation counts per particle per interaction per step
$T_{\rm comm}$	sum of communication time
$T_{\rm force}$	sum of force computation time
$T_{\rm host}$	time spent on the host side
$T_{\rm irr}$	neighbor (irregular) force computation time
$T_{\rm mci}$	MPI communication after irregular blocks
$T_{\rm mcr}$	MPI communication after regular blocks
$T_{\rm mov}$	time spent on data moving for parallel runs
$T_{\rm pre}$	particle prediction time
$T_{\rm reg}$	full (regular) force computation time
$T_{\rm syn}$	interprocessor synchronization time
Bonsai	
$N_{\rm force}$	cumulative number of interactions
$\gamma_{\rm t}$	floating point operation counts per particle per interaction per step
$T_{\rm build}$	time spent on building tree structure
$T_{\rm comm}$	sum of communication time
$T_{\rm corr}$	particle correction time
$T_{\rm dom}$	time spent on updating the particle domain
$T_{\rm ene}$	energy check time
$T_{\rm exch}$	time spent on particle exchange
$T_{\rm force}$	force computation time
$T_{\rm grp}$	time spent on setting active groups
$T_{\rm pre}$	local tree prediction time
$T_{\rm prop}$	node properties computation time
$T_{\rm sort}$	sorting and data-reordering time
$T_{\rm syn}$	interprocessor synchronization time
$T_{\rm tree}$	sum of the whole tree construction time
ϵ	softening to diminish the effect of graininess
θ	opening angle to control the accuracy

obtained the experimental scaling of N. The fitting values with N are obtained under circumstances which use increasing particle numbers and a fixed single processor ($N_{\rm p} = 1$), while for cases of multi-processor-related values (i.e. $T_{\rm mci}$, $T_{\rm mcr}$ and $T_{\rm syn}$, which have no numbers in single processor runs) the number of processors changed to $N_{\rm p} = 2$. The fitting values with $N_{\rm p}$ are obtained by the second step. The datasets are grouped according to N, then every group of data is divided by each N dependent function to get the fitting values with $N_{\rm p}$. The fitting results of

Table 2 Main Components of NBODY6++

Description	Timing variable	Expected scaling N	Expected scaling $N_{\rm p}$	Fitting value [s]
Regular force computation	$T_{\rm reg}$	$\mathcal{O}(N_{\rm reg} \times N)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(2.2 \times 10^{-9} \times N^{2.11} + 10.43) \times N_{\rm p}^{-1}$
Irregular force computation	$T_{\rm irr}$	$\mathcal{O}(N_{\rm irr} \times \langle N_{nb} \rangle)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(3.9 \times 10^{-7} \times N^{1.76} - 16.47) \times N_{\rm p}^{-1}$
Prediction	$T_{\rm pre}$	$\mathcal{O}(N^{kn_p})$	$\mathcal{O}(N_{\rm p}^{-kp_p})$	$(1.2 \times 10^{-6} \times N^{1.51} - 3.58) \times N_{\rm p}^{-0.5}$
Data moving	$T_{\rm mov}$	$\mathcal{O}(N^{kn_{m1}})$	$\mathcal{O}(1)$	$2.5 \times 10^{-6} \times N^{1.29} - 0.28$
MPI communication (Reg.)	$T_{\rm mcr}$	$\mathcal{O}(N^{kn_{cr}})$	$\mathcal{O}(kp_{cr} \times \frac{N_{\rm p}-1}{N_{\rm p}})$	$(3.3 \times 10^{-6} \times N^{1.18} + 0.12)(1.5 \times \frac{N_{\rm p}-1}{N_{\rm p}})$
MPI communication (Irr.)	$T_{\rm mci}$	$\mathcal{O}(N^{kn_{ci}})$	$\mathcal{O}(kp_{ci} \times \frac{N_{\rm p}-1}{N_{\rm p}})$	$(3.6 \times 10^{-7} \times N^{1.40} + 0.56)(1.5 \times \frac{N_{\rm p}-1}{N_{\rm p}})$
Synchronization	$T_{\rm syn}$	$\mathcal{O}(N^{kn_s})$	$\mathcal{O}(N_{\rm p}^{kp_s})$	$(4.1 \times 10^{-8} \times N^{1.34} + 0.07) \times N_{\rm p}$
Sequential parts on host	$T_{\rm host}$	$\mathcal{O}(N^{kn_h})$	$\mathcal{O}(1)$	$4.4 \times 10^{-7} \times N^{1.49} + 1.23$

Notes: Detailed descriptions of the symbols that are used are listed in Table 1.

