Menon et al. *Computational Astrophysics and Cosmology* (2015) 2:1
DOI 10.1186/s40668-015-0007-9

Computational Astrophysics and Cosmology
a SpringerOpen Journal

RESEARCH Open Access

Adaptive techniques for clustered N-body cosmological simulations

Harshitha Menon[1*], Lukasz Wesolowski[1], Gengbin Zheng[1], Pritish Jetley[1], Laxmikant Kale[1], Thomas Quinn[2] and Fabio Governato[2]

Abstract

ChaNGa is an N-body cosmology simulation application implemented using Charm++. In this paper, we present the parallel design of ChaNGa and address many challenges arising due to the high dynamic ranges of clustered datasets. We propose optimizations based on adaptive techniques. We evaluate the performance of ChaNGa on highly clustered datasets: a $z \sim 0$ snapshot of a 2 billion particle realization of a 25 Mpc volume, and a 52 million particle multi-resolution realization of a dwarf galaxy. For the 25 Mpc volume, we show strong scaling on up to 128*K* cores of Blue Waters. We also demonstrate scaling up to 128*K* cores of a multi-stepping run of the 2 billion particle simulation. While the scaling of the multi-stepping run is not as good as single stepping, the throughput at 128*K* cores is greater by a factor of 2. We also demonstrate strong scaling on up to 512*K* cores of Blue Waters for two large, uniform datasets with 12 and 24 billion particles.

Keywords: computational cosmology; scalability; performance analysis; dark matter

1 Introduction

Simulating the process of cosmological structure formation with enough resolution to determine galaxy morphologies requires an enormous dynamic range in space and time. Star formation (SF) is concentrated in dense gas clouds the size of just a few parsecs, while the assembly of galaxies happens over billion of years, driven by large scale structures extending over megaparsecs.

Constraints on cosmology are tightest on scales of tens of megaparsecs and larger due to observations of the Cosmic Microwave Background, giving us detailed initial conditions (Ade et al. 2013); however, our knowledge of the nonlinear evolution of the Universe and of the properties of galaxies is still imperfect, because the detailed properties of Dark Matter (Brooks 2014) and of SF (Pontzen and Governato 2014) remain only partially understood. On the other hand, simulations of large volumes of the Universe (Davis et al. 1985; Springel et al. 2005), and of individual galaxies at high resolution (Guedes et al. 2011; Hopkins et al. 2013) have been fundamental in putting the standard hierarchical Cold Dark Matter dominated model (ΛCDM) on a robust footing (Frenk and White 2012). Further understanding requires numerical simulations of increasing dynamical range, mass and spatial resolution and physical complexity, providing a powerful incentive to develop ever more sophisticated parallel codes (Vogelsberger et al. 2012; Kim et al. 2014).

Scaling such codes to large processor count requires overcoming not only spatial resolution challenges, but also large ranges in timescales. In this paper, we compare two approaches to handling this problem. The first approach involves using different time steps for different particles in relation to their dynamical time scales, leading to an algorithm that is challenging to parallelize effectively. An alternative approach, using a single, uniformly small time step for all particles, leads to more computation, but is simpler to parallelize.

This paper presents the design of ChaNGa, a parallel *n-body* + *SPH* cosmology code for the simulation of astrophysical systems on a wide range of spatial and time scales. Most of the physical modules of ChaNGa have been imported from the well established tree + SPH code GASOLINE and we refer the readers to the existing literature (Wadsley et al. 2004, 2008, Governato et al. 2014) for more details.

*Correspondence: gplkrsh2@illinois.edu
[1]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, USA
Full list of author information is available at the end of the article

In this paper we focus on the optimizations implemented in ChaNGa that allow it to scale to large numbers of processors, and address the challenges brought on by the high dynamic ranges of clustered datasets. We will begin with an overview of the field and place the approach taken by ChaNGa in the context of published material. We then briefly summarize some specific features of ChaNGa (some imported from GASOLINE), including force softening, smooth particle hydrodynamics, star formation, and multi-stepping. The parallel design of ChaNGa, based on over-decomposition of work, allowing a parallel runtime system to dynamically balance the load, is presented next, along with descriptions of the phases of the computation. To set the context, and a baseline, for the optimizations presented, we first describe the single-stepping performance on relatively uniform data-sets. The clustered data-sets are then introduced, and a series of performance challenges along with strategies and optimizations developed to overcome them are described. These are accompanied by detailed performance analysis using the Projections performance visualization tool (Kalé and Sinha 1993). As of Spring 2014 our performance evaluation runs demonstrate scalability to over 131,000 processor cores on NCSA's Blue Waters and up to a 3× speedup over the single-stepping algorithm.[a]

2 Current state of the art

Because of the computational challenge and the non-trivial algorithms involved, cosmological N-body simulations have been an extensively studied topic over the years. In order to frame our work in ChaNGa, we review some of the recent successes in scaling cosmological simulations on the current generations of supercomputers. However, direct comparison of the absolute performance among different codes is difficult. Different choices of accuracy criteria for the force evaluations and the time integration will have a big impact on performance, and the choices for these criteria will be determined by the various scientific goals of the simulation. For example, understanding the development of structures at very high redshift (e.g. Ishiyama et al. 2012) will present different parameter and algorithm choices than simulations that model the observations of current large scale structure (e.g. Habib et al. 2013).

2HOT (Warren 2013) is an improved version of the HOT code which has been developed over the past two decades. It uses an Oct-tree for gravity, and its gravity algorithm is very similar to that of ChaNGa. This code demonstrates near perfect strong scaling up to 262 thousand cores on Jaguar with a 128 billion particle simulation, implying 500,000 particles per core at the largest core count. The actual size of the scaling simulation (in Gigaparsecs) was not reported, but can be presumed to be a box of order 1 Gigaparsec based on the other simulations presented in Warren (2013). 2HOT does implement a multi-step time-stepping algorithm, although it is not clear whether particles have individual time steps, and performance for the multi-step method was not presented.

