NBODY6++GPU: ready for the gravitational million-body problem

Long Wang,1,2⋆ Rainer Spurzem,3,4,5,1 Sverre Aarseth,6 Keigo Nitadori,7 Peter Berczik,3,4,5,8 M. B. N. Kouwenhoven1,2 and Thorsten Naab9

1 Kavli Institute for Astronomy and Astrophysics, Peking University, Yiheyuan Lu 5, Haidian Qu, 100871, Beijing, China
2 Department of Astronomy, School of Physics, Peking University, Yiheyuan Lu 5, Haidian Qu, 100871, Beijing, China
3 National Astronomical Observatories and Key Laboratory of Computational Astrophysics, Chinese Academy of Sciences, 20A Datun Rd., Chaoyang District, 100012, Beijing, China
4 National Astronomical Observatories and Key Laboratory of Computational Astrophysics, Chinese Academy of Sciences, 20A Datun Rd., Chaoyang District, 100012, Beijing, China
5 National Astronomical Observatories and Key Laboratory of Computational Astrophysics, Chinese Academy of Sciences, 20A Datun Rd., Chaoyang District, 100012, Beijing, China
6 RIKEN Advanced Institute for Computational Science, Kobe, Japan
7 Max-Planck Institut für Astrophysik, Karl-Schwarzschild-Str. 1, D-85741 Garching, Germany

ABSTRACT
Accurate direct N-body simulations help to obtain detailed information about the dynamical evolution of star clusters. They also enable comparisons with analytical models and Fokker-Planck or Monte Carlo methods. NBODY6 is a well-known direct N-body code for star clusters, and NBODY6++ is the extended version designed for large particle number simulations by supercomputers. We present NBODY6++GPU, an optimized version of NBODY6++ with hybrid parallelization methods (MPI, GPU, OpenMP, and AVX/SSE) to accelerate large direct N-body simulations, and in particular to solve the million-body problem. We discuss the new features of the NBODY6++GPU code, benchmarks, as well as the first results from a simulation of a realistic globular cluster initially containing a million particles. For million-body simulations, NBODY6++GPU is 400–2000 times faster than NBODY6 with 320 CPU cores and 32 NVIDIA K20X GPUs. With this computing cluster specification, the simulations of million-body globular clusters including 5 per cent primordial binaries require about an hour per half-mass crossing time.

Key words: methods: numerical – globular clusters: general.

1 INTRODUCTION
Direct simulations of star clusters have a long history. As algorithms and hardware have improved, larger numbers of stars could be simulated, allowing a more realistic representation of the dynamical evolution of globular star clusters. NBODY6 (Aarseth 2003) is a state-of-the-art direct N-body simulation code specifically designed for star clusters. It uses several algorithms to enhance the computing speed and accuracy, especially for strong interactions that arise from a large fraction of binaries and relatively short relaxation time-scales (≤100 Myr for typical open clusters and ≤1 Gyr for typical globular clusters) in collisional and dense stellar systems. Here, the terms ‘collisional’ and ‘dense’ are not well defined in the literature. The classical two-body relaxation time, as defined e.g. by Chandrasekhar (1942), Spitzer (1987), describes how important distant gravitational two-body encounters are for the orbital motion of stars. If the relaxation time is very long, a system is denoted as ‘collisionless’ (e.g. galactic discs or bulges); the motion of stars is entirely determined by the smooth mean gravitational field of the system. If the relaxation time is short (e.g. shorter than the lifetime of the system) we denote the cluster as ‘collisional’ (e.g. globular and open star clusters, nuclear star clusters). If the stellar density is high enough, close two-body gravitational encounters and stellar collisions may occur. This aspect is crucial when studying ‘dense’ star clusters. In dense and collisional star clusters a correct integration of stellar motions requires pairwise gravitational interactions to be included between many if not all stars in the cluster. This is the situation for which codes such as NBODY6 are designed.

Direct N-body simulation of star clusters can be very time consuming. In a system with N particles, the full force calculation cost of one particle scales with O(N). With individual time-steps for each particle, the cost per crossing time (tcr) depends on the number of steps per particle (Ncr) which varies with different time-step criteria, integration methods and star cluster properties. Makino & Hut (1988) and Makino & Aarseth (1992) found that for the Hermite

⋯
scheme with a time-step criterion based on relative force change (Aarseth 1985), \( N \) is roughly proportional to \( N^{1/3} \) for systems with homogeneous density. Thus, when using individual time-steps the total computational cost per crossing time of \( N \) particles scales with \( O(N^{2/3}) \). For systems with a power-law density distribution \( \rho \propto r^{-\alpha} \), \( N \) depends on the power index \( \alpha \). Then the cost per crossing time-scales with \( O(N^{2/3}) \) for \( \alpha < 24/11 \) and \( O(N^{6/(2\alpha-1)/3}) \) for \( \alpha \geq 24/11 \) (Makino & Hut 1988). Considering the half-mass relaxation time-scale, \( t_{\text{rh}} \) is proportional to \( t_{\text{rh}}/\ln N \) (Spitzer 1987; Sugimoto et al. 1990), the computational cost per \( t_{\text{rh}} \) is \( O(N^{2/3}/\ln N) \) for homogeneous systems and for power-law systems with \( \alpha < 24/11 \) and \( O(N^{12-8\alpha}/(6\alpha-2\alpha)/\ln N) \) for \( \alpha \geq 24/11 \). Thus, an efficient parallelization of a direct \( N \)-body code is necessary for large particle numbers.