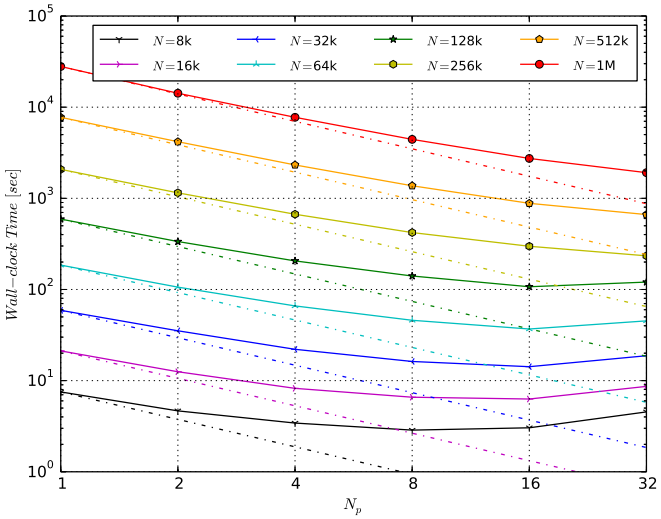

Fig. 1 Total wall-clock time ($T_{\rm tot}$) of NBODY6++ as a function of N and $N_{\rm p}$. Solid lines are the measured values of running time, and dashed lines are the ideal acceleration by increasing processor numbers. (The unit symbols in the legend have the magnitudes: $1{\rm k} = 1024$, $1{\rm M} = 1{\rm k}^2$ and $1{\rm G} = 1{\rm k}^3$, and similarly hereinafter.)

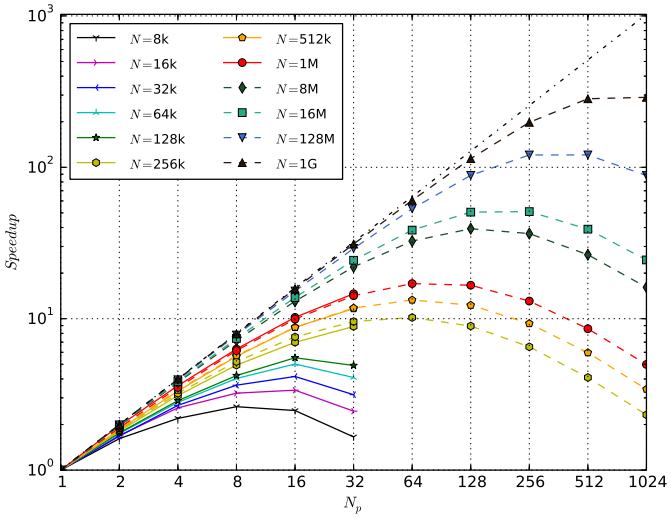

Fig. 2 The speed-up (S) of NBODY6++ as a function of N and $N_{\rm p}$. Solid lines are the measured speed-up ratio between sequential and parallel wall-clock time. Dashed lines are the predicted performance of larger scale simulations.

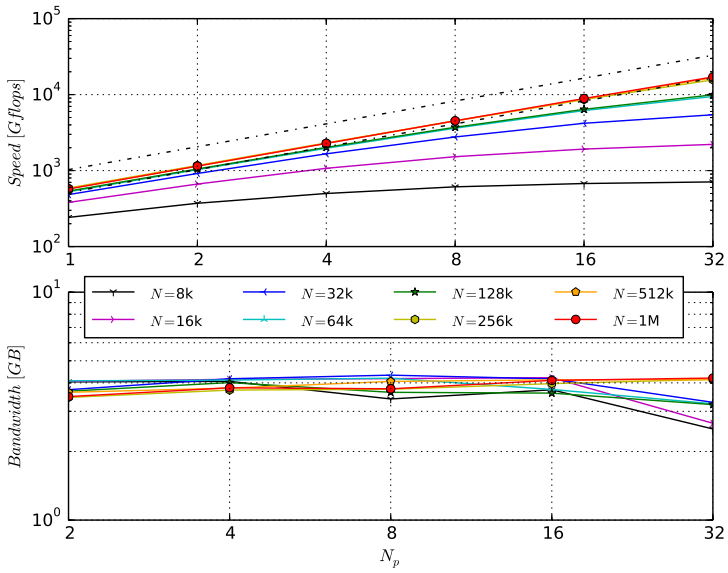

Fig. 3 Hardware performance of NBODY6++ running on the "Milky Way" GPU cluster. The upper panel corresponds to the regular force computation speed ($P_{\rm reg}$), where two dashed lines refer to the peak single and double precision floating point performance. The lower panel corresponds to the bandwidth of the regular part ($B_{\rm reg}$).

every main part are listed in the last column. Considering the expected scaling value of the main parts, as $T_{\rm mov}$, $T_{\rm syn}$ and $T_{\rm host}$ have no significant and direct scaling with N from the code structure while $T_{\rm pre}$ is made up of two prediction branches that are determined by N in the next time step, we expect these values to follow a simple exponential form for N. $N_{\rm reg}$, $N_{\rm irr}$ and $\langle N_{nb}\rangle$, which are used in $T_{\rm reg}$ and $T_{\rm irr}$, are values which are completely dependent on N as $N_{\rm reg} \propto N^{1.18}$, $N_{\rm irr} \propto N^{1.10}$ and $\langle N_{nb}\rangle \propto N^{2/3}$ respectively, so their fitting values are also combined together in an exponential form for N. At last, unified exponent forms for N and $N_{\rm p}$ are used in the last column rather than other symbols used in the middle column.