The HACC (Habib et al. 2013) framework scales to millions of cores on a diverse set of architectures. It uses a modified TreePM algorithm: an FFT based particle-mesh on the large scales, a tree algorithm on intermediate scales and particle-particle on the smallest scales. HACC has been demonstrated to scale with near perfect parallel efficiency up to 16,384 nodes on Titan with 1.1 trillion particles, and up to 1.6 million cores on Sequoia with 3.6 trillion particles. These are weak scaling results, typically with millions of particles per core. They also demonstrated strong scaling up to 8,182 nodes on Titan and 16,384 cores on Sequoia.

The GreeM code (Ishiyama et al. 2012) demonstrates scaling of a trillion particle simulation to 82,944 nodes (663,522 cores) of the K computer, implying 1.5 million particles per core. This code also uses a TreePM algorithm with a hand-optimized particle force loop and a novel method to parallelize the FFT. They report that despite the new parallelization method, the FFT remains the bottleneck in their TreePM code. They also employ a multi-step method that splits the PM and particle forces, but the particles do not have individual time steps.

The GADGET-3 TreePM code (based on GADGET-2 (Springel 2005)) was used to perform a large scale structure, DM-only simulation (the 'Millenium XXL') on 12,288 cores using 303 billion particles (Angulo et al. 2012). With over 16 million particles per core, special effort was needed to optimize the memory usage of the code because the simulation was limited by memory resources.

Most of these cosmological N-body codes with published performance data scale to millions of cores with almost perfect parallel efficiency, given very large problem size (typically trillions of particles). However, it becomes even more challenging to simulate a relatively smaller problem size with higher resolution using large numbers of cores. This is due to the fact that the distribution of the particles in the simulated system tends to become more non-uniform as resolution increases, leading to load imbalance and difficult scaling. The addition of hydrodynamics and cooling only exacerbates this problem. Recent projects that coupled gravity with hydrodynamics in galaxy formation simulations and scaled past a few thousand cores include EAGLE and Illustris. The codes used (GADGET-3 and AREPO (Springel 2010)) share many of the features of ChaNGa that are necessary for galaxy formation, including individual time steps for particles, gas dynamics, and star formation/feedback prescriptions (Schaye et al. 2014; Vogelsberger et al. 2014). While some codes handle non-uniform distributions well (e.g. GADGET-3) they have not been shown yet to scale to large (100,000 or greater)

core counts. Hence, to our knowledge, ChaNGa is the first code to explicitly tackle both the uniform and highly clustered simulations with extremely large scaling. This is achieved by several techniques including multi-stepping and large scale dynamic load balancing described below.

3 ChaNGa

The N-body/Smooth Particle Hydrodynamics (SPH) code ChaNGa (Jetley et al. 2008, 2010), is an application implemented using Charm++. ChaNGa includes a number of features appropriate for the simulation of cosmological structure formation, including high force accuracy, periodic boundary conditions, evolution in comoving coordinates, adaptive time-stepping, equation of state solvers and subgrid recipes for star formation and supernovae feedback. The code is also being compared with similar codes in the AGORA comparison project (Kim et al. 2014). Cosmology research based on ChaNGa includes modeling the impact of a dwarf galaxy on the Milky Way (Purcell et al. 2011), modeling the intracluster gas properties in merging galaxy clusters (Ruan et al. 2013) and distinguishing the role of Warm Dark Matter in dwarf galaxy formation and structure (Governato et al. 2014). In this section we describe the features of ChaNGa, particularly as they relate to cosmological structure formation. In addition to the physics features described below, ChaNGa has a number of usability features required for pushing a large simulation through a production system, such as the ability to efficiently checkpoint and restart on a different number of processors.

3.1 Gravitational force calculation

The gravitational force calculation is based on a modified version of the classic Barnes-Hut algorithm (Barnes and Hut 1986). Details of our modifications are described in Section 4, and many of our optimizations are taken from PKDGRAV (Stadel 2001), upon which our gravity calculation is based. As in PKDGRAV, we choose to expand to hexadecapole order the multipoles used for evaluating the far field due to a mass distribution within a tree node. For the force accuracies required for cosmological simulations, better than 1 percent (Power et al. 2003; Reed et al. 2003), this higher order expansion is more efficient (Quinn et al. 2013).

3.2 Force softening

When simulating dark matter and stars, the goal is to understand the evolution of a smooth distribution function that closely approaches a Boltzmann collisionless fluid. As the N-body code is sampling this distribution using particles, a more accurate representation of the underlying mass distribution is obtained if the particles are not treated as point masses, but instead have their potential softened (Dehnen 2001). Softened forces are also of practical use since they limit the magnitude of the inter-particle force. Typically, the softening length is set at the inter-particle separation at the center of DM (Dark Matter) halos (Power et al. 2003).

Calculating the non-Newtonian forces introduced by softening adds a complication to the multipole calculation: Newtonian forces have symmetries which greatly reduce the complexity of higher order multipoles, and the number of components of the multipole moments that need to be stored. ChaNGa implements softening using a cubic spline kernel, whose compact support means this complexity is not needed beyond a specified separation (convergence with Newtonian gravity is formally achieved at two softening lengths). Furthermore, rather than evaluating the more complex multipoles when softening is involved, ChaNGa evaluates all forces involving softening using only the monopole moments, using a stricter opening criterion to maintain accuracy.

3.3 Periodic boundary conditions

In order to efficiently and accurately simulate a portion of an infinite Universe, we perform the calculation assuming periodic boundary conditions. Because of the long range nature of gravity, the sum over the infinite number of periodic replicas converges very slowly. ChaNGa accelerates this convergence using Ewald summation (Ewald 1921), implemented similarly to Ding et al. (1992) as more fully described in Stadel (2001). This technique has the advantage that the non-periodic force calculated from the tree-walk is not modified, and therefore is simple and fast. We have demonstrated in Reed et al. (2003) that, with suitable choices of the accuracy criterion, the force errors from this method do not compromise the growth of large scale structure.

3.4 Multi-stepping

In order to efficiently handle the wide range of timescales in a non-uniform cosmological simulation, ChaNGa allows each particle to have its own time step. In order to amortize overheads associated with the force calculation, such as tree building, the time steps are restricted to be power-of-two subdivisions of the base time step. Details of this scheme, including how to integrate the equations of motion in coordinates that follow the expansion of the Universe, are described in Quinn et al. (1997). This scheme is also identical to that implemented in GADGET-2; see Springel (2005) for tests of its accuracy.