Sugimoto et al. (1990) discussed the fundamental problem that direct numerical simulations of globular star clusters could not be completed for decades if extrapolating the standard evolution of computational hardware (Moore’s law). They called for the construction of a special-purpose computer GRAPE, which finally was successfully initiated and completed by their team (Makino, Kokubo & Taiji 1993; Makino & Taiji 1998; Makino et al. 2003). In the following years, graphical processing units (GPU) widely replaced GRAPE (e.g. Harfst et al. 2007; Hennebelle, Audit & Miville-Deschênes 2007; Portegies Zwart, Bellemare & Geldof 2007; Bellemain, Bédor & Portegies Zwart 2008; Schive et al. 2008) and much of the GRAPE software could be ported to GPU (Gaburov, Harfst & Portegies Zwart 2009).

Spurzem (1999), Spurzem et al. (2008) and Hensensdorff et al. (2003) discussed several different types of hardware for parallelization and extended \textsc{nboby6} to \textsc{nboby6++} for general parallel supercomputers. Later, Nitadori & Aarseth (2012) developed a GPU-based parallel force calculation for \textsc{nboby6}. As a result, large \( N \)-body simulations (\( N \sim 10^5 \)) became possible on a single desktop computer or workstation with GPU hardware. They also implemented the parallel force calculation based on Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX) for recent CPU architectures. Spurzem et al. (2011), Berczik, Spurzem & Wang (2013b) and Berczik et al. (2013a) discussed the performance of large \( N \)-body simulations with the GPU-accelerated codes \textsc{ph6} and a provisional version of \textsc{nboby6++}. The basic idea is to employ a neighbour list for each particle. The integration is separated into two parts: regular force integration for large time-steps (regular steps) and irregular force integration for small time-steps (irregular steps). The regular force is the summation of the forces from particles outside the neighbour radius and the irregular force accumulates only the neighbour forces. During the irregular step, the regular force and its first order derivative calculated at the last regular step are used for position and velocity prediction. The AC scheme gains efficiency with sequential computing (without parallelization). The speed gained by the AC scheme is roughly proportional to \( N^{1/4} \) (Makino & Hut 1988; Makino & Aarseth 1992). However, in parallel computing, this gain is limited by the complexity of the implementation of this algorithm (see Section 4). Also, the benefit of reducing overheads of particle integration in the block time-step method is strongly limited in the neighbour scheme.

Portegies Zwart & Boekholt (2014) discussed the integration accuracy requirement for self-gravitating systems simulated with direct \( N \)-body codes. They found that for three-body systems the integration should have total energy conserved better than 1/10th. Although this accuracy requirement is uncertain when the simulation is extended to large particle number systems, this work indicates the importance of careful integration treatment for direct \( N \)-body systems. One important feature of \textsc{nboby6} is that it uses the algorithms of Kustaanheimo & Stiefel (1965) (hereafter KS) and chain regularization (Mikkola & Aarseth 1993) to deal with an accurate solution of close encounters, binaries and multiple systems, which play a significant role in star cluster dynamical evolution. These strong interactions require very small time-steps during integration and may produce large errors with standard integrators such as the Hermite scheme. Using KS and chain regularization is also the most important feature of \textsc{nboby6} for star cluster simulations.

Here, we have briefly introduced the main algorithm used in \textsc{nboby6++}. In the next section we will focus on the parallelization of the codes.

3 Parallelization of \textsc{nboby6++}

3.1 MPI parallelization of \textsc{nboby6++}

Spurzem (1999) and Hensensdorff et al. (2003) developed \textsc{nboby6++} based on \textsc{nboby6} using MPI parallelization with the
copy algorithm. Both regular and irregular forces were parallelized. Here different MPI processors calculate different subsets of the active particles. Each MPI processor has the complete particle data set. Another available parallel algorithm is the ring algorithm which splits the full particle data set for different MPI processors. It reduces the memory cost in each MPI process. The benefit of the copy algorithm compared to the ring algorithm is that there is no requirement for extra communication of the neighbour particle data which is not in the same MPI process during the irregular force calculation. The disadvantage is the particle number limit due to memory size on the computing node. The MPI communication with the copy algorithm has constant time cost (independent of MPI processor number except for latency). The scaling of the regular force with different MPI processors is very good. Since the regular force dominates the calculation, this results in a good scaling of the total computing time. Dorband, Hensendorf & Merritt (2003) provided a detailed discussion of these communication algorithms. Lippert et al. (1998) and Makino (2002) suggested an efficient communication algorithm (hypersystolic) for extremely large processor numbers.

3.2 Basic nbody6-GPU implementation

After the GPU computing (CUDA) became popular, the shared memory parallel nbody6-GPU code was developed for workstation and desktop computers (Nitadori & Aarseth 2012). The OpenMP, GPU (CUDA) and AVX/SSE parallel methods are used to make the code as fast as possible. However, nbody6-GPU can only be used in a single node (no massively parallel MPI implementation) so the number of particles is limited for a reasonable simulation time.

3.2.1 Regular force and potential (GPU)

The GPU library of Nitadori & Aarseth (2012) is used for calculating the regular force, which dominates the direct integration, and potential energy calculation. The cost for regular force calculation per particle scales with $O(N)$ and for potential energy calculation scales with $O(N^2)$. The performance of GPU force calculation is very good since the pure force calculation is easy to parallelize. GPUs also help to accumulate the neighbour list very efficiently during the regular force calculation.