By taking the fitting values into the definition of experimental speed-up, we give the prediction about the performance of NBODY6++, which is shown in Figure 2. As a result, the optimal value of $N_{\rm p}$ needed for larger simulations of different scales is shown clearly in the figure.

Figure 3 shows the hardware performance of NBODY6++ in the actual environment on a real GPU accelerated cluster. Because GPUs played the central role in acceleration, we focus on parts related to GPUs in NBODY6++, so $P_{\rm reg}$ and $B_{\rm reg}$ were drawn in the figure. For the figure of $P_{\rm reg}$, two dashed lines, i.e. peak single and double precision floating point performance, are used as the baseline, and the computation speeds of a different group of N runs are increasingly closer to the peak when N is doubled. In the whole NBODY6++ data structure there are two types of precision used in the respective parts. Double precision type is used in main loops of the code which is declared as the type "REAL*8" in the global header file, while for the part that computes regular force which is accelerated by GPU, all of the data are converted to the type "REAL*4" and single precision is used in all relevant parts of the CUDA routine. Therefore a mixed precision data structure is used in the regular part of NBODY6++- double precision in the data moving process and single precision in the data computation process. As the process of computation in the GPU card dominates the regular part, we use single precision to make a comparison. As shown in the figure, the force computation speed of large N runs exceeded over half of the maximum single precision performance (for instance, $P_{\rm reg}$ for 1M particle runs has the values of $530 \sim 570$ Gflops per M2050/M2070 GPU card.) This proportion concurs with the results of Berczik et al. (2011, 2013), who claimed the speed performance of another direct *N*-body code, ϕ-GPU, reached the values $\propto 550$ Gflops per C2050 GPU card and $\propto 1.48$ Tflops per K20 GPU card; both results approached half of the single precision performance peak. Considering the hardware architecture, as operations among various registers and parts of the memory cause extra inevitable time consumption, the proportion is acceptable in practical environments. For calculating $B_{\rm reg}$ when ignoring the "dropping" points, others remained at the level of more than 80% of the maximum bandwidth performance.

3.2 Performance of Bonsai

3.2.1 Performance model

Similar to Equation (2), in Bonsai the total wall-clock time $T_{\rm tot}$ can be written as

$$T_{\rm tot} = T_{\rm tree} + T_{\rm force} + T_{\rm comm} + T_{\rm other}\,, \qquad (7)$$

where $T_{\rm tree} = T_{\rm sort} + T_{\rm build} + T_{\rm prop} + T_{\rm grp}$ is time spent on tree building, which mainly includes sorting and reordering of the particles along a 1D number string mapped along

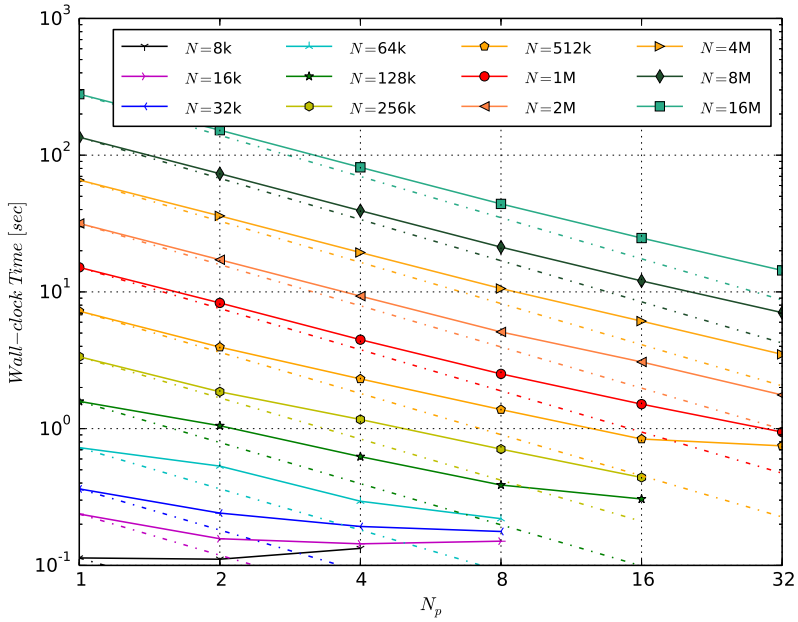

Fig. 4 Total wall-clock time ($T_{\rm tot}$) of `Bonsai` as a function of N and $N_{\rm p}$. The legend is the same as Figure 1. Opening parameters representing the initial conditions are set as $\Delta t = 0.0625$, $\epsilon = 0.01$ and $\theta = 0.5$.