3.5 Smooth particle hydrodynamics

Despite being a small fraction of the energy density of the Universe, baryons play a significant role in the evolution of structure. Not only are they the means by which we can measure structure (e.g. via star light), they can also directly influence the structure of the dark matter via gravitational coupling (Pontzen and Governato 2012). Therefore, following the physics of the baryonic gas is essential

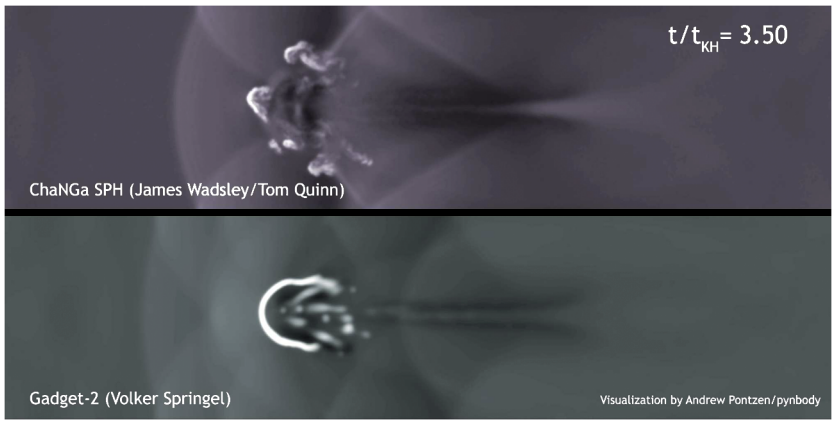


Figure 1 Updated modeling of gas physics in ChaNGa. Central density slices of the time evolution of a high density cloud in pressure equilibrium in a wind. Time is in units of the Kelvin-Helmholtz growth time. ChaNGa (top) vs GADGET-2 (Springel 2005) (bottom). The color density map shows how with the new SPH formulation of pressure gradients, artificial surface tension is suppressed and instabilities rapidly mix the 'blob' with the surrounding medium, while poor handling of contact discontinuities preserve the blob in the now obsolete SPH implementation of GADGET-2. We have verified that ChaNGa gives results quite similar to alternative hydro codes, as the adaptive mesh refinement code ENZO (Collins et al. 2010). This figure was produced with Pynbody (Pontzen et al. 2013).

for accurate modeling of structure formation. ChaNGa uses Smooth Particle Hydrodynamics (SPH) to solve the Euler equations with an implementation that closely follows (Wadsley et al. 2004). Since SPH is based on particles, implementing it is a natural extension of the algorithms to calculate gravity on a set of particles. In particular, the tree structure used for the Barnes-Hut algorithm is used to find the near neighbor particles needed for the SPH kernel sums. SPH is also relatively communication intensive compared to gravity, so we utilize the Charm++ runtime system to adaptively overlap the communication latencies from SPH with the floating point operations needed by gravity. The current implementation of SPH in ChaNGa closely follows techniques already published by independent groups and includes an updated treatment of entropy and thermal diffusion (Wadsley et al. 2008; Shen et al. 2010), pressure gradients[b] and timestepping (Durier and Dalla Vecchia 2012). This last features ensures that sudden changes in the particle internal energy, e.g. caused by feedback, are captured and propagated to neighboring particles by shortening their time step. These improvements lead to a marked improvements in the treatment of shocks (as in the Sedov-Taylor blastwave test), and cold-hot gas instabilities. A qualitative example is shown in Figure 1, where the classic 'blob' test compares ChaNGa with GADGET-2.

As this paper focuses specifically on the scaling performance of ChaNGa we refer to existing works (Wadsley et al. 2004; Governato et al. 2014) and Wadsley et al. (in prep.) for tests of this SPH implementation.

3.6 Star formation and feedback

Again, a necessary component of simulating structure formation is predicting the light distribution. Hence, we need a prescription for where the stars are forming. Furthermore, it is clear that star formation is a self-regulating process due to the injection of energy from supernova, ionizing radiation and stellar winds into the star-forming gas. These processes are all happening well below the resolution scale of even the highest resolution cosmological simulations, so a sub-grid model is needed to include their effects. ChaNGa includes the physics of metal lines and molecular hydrogen cooling (Shen et al. 2010; Christensen et al. 2012) and feedback from supernovae (SNe). In ChaNGa, we have implemented the 'blast-wave' and 'superbubbles' feedback models described in Stinson et al. (2006) and Keller et al. (2014), respectively. In both models SF occurs in high gas density regions, and the time and distance scales for energy injection into the gas is determined by physically motivated models. The 'blastwave' prescription follows an analytic model of the Sedov blast wave, and it has allowed us to successfully model a number of trends in galaxy populations including the Tully-Fisher relation (Governato et al. 2007), the mass-metallicity relation (Brooks et al. 2007), the stellar mass-halo mass relation (Munshi et al. 2013) and the formation of DM cores in dwarf galaxies (Governato et al. 2012).

4 Parallelization approach

In ChaNGa, the particle distribution in space is organized in a hierarchical tree structure where each node represents a portion of the 3D space containing the particles in that volume. The root node represents the entire simulation space and the children represent sub-regions. The

leaf nodes of the tree are *buckets* containing a small set of particles.

4.1 Domain decomposition

During domain decomposition, particles are divided among objects called *tree pieces* (or chares in the context of Charm++) which are mapped onto processors by the runtime system. Typically, there are more tree pieces than the number of processors, and this over-decomposition allows the benefits of the overlapping of communication with computation and the load balancing features of Charm++.

ChaNGa supports various domain decomposition techniques, which have been evaluated previously (Sharma 2006). We used space-filling curve (SFC) decomposition for the results in this paper as that is the method currently used for most scientific studies with ChaNGa.