3.2.2 Prediction and irregular force (AVX/SSE)

When GPU accelerates the regular force very efficiently, the irregular force becomes expensive. However, this part is hard to parallelize on GPUs due to the complexity of the AC neighbour scheme. Thus, Nitadori & Aarseth (2012) developed the AVX/SSE and OpenMP parallel library for neighbour particle prediction and irregular force calculation. AVX/SSE is an instruction set for CPUs developed in recent years, which supports vector calculation in the specific cache. The advantage of AVX/SSE with OpenMP is that there is no extra memory copy compared to GPU. For both AVX/SSE and GPU libraries, the data need to be copied once for changing data structure to obtain computing efficiency. This is because nbody6 has a very long development history, thus to completely change the data structure to be consistent with AVX/SSE and GPU libraries is very time consuming. But for GPU, there is extra data copy from the host memory on the mother board to the device memory on GPU. Besides, since the neighbour force calculation is not efficient for the distributed memory parallel method (with MPI parallelization; see discussion below), this kind of shared memory parallel method is more efficient.

3.3 Code improvements in nbody6+GPU

In this subsection we describe our new implementations in nbody6+GPU.

The GPU acceleration, especially of the long-range (regular) gravitational forces, is very efficient so this part does not dominate the computational time any more, as we show below. Secondly, the AVX/SSE implementation accelerates prediction and neighbour (irregular) forces, which is the next most time consuming part of the code.

We have combined the GPU and AVX/SSE acceleration, which was done for a single node in nbody6-GPU, with the MPI parallelized nbody6++ designed for multi-node computing clusters for the new version nbody6+GPU. This work requires additional efforts to keep the code consistent (see below). In addition, we have worked on remaining bottlenecks, such as time-step scheduling and stellar evolution, which become important for million bodies because the usual computationally intensive tasks have been accelerated very effectively by GPU and AVX/SSE.

3.3.1 New algorithm of selecting active particles for block time-steps

For the block time-step method, active particles should be selected at every time-step. It is very expensive to search all particles for the active ones, especially for the irregular force calculation. In this case, for one block time-step the cost of selecting active particles scales with $O(N)$ while the irregular force calculation cost scales with $O(N_i(N_b))$. If $N \gg N_i(N_b)$, the former can be more expensive. When the simulation reaches millions of particles, the block time-step levels can be quite deep (the smallest time-step can reach $0.5^{20}$–$0.5^{22}$) and the deep blocks with few particles and small time-steps can easily satisfy this condition. One may consider to use a temporary list to save particles with small time-steps and only search all particles at some selected time interval from the temporary list each time-step. However, this method is still expensive where there are many particles with small time-steps (such as the wide binaries that are not KS regularized). Indeed, we find that the time of selecting active particles can be much larger than the irregular integration time, even with this temporary list algorithm for one million particles including 5 per cent primordial binaries. Another reason that forces us to deal with this issue is that the active particles selection is very difficult to parallelize efficiently (the cost is almost independent of processor numbers) and would be prohibitive for a million-body simulation. Thus, we propose a better algorithm that uses a time-step sorting list (hereafter sorting list algorithm; see Figs 1 and 2). Zhong (2014) implemented a similar algorithm for φ-grape+GPU and evaluated its performance. The basic idea is that when we have the index list sorted by particle time-step from smallest to largest, and the indicators of each boundary offset $L_0(i)$ between the block of the same step particles (the largest particle index with step $0.5^k$), we only need to find the correct offset at each block time-step by using the algorithm shown in Fig. 1 to select active particles (shown as black squares in Fig. 2). After integration, we adjust the sorted list by sorting the active particles’ new time-steps. The specific sorting method for this adjustment can be optimized to $O(N_i)$ if we ignore the stability of sorting (stability means no exchange of the order for the particles with same steps).
The flow chart for obtaining the correct offset in the sorting list algorithm. \( T_{\text{new}} \) is the time after next integration. \( T_i \) is current time. \( d T_{\text{min}} \) is the current smallest time-step (the time-step of the first particle in sorting list). \( i_{\text{prev}} \) is the previous offset.

Figure 1. Diagrammatic sketch of sorting list algorithm for selecting active particles. The time-step of each particle block is 0.5\( ^\circ \) separated by boundary indicators \( I_{\text{off}}(i) \) (vertical lines). The integration advances vertically in the chart. Active particles are shown as black squares.

Figure 2. and assume that many active particles keep the same step as before or have small time-step changes.

3.3.2 The initialization

The initialization of a simulation in \textsc{nbody6} is relatively expensive. We improve it with MPI, GPU and OpenMP parallelization and a better algorithm. The initial model for million-body simulations is very important and needs to be carefully tested. This improvement is very useful for fast testing of the initial models with large particle numbers, especially for a large number of primordial binaries.

The initialization of \textsc{nbody6} can be divided into four parts:

(i) reading or generating masses, positions, velocities and stellar evolution parameters of all stars;
(ii) scaling all parameters into \( N \)-body units (the \( N \)-body units\(^1\) are defined in Heggie & Mathieu 1986);
(iii) initialization of forces, neighbour lists and time-steps of all stars;
(iv) initialization of primordial KS binaries.