a *Space Filling Curve*, tree structure construction, computation of tree-node properties and setting active groups for following steps; $T_{\rm force}$ is time spent on force computations in tree-traversal; $T_{\rm comm} = T_{\rm dom} + T_{\rm exch} + T_{\rm syn}$ is time spent on distributing or redistributing the particles between processors and synchronizations of processors; $T_{\rm other} = T_{\rm pre} + T_{\rm corr} + T_{\rm ene}$ is time consumptions for other mainly essential parts, like local-tree predictions before tree construction, corrections after force computations and energy check. All of the time variables are measured by the CUDA C function `cuEventElapsedTime` from the CUDA Event Management Driver API and entirely counted on the device (GPU). Trivial time consumptions on the host side are ignored. The entire descriptions are listed in the glossary (Table 1).

In terms of hardware performance, different from Equation (5), the force calculating speed in `Bonsai` is written as

$$P_{\rm force} = \frac{N_{\rm force\_tot}}{T_{\rm force}} = \frac{N_{\rm force} \times \gamma_{\rm t}}{T_{\rm force}}, \qquad (8)$$

where $N_{\rm force\_tot}$ is the total number of floating point operations; $N_{\rm force}$ is the cumulative number of interactions; $\gamma_{\rm t}$ is the number of operation counts for each interaction in the tree-code, for which we used a constant value of $\gamma_{\rm t} = 38$ from the works Warren & Salmon (1992); Kawai et al. (1999); Hamada et al. (2009); Hamada & Nitadori (2010), and the resulting plot is shown in Figure 6. Note that Bédorf et al. (2014) used other separate values for operation counts, which are 23 and 65 for particle-particle and particle-cell interactions respectively.

$$P = \frac{N_{\rm pp\_tot} + N_{\rm pc\_tot}}{T_{\rm force}} = \frac{\gamma_{\rm pp} \times N_{\rm pp} + \gamma_{\rm pc} \times N_{\rm pc}}{T_{\rm force}}, \qquad (9)$$

where $N_{\rm [pp|pc]\_tot}$ is the total number of floating point operations of *p-p*/*p-c*; $N_{\rm [pp|pc]}$ is the cumulative number of *p-p*/*p-c*; $\gamma_{\rm [pp|pc]}$ is the number of operation counts for each *p-p*/*p-c* which are constant values of $\gamma_{\rm pp} = 23$ and $\gamma_{\rm pc} = 65$ from the work Bédorf et al. (2014). Here *p-p* and *p-c* refer to particle-particle and particle-cell interactions respectively.

3.2.2 *Opening parameters in tree-code*

Three opening parameters play important roles in running the tree-code and consequently affect the performance with different results.

θ: This is a dimensionless parameter defined in Equation (1) that controls the accuracy. Our test results showed that a smaller θ makes the running time increase sharply, then stop rising in a certain range ($\theta \approx 0.01$ as an experimental value); while a bigger θ ($\theta \geqslant 0.2 \sim 0.35$ influenced by N) causes less accuracy in the simulations.

ϵ: The softening parameter ϵ does not contribute to the running time; on the other hand an optimal ϵ could lead to the best approach to the minimum error. For too small of a softening the estimates of forces will be too noisy, but for too large of a softening the force estimates will be systematically misrepresented; in between there is an optimal softening. The optimal ϵ depends on both

Table 3 Main Components of `Bonsai`

Description	Timing variable	Expected scaling		Fitting value [s]
		N	$N_{\rm p}$	
Sorting and reordering	$T_{\rm sort}$	$\mathcal{O}(N)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(1.5\times10^{-6}\times N+2.45\times10^{-4})\times N_{\rm p}^{-1}$
Tree construction	$T_{\rm build}$	$\mathcal{O}(N)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(2.8\times10^{-7}\times N+2.06\times10^{-2})\times N_{\rm p}^{-1}$
Node properties	$T_{\rm prop}$	$\mathcal{O}(N)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(9.1\times10^{-8}\times N+5.78\times10^{-3})\times N_{\rm p}^{-1}$
Set active groups	$T_{\rm grp}$	$\mathcal{O}(N)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(1.7\times10^{-9}\times N+1.16\times10^{-3})\times N_{\rm p}^{-1}$
Force computation	$T_{\rm force}$	$\mathcal{O}(N{\rm log}N)$	$\mathcal{O}(N_{\rm p}^{-kp_{g1}})$	$(2.5\times10^{-6}\times N{\rm log}N-0.10)\times N_{\rm p}^{-0.88}$
Domain update	$T_{\rm dom}$	$\mathcal{O}(N{\rm log}N)$	$\mathcal{O}(1)$	$5.4\times10^{-10}\times N{\rm log}N+2.96\times10^{-3}$
Exchange	$T_{\rm exch}$	$\mathcal{O}(N{\rm log}N)$	$\mathcal{O}(1)$	$2.1\times10^{-9}\times N{\rm log}N+1.16\times10^{-2}$
Synchronization	$T_{\rm syn}$	$\mathcal{O}(N^{kn_s})$	$\mathcal{O}(kp_{s1}\times N_{\rm p}^{kp_{s2}})$	$(1.4\times10^{-4}\times N^{0.45}+9.3\times10^{-4})(0.5\times N_{\rm p}^{0.49})$
Prediction	$T_{\rm pre}$	$\mathcal{O}(N)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(1.5\times10^{-8}\times N+1.49\times10^{-3})\times N_{\rm p}^{-1}$
Correction	$T_{\rm corr}$	$\mathcal{O}(N)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(3.8\times10^{-8}\times N+7.88\times10^{-4})\times N_{\rm p}^{-1}$
Energy check	$T_{\rm ene}$	$\mathcal{O}(N)$	$\mathcal{O}(N_{\rm p}^{-1})$	$(8.8\times10^{-9}\times N+7.14\times10^{-4})\times N_{\rm p}^{-1}$