The goal of this scheme is to identify a set of splitting points (*splitters*) along the space filling curve such that each range contains approximately equal numbers of particles. The algorithm used to identify the splitter keys is similar to the parallel histogram sort (Solomonik and Kale 2010). First, a single master object calculates a set of splitters along the SFC that partition the simulation domain into disjoint areas of roughly equal volume. It then broadcasts the splitter keys to all the tree pieces. The tree pieces evaluate the count of particles for each bin, which is reduced across all tree pieces back to the master process. The candidate keys are then adjusted based on the contributions received, and new splitters are broadcast for any bins that are not sufficiently close to an optimal partition. This process is repeated until a suitable set of splitter keys is determined such that all the bins have roughly equal numbers of particles. After the splitter keys are identified, the particles are globally distributed to tree pieces according to the splitters, where each bin corresponds to one tree piece.

4.2 Tree build

After the particles have migrated and domain decomposition is finished, each tree piece builds its tree independently. The tree build is done in a top-down manner. The algorithm starts from the root, which contains the entire simulation space, and proceeds downwards to the leaves, which are buckets containing a small number of particles, typically 8 to 12. A tree piece has information about the extent of the domain held by other tree pieces; this information is used in the tree building process. A spatial binary tree is constructed by bisecting the bounding box containing particles in the given volume. The tree building process bisects each node, starting at the root, into children, which represent sub-regions within the space, until a leaf node is constructed. If a node in the tree held by a tree piece contains particles in another tree piece, then that node becomes a boundary node.

We also take advantage of the fact that a tree piece can access other tree pieces within the same address space. All the tree pieces within the same address space are merged. After the merge, each tree piece has read-only access to the tree data structure that is constructed by merging multiple tree pieces. For additional details, we refer the reader to Jetley et al. (2008).

4.3 Tree traversal and gravity

The goal of tree traversal is to identify for each bucket of particles in the tree the list of nodes and particles whose information is needed for the gravity calculation. These *interaction lists* are constructed on a per bucket basis to amortize the overhead of the tree traversal.

Another optimization that is implemented in ChaNGa to improve the performance of the gravity phase is based on the observation that nearby buckets tend to have similar interaction lists (Stadel 2001). The algorithm constructs the interaction list of a parent node before proceeding to the children, and maintains a *checklist*, passed down the tree, that reduces the number of nodes that need to be evaluated at each level.

Tree traversal requires remotely accessing nodes which are part of tree pieces on other processors. To optimize this remote data access, we have implemented a software cache, as shown in Figure 2. The *Cache Manager* serves node and particle requests made by a tree piece. If a node request is missed in the cache, then a request is sent to the corresponding tree piece. If there is already an outstanding request in the cache, no additional request is sent. When the response arrives, the requestors are informed and the walk resumes. This improves the performance by hiding the latency of remote requests and by reducing the number of messages sent and received for the remote node. To further reduce cache misses, we also perform a prefetch walk which obtains remote node information.

To effectively overlap communication and computation, we divide the tree traversal into local and remote parts. A local traversal is done on the portion of the tree which is within the local address space whereas a remote traversal is done on the remaining part of the tree and requires communication between the tree pieces. We use prioritization to give precedence to the remote traversal, which requires communication, over the computation-dominated local traversal. When the remote walk has sent out requests for the node and is waiting for the response, the local walk can be done. This enables overlap of communication with local computation and helps mask message latency. Figure 2 diagrams the gravity calculation in ChaNGa with a software cache.

Sequential code in ChaNGa is also well optimized. In particular, we take advantage of single-instruction, multiple-data (SIMD) parallelism inherent in the force calculation to accelerate that part of the computation using FMA or SSE vector instructions.

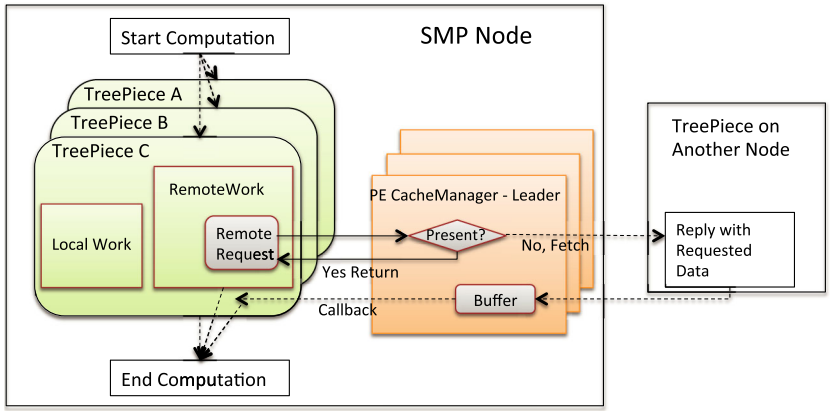


Figure 2 ChaNGa overview. An overview of the gravity force calculation in ChaNGa with a software cache.

5 Datasets and systems

We first describe the datasets used for our experiments and their characteristics. We have two large, uniform (Poisson distributed) datasets with 12 and 24 billion particles. Other than having periodic boundary conditions these two datasets are not particularly interesting for cosmology. We include them here to demonstrate the scaling of ChaNGa to large core counts. *cosmo25* is a more challenging dataset: it is a 2 billion particle snapshot taken from the end (i.e. representing the current, $z \sim 0$, very clustered, structure of the Universe) of a dark matter simulation of a 25 Megaparsec cube in a ΛCDM Universe. The force softening is 340 parsecs, and the simulation represents a challenge for load balancing. This simulation was evolved using ChaNGa from initial conditions at $z = 109$ based on cosmological parameters derived from the Planck data (Ade et al. 2013). The version of this simulation with gas dynamics and star formation is able to resolve the disks of spiral galaxies within this volume [Anderson et al., in preparation]. *dwarf* is our most challenging dataset: while it contains only 52 million particles spread throughout a 28.5 Megaparsec volume, most of the particles are in a single high resolution region in which a dwarf galaxy is forming. The mass resolution in this region is equivalent to having 230 billion particles in the entire volume, and the force resolution within this region is 52 parsecs. This is a high resolution version of the DWF1 simulation discussed in Governato et al. (2007). See the description of DWF1 in that paper for more details about the galactic and cosmological parameters of this simulation. While, as described above, ChaNGa is capable of handling hydrodynamics and star formation, in the benchmarks below we show the performance of dark matter only simulations. We will comment on SPH performance in the discussion.