In the second part of the initialization, the total potential energy of the system is needed and costs \( O(N^2) \). Actually, \textsc{nbody6++} does this calculation twice for scaling purpose in the case of an external tidal field. The GPU is used in \textsc{nbody6++} to speed up this part and it is very efficient. Our new improvements are for the third and fourth parts. In the traditional \textsc{nbody6++} version forces and neighbour lists are initialized separately without parallelization.

\textsc{nbody6++} parallelizes the scaling and initialization of the force parts, but only through MPI. For million-body simulations this is very slow and requires hours to be finished. We improved it by using GPU-based force and neighbour list calculations (the same as for the regular force calculation). The fourth part is very costly with more than 5 per cent primordial KS binaries in the traditional \textsc{nbody6++} (several hours). During initialization of KS binaries, the force and its three derivatives (Hermite scheme) need to be renewed for centre-of-mass particles. All neighbour lists that contain KS binary component indices also need to be replaced by the centre-of-mass particle indices. The cost is approximately \( O(N(N_{\text{KS}})N_{\text{KS}}) \) where \( N_{\text{KS}} \) is the number of primordial KS binaries. We find a much simpler way to initialize KS binaries [cost scales with \( O(N_{\text{KS}}) \)] by just switching the order of the third and fourth parts: initialize KS binaries first without recalculating forces, their derivatives and neighbour lists (only the KS transformation is needed) and then do the third process with the new centre-of-mass particles data generated by former process instead of each binary component in the old way. In this case there is no need to update the forces and neighbour lists.

3.3.3 Position and velocity prediction

During the force calculation, the predicted positions and velocities are used to calculate the force and its first derivative for the Hermite integrator. In principle, we can avoid prediction of the same particles with the AC neighbour scheme and block time-steps. However, in practice we need to search all neighbours of each active particle and the search itself is computationally expensive. Thus, it does not save much time to avoid neighbour prediction overlap and it is much simpler to predict all neighbours and do the force calculation within one loop. The disadvantage of this method is that it costs more when the average neighbour number \( \langle N_{\text{b}} \rangle \) multiplied by the active particle number \( N_i \) is larger than the total particle number \( N \), compared to all the particle predictions with a non-AC scheme block time-step. One solution is to try predicting all particles once instead of predicting each neighbour when \( \langle N_{\text{b}} \rangle N_i / N > N \). But the mixture of predicting only neighbours and predicting all particles increases the complexity of code. We therefore use only neighbour prediction in the code.

However, there is a major complication for the parallel neighbour prediction in \textsc{nbody6++}, which does not exist in \textsc{nbody6++}. Since we use AVX/SSE and GPU and the code is mixed with \textsc{cuda}, \textsc{c++} and \textsc{fortran} 77 programming language, the AVX/SSE and GPU libraries keep the individual copies of particle data sets. Thus, the predictions of particles have overlaps and are usually inconsistent for different copies distributed on MPI processors. Due to the complexity of \textsc{nbody6++} (e.g. using predicted positions for regularization) this leads to problems of synchronization later on, such as differences of time-steps for the same particle on different processors. The safest but very costly way is to always predict all

\(^1\) It has been suggested to name the \( N \)-body time unit to honour M. Hénon as Hénon time unit (D.C. Heggie, private communication).
Figure 3. NBODY6++GPU code structure. It shows one cycle of simulation. Based on the time-steps, the integration can be divided into three hierarchical parts (see Table 1): KS calculation (KS), irregular integration (Irr.) and regular integration (Reg.). The KS has smallest time-step distribution. Thus, between two nearest Irr. block time-steps there are several KS steps. Similarly, between two Reg. block time-steps there are several Irr. time-steps. After several Reg. time-steps there is one ‘Adjust’ (see Table 1). Inside one node, Reg. and Adjust are parallelized by multiple GPUs and Irr. is parallelized by AVX/SSE with OpenMP. MPI parallelization are done for all four parts between different nodes.

4.1 Pure MPI and hybrid MPI

Fig. 4 shows significant improvement of NBODY6++GPU by using hybrid MPI including GPU, as compared to the pure MPI case. We see that the GPU gives about 33 times faster regular force integration. This is to be expected since GPU is designed for large parallelization by using many computing cores and large memory bandwidth within one card. Using AVX/SSE with OpenMP gives about three times faster irregular integration including predictions. OpenMP reduces the MPI communication cost by a factor of 5–10. The individual MPI communication process is not directly sped up by OpenMP. When we use MPI parallelization together with OpenMP method, inside one node the irregular and regular force calculations are done by multiple threads with OpenMP instead of MPI parallelization, thus we can set a larger block particle number threshold for MPI parallelization by a factor of the OpenMP thread number and reduce the total MPI communication frequency. This then results in shorter total MPI communication time.
4.2 Scaling with different particle numbers and processors

The scaling with different particle numbers and processors demonstrates the possibility of using large computing resources for simulations. We test hybrid parallel NBODY6++GPU scaling with different node numbers $N_{\text{node}}$ (1, 2, 4, 8 and 16; up to 320 CPU cores and about 80k GPU cores) and different particle numbers (16k, 32k, 64k, 128k, 256k and 1024k) on the ‘Hydra’ cluster of the Max-Planck Supercomputing Centre (RZG) Germany. Each node is completely controlled (no other tasks on the node) and has two NVIDIA K20X with 2688 cores each (732 MHz per core) and 20 Intel Ivy Bridge cores (2.8 GHz per core). The total computing time for one $N$-body time unit $T_{\text{tot}}/T_{\text{NB}}$ is shown in Fig. 5. The irregular and regular force integration computing time ($T_{\text{irr}}$ and $T_{\text{reg}}$) is shown in Fig. 6. All these three times are the averaged computing times of the first two $N$-body time units of each simulation.