Notes: Detailed descriptions of symbols that are used are listed in Table 1.

Table 4 Comparison with `Bonsai` and `NBODY6++`

N	8k	16k	32k	64k	128k	256k	512k	1M
$T_{\rm tot}$ [s]	8.06	22.48	60.92	192.31	629.4	2,071.38	8,010.08	28,737.83
Δt_{bs}	1.30×10^{-3}	1.13×10^{-3}	0.96×10^{-3}	0.78×10^{-3}	0.61×10^{-3}	0.46×10^{-3}	0.29×10^{-3}	0.19×10^{-3}
Δt_{nbi}	6.05×10^{-5}	4.29×10^{-5}	3.25×10^{-5}	2.48×10^{-5}	1.89×10^{-5}	1.49×10^{-5}	1.41×10^{-5}	1.31×10^{-5}
ΔE_{bs}	6.14×10^{-6}	3.45×10^{-6}	1.32×10^{-6}	0.98×10^{-6}	1.27×10^{-6}	1.74×10^{-6}	2.80×10^{-6}	4.36×10^{-6}
ΔE_{nb}	5.71×10^{-7}	5.07×10^{-7}	4.88×10^{-7}	3.44×10^{-7}	4.73×10^{-7}	2.33×10^{-7}	4.92×10^{-7}	4.47×10^{-7}

the number of particles and the size of time steps. From the work Athanassoula et al. (2000) when ϵ has different minimum values, the conditions of simulations are not the same and there is a relationship between N and optimal ϵ. Through a comparison of their conclusions with groups of our test results of `Bonsai` code (using ΔE instead of *MASE*, $\theta = 0.5$; $\Delta t = 0.0625$), we conclude that the value of ϵ leading to a minimum ΔE is consistent with conclusions of the reference.

Δt: The value of Δt affects both running time and energy error. On the running time side, Δt is noticeably linearly dependent with time as: $t \propto 1/\Delta t$; on the relative energy error side, the test results are more complex. Our results show that $\Delta E_{\rm min}$ varies sensitively under the chosen combinations of Δt and ϵ, as well as different N.

3.2.3 Performance results

The measured total wall-clock time of `Bonsai` is shown in Figure 4.

In Figures 4 and 5 we do not use any data for the case of the number of particles being smaller than the optimal capacity of GPU nodes, because the performance goes down and the GPUs are not fully loaded in this regime.

We decomposed $T_{\rm tot}$ into main components as described in Section 3.2.1. For every component we measured the running time separately, and obtained fitting formulae as a function of N and $N_{\rm p}$. The fitting procedures were the same as in the `NBODY6++` part described in Section 3.1.1. The results are shown in Table 3.

Figure 5 shows the experimental speed-up of `Bonsai` defined as Equation (4). Compared with the result of Figure 2, `Bonsai` has a tendency to yield a lower peak but wider scope. Considering the weight factors of the force computational part in Tables 2 and 3, quantitative information about the behavior of the code is revealed, while the different weight of the communication part is the main determinant for descendant lines.

Figure 6 shows the hardware performance of `Bonsai` in a practical environment on a real GPU accelerated cluster. The performance in floating point operations per second is only given for the dominant part of the force computations. The figure indicates that for a large enough N we get nearly half of the peak single precision of our M2050/M2070 GPU card and this increases steadily for large scale $N_{\rm p}$. This proportion is similar to the result of `NBODY6++` code discussed in Section 3.1. The utilization of the GPU is quite good considering the tree-code structure.