We show the performance of ChaNGa on Blue Waters. Blue Waters is a hybrid Cray XE/XK system located at the National Center for Supercomputing Applications (NCSA). It contains 22,640 Cray XE6 nodes and 4,224 Cray XK7 nodes that include NVIDIA GPUs. Each dual-socket XE6 compute nodes contains two AMD Interlagos 6276 processors with a clock speed of 2.3 GHz and 64 GB of RAM.

6 Single stepping

We now describe essential optimizations required for scaling the simpler datasets that are not highly clustered, and evaluate their performance. Later sections will describe optimizations for clustered datasets.

6.1 Single stepping improvements

We observed that from-scratch domain decomposition is not required at every step, especially for datasets which are not highly clustered. After the initial domain decomposition, it needs to be performed only when there is an imbalance in the load of tree pieces. By reusing the previously determined splitters, we reduce the overhead incurred in finding the splitters as well as the number of particle migrations. We use an adaptive mechanism to determine when to perform the domain decomposition. In this approach, load statistics of the tree pieces are collected and domain decomposition is only performed if an imbalance is detected. Otherwise, only particle migration is done based on the previous splitters. We use the quiescence detection (Sinha et al. 1993) mechanism implemented in Charm++ to determine when all the migrations are finished.

In the unoptimized version of the code, the tree build requires all tree pieces to send the information about the first and the last particle in their domain, subject to the SFC. This information is used to determine ownership of nodes in the tree but requires heavy communication. We avoid this by using the boundary information to determine a set of candidate tree pieces which may have information about

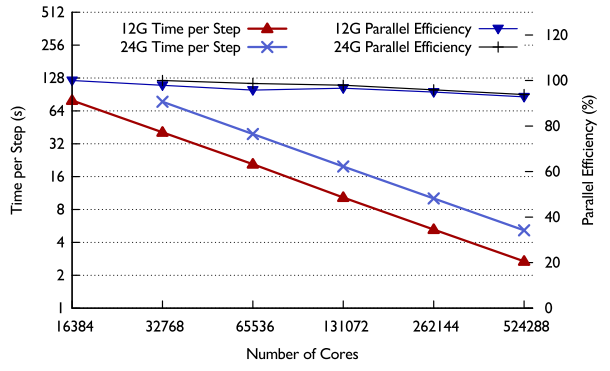


Figure 3 Performance on Blue Waters. Time per step and parallel efficiency for 12 and 24 billion particles on Blue Waters. Both the cases scale well achieving a parallel efficiency of 93%.

Table 1 Breakdown of time for 1 step in seconds for 12 billion particle (top half) and 24 billion particle (bottom half) datasets run on Blue Waters with the proposed optimizations

#cores	Gravity	DD	TB	LB	Total time
16,384	77.556	1.299	0.729	0.128	79.712
32,768	39.254	0.698	0.617	0.136	40.705
65,536	19.876	0.496	0.367	0.062	20.801
131,072	9.967	0.181	0.138	0.027	10.313
262,144	5.051	0.109	0.076	0.013	5.249
524,288	2.569	0.073	0.034	0.008	2.684
32,768	75.090	1.553	0.735	0.186	77.564
65,536	37.941	0.787	0.462	0.111	39.301
131,072	19.062	0.428	0.245	0.063	19.798
262,144	9.682	0.232	0.152	0.042	10.108
524,288	4.903	0.146	0.095	0.022	5.166

the required node. One of them is then queried and in case that tree piece does not have the information, it forwards it to the appropriate tree piece.

Since load balancing incurs overhead, it should be done sparingly. We use the *MetaBalancer* (Menon et al. 2012) framework in CHARM++ to determine when to invoke the load balancer. *MetaBalancer* monitors the application characteristics and predicts when the load balancing should be done. *MetaBalancer* invokes the load balancer when: (1) an imbalance is detected and (2) the benefit of load balancing is more than the cost incurred due to load balancing.

6.2 Performance

Figure 3 shows strong scaling results on up to 512K cores on Blue Waters evolving 12 and 24 billion particles. Our application exhibits almost perfect scaling up to the maximum number of cores. Each iteration consists of domain decomposition, load balancing, tree building and the force calculation. Table 1 shows the break down of the time per step into the different phases. For the simulation evolving 12 billion particles, we achieve 93% parallel efficiency at 512K cores with the time per step being 2.6 seconds. For the 24 billion particle simulation, we achieve 93.8% parallel efficiency with a time per step of 5.1 seconds. The efficiency is calculated with respect to 16K cores and 32K cores for 12 and 24 billion particles, respectively.

The good scaling of the gravity phase is due to the overlap of communication and computation, the improved tree walk algorithm using an interaction list, the software request cache, prefetching, and other optimizations. The time for domain decomposition also scales with the increase in number of cores. Table 1 shows, for the 12 billion particles at 512K cores on Blue Waters, that domain decomposition takes on average 73 ms per step. At 128K cores the domain decomposition is 9 times faster in comparison to the unoptimized version. This is due to the use of the adaptive technique to determine when to perform full domain decomposition. The tree build time also scales well and takes 34 ms at 512K cores. At 128K cores, the tree build is approximately 6 times faster than the unoptimized version. Similar trends are seen in the 24 billion particle simulation.

Table 2 Breakdown of time for 1 step in seconds for 12 billion particles (top half) and 24 billion particles (bottom half) dataset run on blue waters without the proposed optimizations

#cores	Gravity	DD	TB	LB	Total time
16,384	82.424	2.81	0.995	7.79	94.019
32,768	42.712	1.966	1.005	6.854	52.537
65,536	21.438	1.731	0.729	6.482	30.38
131,072	12.162	1.674	0.803	5.718	20.357
32,768	80.144	2.859	1.366	16.173	100.542
65,536	41.279	2.356	1.032	9.338	54.005
131,072	22.958	2.142	1.018	8.854	34.972

Table 2 contains the breakdown of the total time per step for the unoptimized version of the code. Comparing the results with Table 1, for the 12 billion particle simulation, we reduce the total time by 15 to 49%. For the 24 billion particle simulation, we reduce the total time per step by 22 to 43%. The reduction in time occurs for all phases of the application.

Figure 4 shows the time profile graph obtained using Projections (Kalé and Sinha 1993). This shows the average processor utilization over the course of one time step evolving 12 billion particles on 16K cores of Blue Waters. We can see that the local work, which is given a lower priority, overlaps with the communication needed for the higher-priority remote work, resulting in close to 100% processor utilization.