We test two basic initial models. One has no primordial binaries and another has 5 per cent binaries. Both use a Plummer sphere (Plummer 1911) and initial mass function (IMF) from Kroupa, Tout & Gilmore (1993) with mass range 0.08–20 $M_\odot$ and no stellar evolution.

In the non-binary case, the scaling with different $N_{\text{node}}$ for the total time is not ideal because of the communication cost. Here, the speed-up saturates at about 8–16 nodes depending on the particle number. But if we consider the number of cores per node (20 CPU cores and 5376 GPU cores), the scaling with cores is excellent, since with 16 nodes 320 CPU cores and 86016 GPU cores are used. With one node, the performance of NBODY6++GPU is similar to that of NBODY6-GPU. Nitadori & Aarseth (2012) showed that NBODY6-GPU gives about 100 times speed-up compared to the sequential NBODY6 with two NVIDIA GeForce GTX 560 Ti with 384 cores each (822 MHz per core) and 4 Intel i7-2600 K cores (3.40 GHz per core). On the ‘Hydra’ cluster node, the speed-up can reach 100–500 depending on the particle number. Thus, with 16 nodes for one million particles, we can reach a factor of 400–2000 speed-up compared to the sequential NBODY6. Besides, the absolute time cost is very good, especially for the million-body case, the total time is about 800 s for $N_{\text{node}} = 16$. For the $\phi$GPU code tested in the ‘Laohu’ cluster with 32 NVIDIA K20 GPU, one million particles take about 1500 s. Although we cannot compare the two codes directly with different computing cluster specifications, with the similar GPU type and number NBODY6++GPU can reach better performance. CPU cores are ignored here because the time fractions on the CPU for these two codes are very different: $\phi$GPU spends about 90 per cent computing time on the GPU while NBODY6++GPU has much less time fraction (Section 4.3). In the case with 5 per cent primordial binaries, the scaling is not as good as for the case with no binaries due to the KS calculation. We discuss this issue in more detail in Section 6.

The regular and irregular integration times in Fig. 6 are close to ideal for million-body simulations. It means that when ignoring MPI communications, the MPI parallelization speeds up regular and irregular calculation excellently for large number of particles. For small particle numbers (< 10⁵) the scalings of both regular and irregular integrations depart from the ideal parallel limit. The reason is that the number of operations on each node (for regular integration on GPU) is small. Thus, the cost of internal memory accessing and modification during integration, which cannot be scaled with computing cores or nodes, dominates the time.

4.3 Time fraction for different parts

We show the fraction of time spent on different parts of NBODY6++GPU in Fig. 7. In the model without binaries, MPI communication and data moving consume about half of the total time in the case of 1024k particles with $N_{\text{node}} = 16$ and 128k particles with $N_{\text{node}} = 8$, which means the scaling reaches the MPI parallelization speed-up break-even point. For the 1024k particles with 5 per cent binaries, the KS takes about half of the calculation.
time when $N_{\text{node}} \geq 8$. Thus, the KS procedures become the performance bottleneck.

### 4.4 Sorting list algorithm for selecting active particles

In Fig. 8, we compare the performance of the sorting list algorithm and temporary list algorithm described in Section 3.3.1. The star cluster in our test simulation is modelled as a King sphere (King 1966) with $W_0 = 6$ using 1024k stars, 5 per cent of primordial binaries and eight nodes with the same node configuration as in Fig. 5. To indicate that the sorting is very fast, the time of pure sorting part in this algorithm is also shown. We can see the sorting list algorithm is about five times faster than temporary list algorithm.

The time fraction of active particle selection with the new algorithm is shown as yellow part (Init.B.) in Fig. 7. The definitions of the legends are shown in Table 1. We can see Init.B. costs more for simulations with a larger number of particles. Even with this new method, it is close to irregular integration cost for the one million particles case ($\sim 7$ per cent).

### 5 Application

The main task for Nbody6++GPU is to simulate large star clusters. For a typical globular cluster, the total mass is $10^5 - 10^6 \, M_\odot$, thus the total number of particles is of the order of $10^6$. The typical age is about 12 Gyr. In our 1M stars with 5 per cent primordial binary globular cluster model (the same as shown in Fig. 8), we choose the parameters similar to NGC 4372 (Harris 1996). The initial half-mass radius is 7.5 pc and the tidal radius is 89.2 pc with a circular orbit around a point-mass galactic potential. One N-body time unit corresponds to 0.622 Myr. The computing time and number of particles are shown in Fig. 9.