4 DISCUSSION

In this paper, we analyze the performance of two very different kinds of *N*-body codes, both of which are pioneers in their fields and both are heavily optimized for GPU acceleration and parallelization - `NBODY6++` and `Bonsai`. There is always a question of what is the break-even point for the codes, or how do they compare with each other. Due to the very different natures of the two codes such a comparison is inevitably unfair - `NBODY6++` has few-body

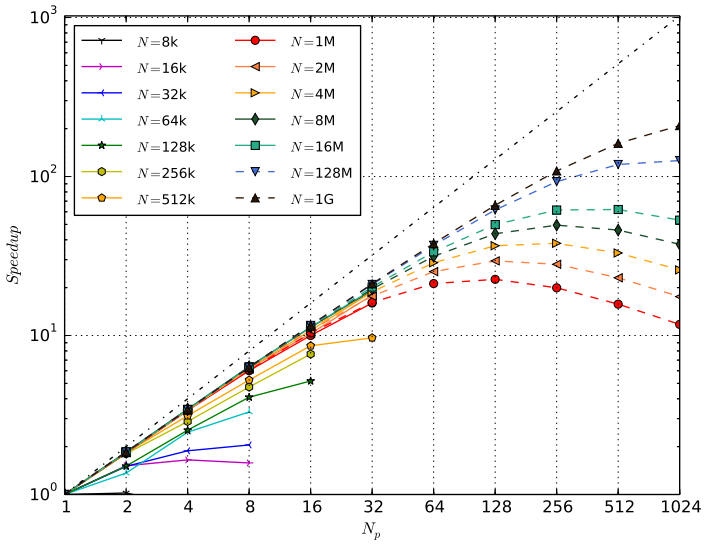

Fig. 5 The speed-up (S) of `Bonsai` as a function of particle number N and $N_{\rm p}$. The legend is the same as in Figure 2. Opening parameters of initial condition are set as $\Delta t = 0.0625$, $\epsilon = 0.01$ and $\theta = 0.5$.

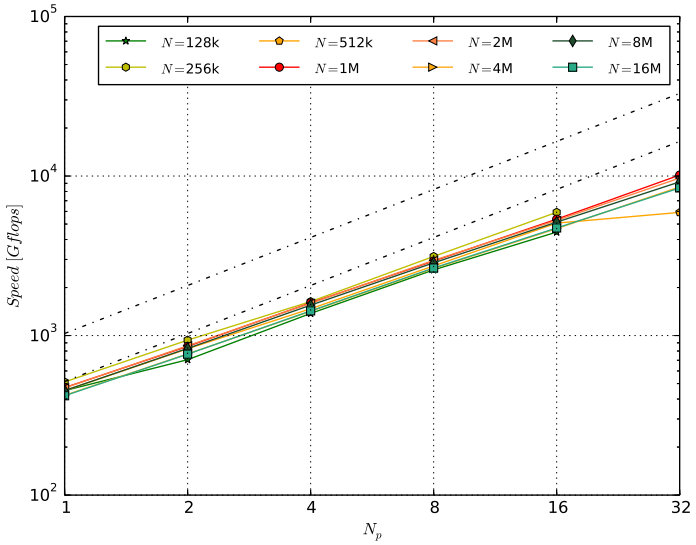

Fig. 6 Hardware performance of force computation speed ($P_{\rm force}$) of `Bonsai` running on the "Milky Way" GPU cluster. Two dashed lines in the figure refer to the peak single and double precision floating point performance.

regularizations and is aimed at high accuracy of both near and more distant gravitational forces; `Bonsai` achieves optimal performance if the opening parameter θ is relatively large, providing rather less accurate gravitational forces. But in certain ranges of parameters, both codes may overlap in terms of performance, accuracy and efficiency. It is the goal of this paper to provide quantitative information about this.

We do this with the help of the four panels in Figure 7 – they show wall-clock time and energy accuracy as a function of the average time step; the main curves are for `Bonsai` as indicated in the caption, for two different opening parameters. However, data for `NBODY6++` are shown for comparison: wall-clock time and accuracy as a function of average time step. In addition, we show that for a fixed particle number the time step of `Bonsai` results in the same wall-clock time as for `NBODY6++`.

The following main conclusions can be drawn: at the same wall-clock time and same particle number (and $\theta = 0.2$) `Bonsai` typically runs with time steps of a factor that are $10 - 50$ larger. In other words `NBODY6++` provides a much smaller time step and a factor of 10 better accuracy (see lower panels of Fig. 7). Here energy error is used as the criterion to compare the accuracy. In our case the time evolution of the energy error contains two main parts. One part comes from the machine accuracy of the potential and force calculations. This is in our cases close to the single precision machine accuracy of order $10^{-7} \sim 10^{-8}$.