Figure 4 Processor utilization. Time profile graph which shows processor utilization over time for 16*K* cores on Blue Waters for 12 billion particles. This shows overlap of communication with computation to achieve high utilization.

7 Clustered dataset challenges

For datasets such as *dwarf*, the particle distribution is concentrated at the center of the simulation volume and therefore highly clustered. This creates many challenges in scaling, of which one of the most significant is communication imbalance. During the gravity phase, remote requests are sent for tree nodes that are not present in the local cache. In a clustered dataset, some tree nodes are requested many more times than others. This results in the tree pieces owning those tree nodes receiving a large volume of node request messages. Figure 5(a) shows the number of requests received by processors for the *dwarf* simulation at 8*K* cores on Blue Waters. We can see that a handful of processors receive as many as 30*K* messages. Even though there is overlap of communication with computation, this causes significant performance degradation. This is because, at this scale, there is not enough local computation to overlap seconds of delay in receiving messages. One way to mitigate this problem is to replicate the information that is being requested to prevent a few processors from being the bottleneck.

We replicate the information about the tree nodes on multiple processors ensuring that no single processor becomes overloaded. Before the gravity phase begins, tree pieces send their node information to a set of tree piece proxies on other processors. The responsibility of the tree piece proxy is to store the node information sent to it and handle requests for those nodes. When a tree piece needs to request for a remote node, it chooses randomly one of the tree piece proxies to send the request to. Figure 5(b) shows the number of messages received by the processors when four tree piece proxies are created for each tree piece. For an 8*K* core run on Blue Waters, replication reduces the maximum number of messages received from 32*K* to 4.2*K*, and the requests are better distributed among all processors. Figure 6 shows the time-profile graph where the x-axis is the time and y-axis is the processor utilization. Here, yellow regions constitute the local work, blue the Ewald and maroon the remote work. Note the idle time, in Figure 6(a), before the remote work begins which is due to the delay in receiving messages and the lack of local work overlap. Figure 6(b) shows the impact of replication. The remote work can start earlier due to a smaller delay in request messages. The local work overlaps with the communication until remote work is ready to start. This is a very good example that shows prioritization of remote work over local work and the overlap of communication with computation. Figure 7 shows the strong scaling performance for this dataset on core counts ranging from 1*K* to 16*K*. We compare the time for the gravity phase because the rest of the phases are the same in both cases. The gravity time is improved from 2.4 seconds to 1.7 seconds for 8*K* cores and from 2.1 seconds to 0.99 seconds on 16*K* cores. At 16*K* cores the parallel efficiency without replication is 48% cores whereas replication helps achieve an efficiency of 98%.

8 Multi-stepping challenges

A wide variation in mass densities can result in particles having dynamical times that vary by a large factor. In a single-stepping mode, good accuracy can only be achieved by performing the force calculation and particle position and velocity updates at the smallest timescale. However, hierarchical time stepping schemes can be used for a large

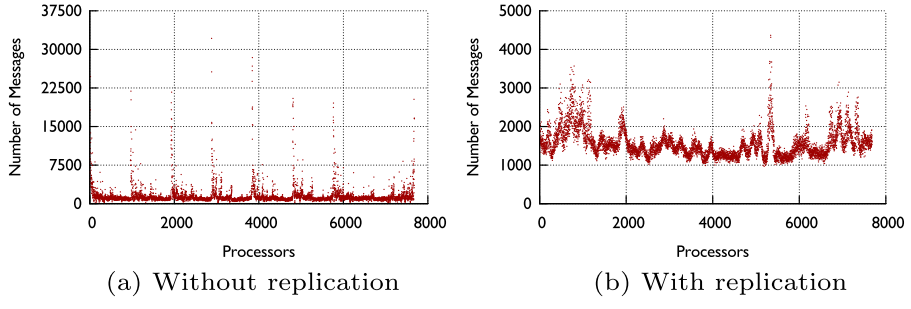
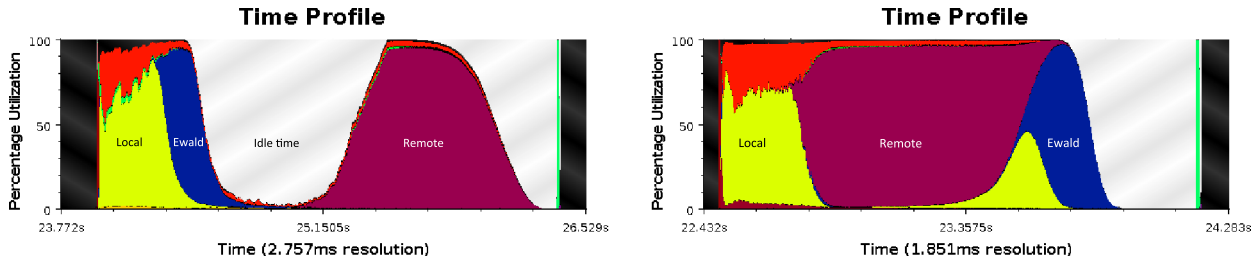


Figure 5 Messages with and without replication. Number of messages received by processors for a simulation of the *dwarf* dataset on 8*K* cores on Blue Waters. Note that replication reduces the maximum requests received by a processor from 30*K* to 4.5*K*.



(a) Without replication (b) With replication

Figure 6 Processor utilization with and without replication. Time profile graph showing processor utilization over time for the simulation of the *dwarf* dataset on 8*K* cores. Note the idle time without replication which is removed by the replication and the gravity time is improved from 2.4 seconds to 1.7 seconds.

dynamic range in densities at a small additional cost. We use adaptive time scales where forces are evaluated only on relevant particles instead of evaluating forces on all the particles at the smallest time scale. In a multi-step simulation, particles are assigned to time step rungs corresponding to the shortest time scale required for an accurate simulation. Rungs corresponding to short time scales are evaluated more frequently than those for long time scales.