Initially, the computing time per N-body time unit was about 3000 s and this increased when several small time-step particles formed. Later, we carried out several adjustments, then the simulation sped up and became about 1500 s. The number of particles only decreased slightly during 4500 N-body time units, but the computing speed actually increased at a later stage. The reason for the early slow speed was the two unsuitable criteria for triggering or terminating the two-body KS regularization. The first is the separation criterion $R_{\text{cl}}$ and the second is the time-step criterion $\Delta t_{\text{cl}}$. If the auto-adjustments of $R_{\text{cl}}$ and $\Delta t_{\text{cl}}$ are used, they are determined
Figure 7. Pie charts showing the same test as Fig. 5 but time fraction of different components in Hybrid MPI parallel nbody6++. In each chart different rings show different $N_{\text{node}}$. From inside to outside rings, $N_{\text{node}}$ are 1, 2, 4, 8 and 16. The two pie charts on the left show the model without primordial binaries and the pie chart on the right shows the model with 5 per cent binaries. The models in top two pie charts include 1024 particles (singles + binaries) and the model in the pie chart at the bottom includes 128k single particles. An explanation of the legends is provided in Table 1.

Figure 8. Comparison of performance between the sorting list algorithm and the temporary list algorithm. The ‘Pure Sorting’ means the time cost of sorting part in sorting list algorithm.

following Aarseth (2003)

$$R_{\text{cl}} = \frac{4R_0}{N(\rho_3/\rho_b)^{1/7}},$$

$$\Delta t_{\text{cl}} \simeq 0.04 \left( \frac{\eta}{0.02} \right)^{1/2} \left( \frac{R_0}{m} \right)^{1/2}$$

where $\rho_3/\rho_b$ is the central density contrast, $\eta$ is the standard irregular time-step coefficient and $(m)$ is average mass. The factor $4R_0/N$ is the impact parameter for a 90° deflection in a two-body encounter. The auto-adjustment results in $R_{\text{cl}} = 1.4 \times 10^{-6}$ and $\Delta t_{\text{cl}} = 6.8 \times 10^{-8} N$-body units at the beginning of this simulation. But these values are too small and many wide binaries including some unperturbed binaries are not regularized. Thus we switched off auto-adjustment and used $R_{\text{cl}} = 5.0 \times 10^{-6}$ and $\Delta t_{\text{cl}} < 2.0 \times 10^{-7}$ before about 2800 time units. We found that the $R_{\text{cl}}$ and $\Delta t_{\text{cl}}$ parameters were still too small, thus we enlarged $R_{\text{cl}}$ to $1.0 \times 10^{-5}$ and $\Delta t_{\text{cl}}$ to $5.0 \times 10^{-7}$. Then the computing sped up after 2800 time units. The small parameters from auto-adjustment is because equation (1) is originally designed for small number of particles ($N = 10^{2−3}$). For the million-body simulation, the central density is usually high and $\rho_3/\rho_b$ is large. The criterion from equation (1) is only suitable for the central region of the cluster but too small for the outer region. There were several jumps in the computing time after the auto-restart with reduced time-steps. These happened when a large energy error appeared due to specific events, such as difficult triple systems or the sudden change of force caused by large mass loss or premature perturbation of the neighbour sphere (such as neutron stars with high kick velocities). After we restore the normal time-step parameters the computing time was again reduced.

There are also a few more models currently in progress and we will report in detail about the results from these simulations in a future publication.

6 DISCUSSION

While standard Hermite codes report a high efficiency using up to 700 000 cores on hundreds if not thousands of GPUs (Berczik et al. 2013a,b), we find that our performance saturates at about 86 000 GPU cores and 320 CPU cores. This is not surprising, because nbody6 and nbody6++ are inherently more efficient than standard Hermite codes (less operations for the same physical result). A more detailed scaling analysis of nbody6++ will be published separately (Huang et al., private communication).

As discussed in Sections 4.2 and 4.3, the data movement and MPI communications become the bottleneck when the node number $N_{\text{node}}$ is large since they have constant cost and the KS integration dominates the calculation when there are many primordial binaries. For the data copying and communication limit, a better communication algorithm [such as non-blocking communication as suggested by Dorband et al. (2003), which we will probably work on in the future], a higher network bandwidth between nodes and faster memory access is required.

In the common computer architecture today, the pure calculation operations for CPU is about two orders of magnitude faster than to access data from the host memory. For the non-shared memory parallelization like MPI, if the data communication consumption is larger than the calculation, the parallelization cannot improve the performance and sometimes even reduces the speed. Table 2 compares the calculation and communication costs for the regular force, the irregular force and the KS perturbation calculations. The ratio of calculation cost to communication cost, $R_c$, for the regular force is proportional to the full particle number $N$. Thus, the GPU and MPI parallelization for the regular force gives a very good scaling. For the irregular force, $R_i$ is proportional to the average neighbour number. When there are many neighbours, the MPI parallelization is good. For typical star cluster simulations, the neighbour number $N_c$ is a few hundred, thus it is acceptable. In nbody6+++, the data movement and MPI communication are significant for the irregular force (Fig. 7). For KS perturbation calculation, $R_k$ is proportional to the average perturber number $N_p$, which is usually quite small (less
than 100). Thus, MPI parallelization for KS can be inefficient. The reason for the small \(N_p\) is that usually in star cluster simulations, a large fraction of the KS binaries is unperturbed with \(N_p = 0\), and perturbed KS binaries also tend to have small \(N_p\) (otherwise they would be terminated or transformed to hierarchical systems). Therefore, to get good performance of KS parallelization, the unperturbed and perturbed KS parts should be treated separately, since unperturbed KS only needs few operations and should avoid communication when parallelized (shared memory parallelization such as OpenMP or MPI-3). We are working on this and will show our KS parallelization method and benchmarks in a future publication.