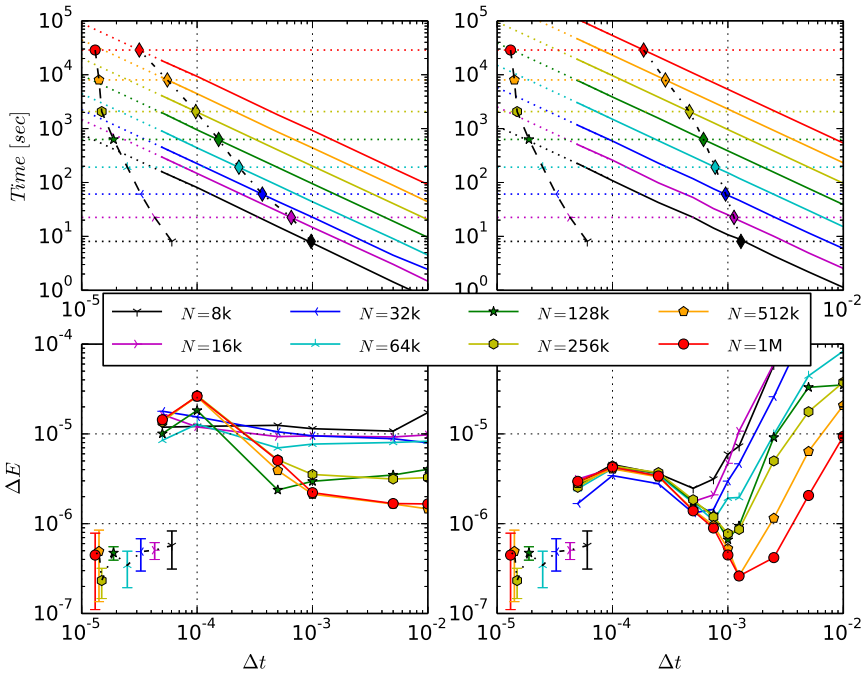

Fig. 7 Comparisons of wall-clock time and relative energy error of NBODY6++ and Bonsai as a function of Δt. Opening parameters of Bonsai are set as $\epsilon = 0.01$ and $\theta = 0.5$ in the left column, and smaller values of $\epsilon = 0.001$ and $\theta = 0.2$ as the control group in the right column. In each panel the left dashed line corresponds to NBODY6++ benchmark data, and solid lines are Bonsai data. The diamond symbols indicate junctions of Bonsai which have the same running time as NBODY6++ in the case of the same N.

The other component of error comes from the numerical integration process itself, which plays the dominant role in total energy error. In NBODY6++ we are using the complex fourth-order Hermite individual block time step integration combined with the Ahmad-Cohen neighbor scheme. We have chosen the time step parameter η of the Aarseth time step criteria (for regular and irregular time steps, the values of which are set as 0.02 in the initial input files) such that the energy error stays at the level of $10^{-6} \sim 10^{-7}$. How the global energy error of our integrator in NBODY6++ behaves can be found in a comprehensive study by Makino (1991). In the case of the Bonsai, the code that uses the simple leap-frog integration scheme, which (for a reasonable computational speed to reach the 1 N-body Time Unit) has an average energy error at a level of $10^{-5} \sim 10^{-6}$. So far we have not discovered anything unexpected; however Bonsai can reach surprisingly good accuracy in total energy (like 5×10^{-6}) at wall-clock times comparable to NBODY6++. With a larger opening angle ($\theta = 0.5$) the time step and wall-clock parameters approach each other more (by a factor of two to three, for one million bodies). In such a case, there is a quite considerable accuracy in energy of order 10^{-5}. It seems that levels of energy accuracy arguably may be sufficient even for collisional gravothermal systems.

However, total energy conservation is not the only criterion to judge the use of a code and its accuracy. In NBODY6++ close encounters and interactions of compact or hierarchical multiple systems are treated with regularization methods and zero softening, while Bonsai uses an artificial softening of the gravitational potential at small distances. Reasonable energy conservation refers to artificial gravitational potential including softening, which is conservative as well, but not to the true few-body potential. So, the additional numerical efforts necessary for NBODY6++ go on one hand into the exact resolution of all kinds of close interactions below the softening length used in Bonsai. But also on the other hand, for long-range interactions, Bonsai uses the standard tree-code procedure of approximating forces from groups of particles by forces from their centers of mass and multipoles. This feature needs to be tested by simulation of core collapsing star clusters, where long range gravitational interactions determine the global evolution, which is beyond the scope of this paper.

Acknowledgements We want to thank NBODY series developer Sverre Aarseth for providing the NBODY6 code and lectures about how to use it. We thank Long Wang for continuously developing the newest version of NBODY6++ code. We also want to thank Bonsai developers Jeroen Bédorf, Evghenii Gaburov and Simon Portegies Zwart for providing their N-body code. We acknowledge support by Chinese Academy of Sciences through the Silk Road Project at NAOC, through the Chinese Academy of Sciences Visiting Professorship for Senior International Scientists, Grant Number 2009S1-5 (RS), and through the "Qianren" special foreign experts program of China.