Using multi-stepping for clustered datasets introduces a variety of challenges. The irregular distribution of particles in the simulation space as well as the division of particles into rungs creates severe load imbalance. In general, the challenge is higher for datasets with fewer particles. We discuss various optimizations that enable ChaNGa to scale a medium-sized 2 billion particle clustered dataset, *cosmo25*, on up to 128*K* cores on Blue Waters. Reaching this level of performance required overcoming challenges related to load imbalance, communication overhead with a decrease in computation per processor as well as the scalability of other phases of the simulation. Strong scaling of this nature will be required to run clustered cosmological simulations on future machines with hundreds of Petaflop/s performance, and presents a realistic proving ground for parallel strategy innovations.

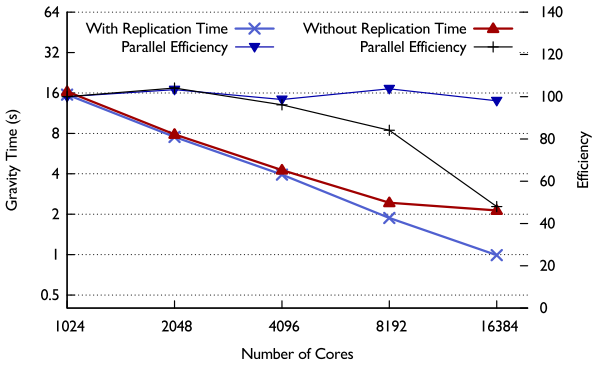


Figure 7 Performance for the *dwarf* simulation. Gravity time for the *dwarf* simulation on 8*K* cores on Blue Waters with and without the replication optimization.

8.1 Optimizations for the gravity phase

In a multiple time step simulation, the number of particles active in the fastest rung is typically only a fraction of the total number of particles being simulated. These active particles tend to be clustered, and therefore the distribution of particles among the tree pieces is highly imbalanced. One may consider performing from-scratch domain decomposition based on the active set of particles for these time steps but that results in large jumps of the domain boundaries. To prevent such sudden large variations of the boundaries, we perform from-scratch domain decomposition only when there is a significant number of particles active for that time step. But as one can imagine, this will result in tree pieces with a large variation in active particles and load. Figure 8 shows the distribution of the load on tree pieces for the fastest rung (rung 4) and the slowest rung (rung 0) of the *cosmo25* dataset. The slowest rung has tree pieces with loads distributed around the mean. But the fastest rung has only 2,405 tree pieces with active particles and some of them have a load which is 3,000 times the average load of tree pieces and 40 times the average load of the system. Even though periodic load balancing is performed to distribute the load, the maximum load of the system will be limited by the most overloaded processor which in this case is the one having the most loaded tree piece. At larger scales of 128*K* cores there is not enough work to be distributed among all the cores which results in significant degradation of performance. We propose two adaptive strategies to overcome this problem.

8.2 Intra-node work pushing

We use the SMP mode of Charm++ to take advantage of the shared memory multiprocessor nodes used in HPC systems (Mei et al. 2010). The SMP mode supports multithreading, where one Charm++ process is assigned per

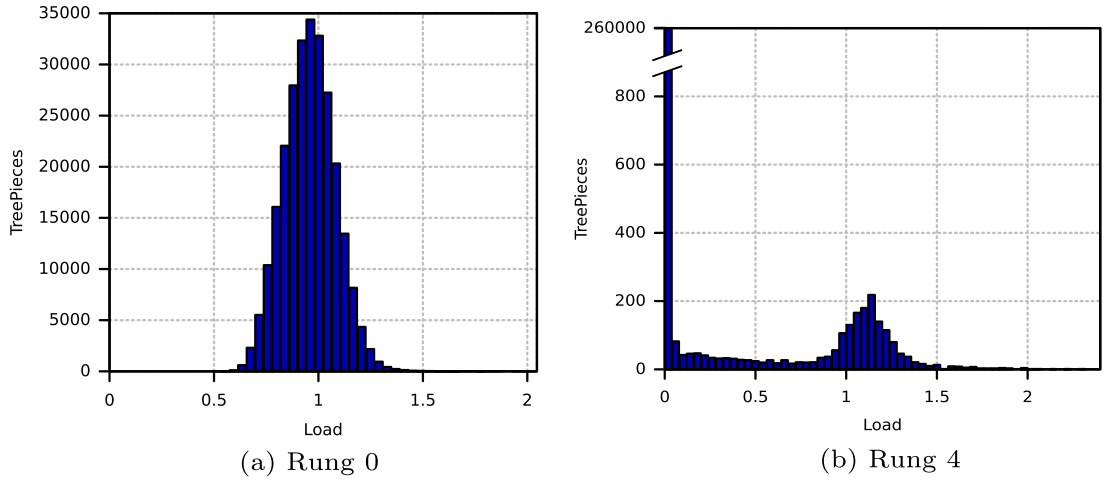


Figure 8 Load distribution. Distribution of tree piece load for rung 0 (slowest) and rung 4 (fastest). Rung 0 has loads distributed around the mean. Rung 4 has only 2,405 active tree pieces with a maximum load of 2.3.

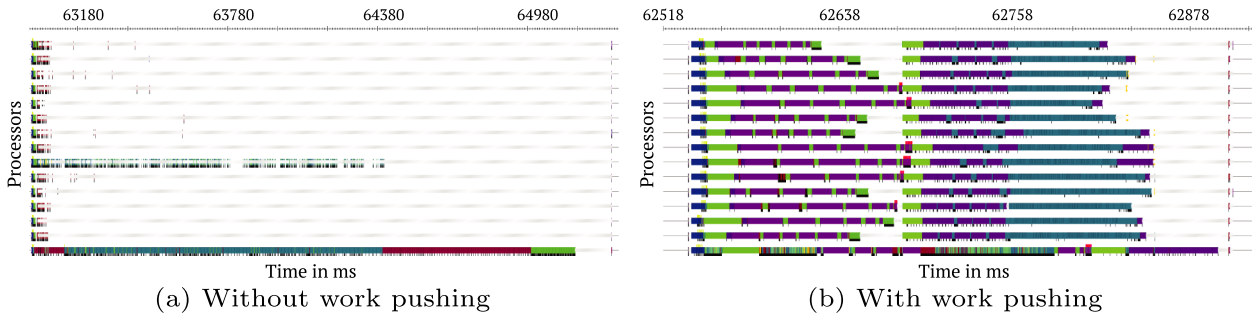


Figure 9 Performance analysis with and without work pushing. Time line profile for all the PEs (rows) on a SMP process for the 16*K* cores run. White shows idle time and colored bars indicate busy time. Work pushing achieves better distribution of work among PEs. The total time per step reduces from 2.3 seconds to 0.3 seconds.