There is also another effort to parallelize KS with block time-steps (Nitadori 2014, private communication).

We also find that the KS initialization and termination can be costly when there are wide binaries that frequently switch between KS and Hermite solutions. As discussed in Section 3.3.2, during the KS initialization and termination, the force and its first three derivatives need to be renewed for centre-of-mass particles or two components [cost of \(O(N)\)] and the neighbour list of every particle and perturber list of KS pairs should be updated with new particle index [cost of \(O(N(N_p))\)]. The regular part can be improved by using existing values instead of a direct calculation. The latter can be improved by using a reverse neighbour list for fast searching which particle has the KS pair as its neighbours. However, this requires large memory cost and coding effort.

When testing the code performance in computer clusters, we usually use the empty nodes where no other tasks are performed simultaneously. But for the applications, whether we can use scheduling whole nodes depends on the task management system in the clusters. Some clusters, such as ‘Laohu’ at NAOC and ‘Milkyway’ at the Jülich Computing centre, only allow very few CPU cores for GPU tasks (1–2 CPU cores per GPU) and all other CPU cores in the same nodes are reserved for pure CPU tasks. NBODY6++GPU is not suitable for these kinds of clusters since it relies on heavy calculation on GPU (irregular and KS integration, data movements; see Fig. 7). Moreover, in the shared nodes, different tasks compete with each other for network bandwidth, CPU loading and host memory. This sometimes results in a serious load imbalance: the MPI barrier time (Table 1) covers almost half of the total computing time. The only solution is to use computing clusters in which GPU nodes can be fully occupied by one GPU task each time.

Both NBODY6 and NBODY6++ have been developed over a long time. The codes have become more and more complicated which makes it difficult for beginners. Therefore, we also present documentation for the new version of NBODY6++GPU. The document includes a detailed description of all input parameters and output data and will be updated with more details and new implementations. We also show several important differences between NBODY6++GPU and NBODY6-GPU in the Appendix.

The future improvements of the codes and hardware may lead to simulating even larger particle numbers, e.g., for nuclear star clusters using more GPU nodes appears feasible. The key to keep total wall clock times reasonable will be further optimization of communication and data management, especially for particles with very small time-steps near a central black hole. Also, bandwidth and latency of communication hardware may help to gain one more order of magnitude, but not to reach the Exaflop/s regime. For the latter, hybrid codes seem more appropriate, which treat a large number of particles in the outskirts self-consistently, but not with full \(N^2\) accuracy of the force computation (see for recent examples, e.g. Meiron et al. 2014; Karl et al. 2015).

---

**Table 1.** The definitions of abbreviations for all figures.

| Abbreviation | Definition |
|--------------|------------|
| Reg.         | Regular integration (force) and neighbour list determination |
| Irr.         | Irregular integration [force, prediction (AVX/SSE version) and correction] |
| Pred.        | Neighbour (non-AVX/SSE version) and all particles (for regular force) prediction |
| Init.B       | Initialization of active particle list for block time-step |
| Move         | Particle data copy prepared for MPI communication |
| Comm.I       | MPI communication for irregular integration |
| Comm.R       | MPI communication for receiving integration |
| Send.I       | Particle data copy for AVX/SSE irregular force calculation |
| Send.R       | Particle data copy for GPU regular force calculation |
| Adjust       | Energy checking, adjustment of parameters and data results |
| KS           | KS regularization calculation (binary and hierarchical systems) |
| Barr.        | MPI communication barrier waiting time due to the imbalance and network traffic between different nodes |

**Table 2.** Estimation of calculation and communication cost.

| Cost          | Regular force | Irregular force | KS perturbation |
|---------------|---------------|-----------------|-----------------|
| Calculation   | \(O(N_i)\)     | \(O(N(N_p))\)   | \(O(N_i(N_p))\) |
| Communication | \(O(N_i)\)     | \(O(N_i)\)      | \(O(N_i)\)      |

Notes. \(N_i\): active particle number. \(N\): full particle number. \(N_p\): average neighbour number. \(N_p\): average perturber number for KS.
7 CONCLUSIONS

Direct numerical simulations of star clusters contribute significantly to the theoretical understanding of star cluster dynamics. Due to hardware and software limits, direct N-body simulations of real globular clusters with large number of particles have been a major challenge for many years. Sugimoto et al. (1990) pointed out that direct numerical simulations of globular star clusters could not be completed for the next decades unless there are breakthroughs in parallel computing which violate Moore’s law. After that, many efforts were made to reach this goal by using specially designed acceleration hardware (GRAPE and GPU).

In this paper, we present NBODY6++GPU. It combines for the first time the massively parallel multi-node code (MPI parallelized) NBODY6++ (Spurzem 1999; Hemsendorf et al. 2003) with the GPU and AVX/SSE acceleration on each node, using the libraries of Nitadori & Aarseth (2012). We discuss the performance tests (Figs 4–7) and new algorithms (Figs 1, 2 and 8) to accelerate the NBODY6++GPU. For the non-binary case, the overall scaling is good up to 16 nodes (320 CPU cores and 32 NVIDIA K20x GPUs including 86016 GPU cores) with a speed up of 400 up to 2000 depending on the particle numbers. The speed up is mainly achieved by the usage of GPUs to accelerate the long-range (regular) gravitational forces, which gives about 33 times faster force calculation (Fig. 4). The AVX/SSE increases the speed of prediction of positions and velocities and neighbour (irregular) forces by a factor of 3. We also worked on the consistency of the code when combining several parallel methods together to ensure the stability. When GPU and AVX/SSE accelerate the force calculation very efficiently, other parts become bottlenecks of performance, such as time-step scheduling and stellar evolution. We designed new algorithms to improve these parts.