The special GPU accelerated supercomputer “Laohu” at the Center of Information and Computing at National Astronomical Observatories, Chinese Academy of Sciences, funded by the Ministry of Finance of the People’s Republic of China under the grant ZDY Z2008-2, has been used for the simulations, as well as the supercomputer “The Milky Way System” at Jülich Supercomputing Centre in Germany, built for SFB881 at the University of Heidelberg, Germany.

PB acknowledges the special support by the NAS Ukraine under the Main Astronomical Observatory GPU/GRID computing cluster project.

References

Aarseth, S. J. 1999, PASP, 111, 1333

Ahmad, A., & Cohen, L. 1973, Journal of Computational Physics, 12, 389

Amdahl, G. M. 1967, in Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS’67 (Spring) (New York: ACM), 483

Athanassoula, E., Fady, E., Lambert, J. C., & Bosma, A. 2000, MNRAS, 314, 475

Barnes, J., & Hut, P. 1986, Nature, 324, 446

Bédorf, J., Gaburov, E., Fujii, M. S., et al. 2014, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC’14 (Piscataway: IEEE Press), 54

Bédorf, J., Gaburov, E., & Portegies Zwart, S. 2012a, Journal of Computational Physics, 231, 2825

Bédorf, J., Gaburov, E., & Portegies Zwart, S. 2012b, in Astronomical Society of the Pacific Conference Series, 453, Advances in Computational Astrophysics: Methods, Tools, and Outcome, ed. R. Capuzzo-Dolcetta, M. Limongi, & A. Tornambè, 325

Berczik, P., Spurzem, R., & Wang, L. 2013, in Third International Conference ”High Performance Computing”, HPC-UA 2013, 52

Berczik, P., Nitadori, K., Zhong, S., et al. 2011, in International Conference on High Performance Computing, Kyiv, Ukraine, October 8-10, 8

Dorband, E. N., Hemsendorf, M., & Merritt, D. 2003, Journal of Computational Physics, 185, 484

Genel, S., Vogelsberger, M., Springel, V., et al. 2014, MNRAS, 445, 175

Gualandris, A., Portegies Zwart, S., & Tirado-Ramos, A. 2007, Parallel Computing, 33, 159

Hamada, T., Narumi, T., Yokota, R., et al. 2009, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC’09 (New York: ACM), 62:1

Hamada, T., & Nitadori, K. 2010, in 2010 International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 1

Harfst, S., Gualandris, A., Merritt, D., et al. 2007, New Astron., 12, 357

Hut, P. 2003, in IAU Symposium, 208, Astrophysical Supercomputing using Particle Simulations, eds. J. Makino, & P. Hut, 331

Kawai, A., Fukushige, T., & Makino, J. 1999, in Proceedings of the 1999 ACM/IEEE Conference on Supercomputing, SC’99 (New York: ACM)

Khalisi, E., Omarov, C., Spurzem, R., Giersz, M., & Lin, D. 2003, in High Performance Computing in Science and Engineering’03, ed. E. Krause, W. Jáger, & M. Resch (Springer Berlin Heidelberg), 71

Kustaanheimo, P., & Stiefel, E. 1965, J. Reine Angew Math. 218, 204

Makino, J. 1991, ApJ, 369, 200

Makino, J. 2002, New Astron., 7, 373

Makino, J., Fukushige, T., Koga, M., & Namura, K. 2003, PASJ, 55, 1163

Nitadori, K., & Makino, J. 2008, New Astron., 13, 498

Shin, J., Kim, J., Kim, S. S., & Park, C. 2014, Journal of Korean Astronomical Society, 47, 87

Springel, V. 2005, MNRAS, 364, 1105

Spurzem, R. 1999, Journal of Computational and Applied Mathematics, 109, 407

Spurzem, R., Berczik, P., Zhong, S., et al. 2012, in Astronomical Society of the Pacific Conference Series, 453, Advances in Computational Astrophysics: Methods, Tools, and Outcome, eds. R. Capuzzo-Dolcetta, M. Limongi, & A. Tornambè, 223

Spurzem, R., Berentzen, I., Berczik, P., et al. 2008, in Lecture Notes in Physics, Berlin Springer Verlag, 760, The Cambridge N-Body Lectures, eds. S. J. Aarseth, C. A. Tout, & R. A. Mardling, 377

Vogelsberger, M., Genel, S., Springel, V., et al. 2014, MNRAS, 444, 1518

Wang, L., Spurzem, R., Aarseth, S., et al. 2015, MNRAS, 450, 4070

Warren, M. S., & Salmon, J. K. 1992, in Proceedings of the 1992 ACM/IEEE Conference on Supercomputing, Supercomputing’92 (Los Alamitos: IEEE Computer Society Press), 570

Winkel, M., Speck, R., Hübner, H., et al. 2012, Computer Physics Communications, 183, 880