We have demonstrated how NBODY6++GPU can simulate a realistic globular cluster with one million particles, stellar evolution and 5 per cent primordial binaries for several Gyr (one half-mass crossing time requiring about an hour computational time; see Fig. 9). A few more models are currently in progress and we will report the detailed results of these simulations in future publications. With our final code version, which is publicly available, we can claim to have finally reached the goal of Sugimoto’s ‘dream’ of 1990. A million-body cluster can be simulated for about 20 crossing times in one day on 320 cores with 32 GPUs. In the future, with the faster bandwidth and latency of hardware as well as optimizations of communication and data management, even larger system like the nuclear star clusters may be simulated by direct N-body codes.

The previous paragraphs show that our contribution to this would be impossible without the achievements of our predecessors and collaborators; in particular the current dominance of GPU hardware has been assisted by the development of GRAPE software over the last few decades which finally could be ported to GPU without fundamental problems.

ACKNOWLEDGEMENTS

This work has been partly funded by National Natural Science Foundation of China, No. 11073025 (RS). We acknowledge support through the Silk Road Project at National Astronomical Observatories of China (NAOC, http://silkroad.bao.ac.cn).

RS and PB are grateful for support by the Chinese Academy of Sciences Visiting Professorship for Senior International Scien-

REFERENCES

Aarseth S. J., 1985, in Brachill J. U., Cohen B. I., eds, Multiple Time Scales. Academic Press, New York, p. 377
Aarseth S. J., 2003, Gravitational N-Body Simulations. Cambridge Univ. Press, Cambridge
Ahmad A., Cohen L., 1973, J. Comput. Phys., 12, 389
Belczynski K., Kalogera V., Bulik T., 2002, ApJ, 572, 407
Bellemann R. G., Bédorf J., Portegies Zwart S. F., 2008, New Astron., 13, 103
Berczik P., Spurzem R., Zhong S., Wang L., Nitadori K., Hamada T., Veles A., 2013a, Lecture Notes in Computer Science, Vol. 7905. Springer-Verlag, Berlin, p. 13
Berczik P., Spurzem R., Wang L., 2013b, Third International Conference ‘High Performance Computing’, HPC-UA 2013, p. 52
Chandrasekhar S., 1942, Chicago Ill. The University of Chicago press, Chicago, IL
Dorband E. N., Hemsendorf M., Merritt D., 2003, J. Comput. Phys., 185, 484
Eldridge J. J., Tout C. A., 2004, MNRAS, 353, 87
Gaburov E., Harfst S., Portegies Zwart S., 2009, New Astron., 14, 630
Harfst S., Gualandris A., Merritt D., Spurzem R., Portegies Zwart S., Berczik P., 2007, New Astron., 12, 357
Harris W. E., 1996, AJ, 112, 1487
Heggie D. C., 2014, MNRAS, 445, 3435
Heggie D. C., Mathieu R. D., 1986, Lecture Notes in Phys. Vol. 267. The Use of Supercomputers in Stellar Dynamics. Springer-Verlag, Berlin, p. 233

2 We use Subversion and Github to manage NBODY6++GPU. The beta version can be downloaded by commands `svn co http://silkroad.bao.ac.cn/repos/betanb6' or `git clone https://github.com/lwang-astro/betanb6pp.git'
APPENDIX A: DIFFERENCES BETWEEN NBODY6++ AND NBODY6

There are several differences with NBODY6. Table A1 lists some of the most important. The manual in the NBODY6++/GPU code directory gives more details.

| Subroutine | NBODY6++ | NBODY6 |
|------------|----------|--------|
| Installation | Use configure script (see GPU Autoconf software) | Use Makefile |
| Parallelization | Can enable/disable features among MPI, GPU, OpenMP and AVX/SSSE, except AVX/SSSE requires OpenMP enabled | Use GPU, OpenMP and AVX/SSSE together or OpenMP with AVX/SSSE (only OpenMP or GPU with OpenMP are not supported) |
| Data files | Rename most of output data files, change contents of some files and describe all data in a manual | Describe a document |
| Basic data initialization | Support different reading data format (see the manual for option KZ(22)) | Support N-body and astronomical unit data format |
| Primordial binary initialization | binpop.[f/F] | Support period distribution (Kroupa 1995) |
| Neighbour criterion | regcor_gpu.f gpucor.f | Adjust neighbour number to input parameter NNBOPT |
| Stellar evolution | kick.[f/F] | Use Maxwellian distribution of neutron star kick velocity with velocity dispersion 265 km/s (one dimension; Hobbs et al. 2005) |
| | hrdiag.f kick.[f/F] | Use mass fall back for black hole kick (increase the remnant mass and reduce kick velocity; Belczynski, Kalogera & Bulik 2002) |
| | brake4.f | – |
| Galactic tidal force | xtnlf.f fbulge.f | Support point-mass + disc + halo + Plummer model |

Note. 1 http://www.gnu.org/software/autoconf/

This paper has been typeset from a T\textsc{e}X/\textsc{b}e\textsc{x} file prepared by the author.