SMP node, with a single thread mapped to each physical core. One thread within a node is normally assigned as a communication thread responsible for internode communication, while the rest are used as worker threads that implement *processing elements (PEs)*.

Within a CHARM++ SMP process, data can be shared via pointers. The load balancing strategy works in a hierarchical fashion. Details are given in Section 8.6, but in essence it first tries to achieve load balance among the SMP processes and then balances the load among cores within the SMP process.

LBManager, which is an object present on each PE, has information about the average load of the system and the load of other PEs on the same SMP process. The *LBManager*, on identifying that a PE is overloaded, instructs overloaded tree pieces at that PE to distribute the work among other less loaded PEs within the SMP process. A tree piece is responsible for calculating forces on a set of particles in its domain, grouped into buckets. We consider the bucket to be the smallest entity of work that can be distributed. PEs receiving a foreign bucket have access to the tree and all the data structures of the owner tree piece so that they can perform the tree traversal and gravity force calculations for the foreign bucket. Once the force calculations are done, the foreign bucket is marked as complete and the original PE is informed. Once all the foreign and local buckets are completed, the tree piece is done with the gravity calculations.

This work pushing adaptive strategy reaps the most benefit for time steps where the fastest rung is active. For the slowest rung, the forces on all the particles need to be calculated, and the load balancing is very similar to that in single stepping runs. Figure 9(a) shows the time-line view from the projections tool (Kalé and Sinha 1993) for rung 4 (the fastest rung). Here, each line corresponds to a PE and colored bars indicate busy time while white shows idle

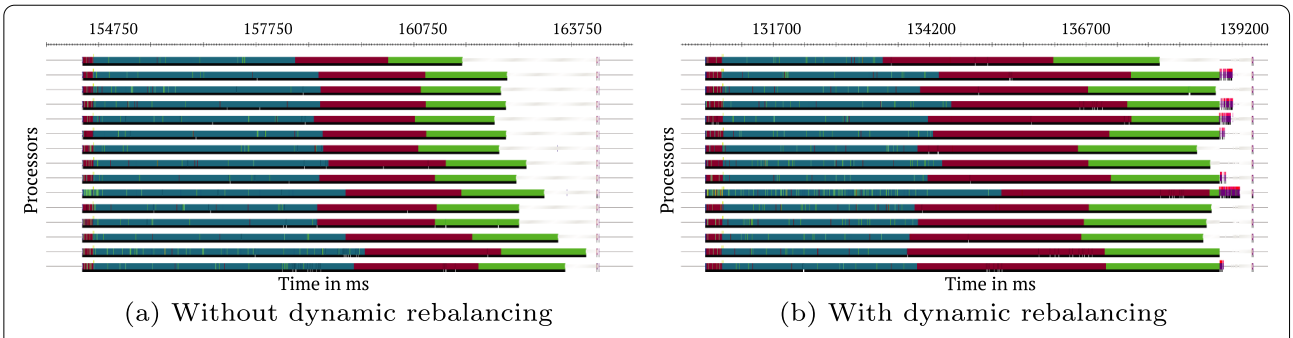


Figure 10 Performance analysis with and without dynamic rebalancing. Time line profile for all the PEs (rows) on a SMP process for the 16*K* cores run. White shows idle time and colored bars indicate busy time. Dynamic rebalancing eliminates trailing idle time resulting in better utilization. The total time per step reduces from 9.8 seconds to 8.5 seconds.

time. This plot is for a 32*K* core run on Blue Waters, and we have chosen the PE and the corresponding SMP process with the maximum load. We can see that the most loaded PE, which also contains the most loaded tree piece, is busy for about 2 seconds while other PEs are idle. Figure 9(b) shows the time line for the work pushing strategy for a set of PEs in the SMP process where one of the PEs is assigned the most loaded tree piece. With the work pushing strategy, we are able to successfully distribute the work load among other PEs within the node. This results in a reduction of the gravity time from 2.3 seconds to 0.3 seconds for the fastest rung.

8.3 Intra-node dynamic rebalancing

For clustered datasets, it is often the case at the trailing end of the gravity calculation that some of the PEs are idle while others are busy. This could be due to misprediction of load or inability of the load balancer to balance the load perfectly. Figure 10(a) shows the Projections time-line view for this scenario where the colored bars indicate busy work while the white shows idle time. We found that such slight load imbalances in the application can be mitigated by more fine-grained parallelism within the SMP process. We use an intra-node dynamic rebalancing scheme where the idle PEs within the node pick work from the busy ones. The scheme is implemented using the *CkLoop* library (Mei et al. 2010) in Charm++, which enables fine-grained parallelism within an SMP process.

As with the work-pushing scheme, buckets are the smallest entity of work that can be reassigned.

If all the tree pieces residing on a PE have finished their work, then the PE becomes idle. At each PE, the *LBManager* maintains a PE-private variable which keeps track of its status. Since the memory address is shared among the PEs on a SMP process, the *LBManager* can access the status variable of all the PEs within the SMP process. Whenever there is a significant number of idle PEs, the dynamic rebalancing scheme kicks in. Tree pieces then create chunks of work out of the unfinished buckets and add these to the node-level queue. The idle processors access the node-level queue and pick up work to execute. Due to the overhead associated with the node-level queue we only use the work-stealing scheme adaptively for the trailing end of the computation.

Figure 10(a) shows the time-line for the slowest rung, rung 0, of *cosmo25* dataset simulation for a 32*K* core run on Blue Waters. We pick a subset of PEs to show this problem. We can see that the load is almost balanced, but towards the end of the step there are some PEs which are idle while others are busy. Figure 10(b) shows the time-line with dynamic rebalancing. It is able to successfully handle small amounts of load imbalance and reduce the gravity time from 9.8 seconds to 8.5 seconds for rung 0.

8.4 SMP request cache

Data reuse can be critical in determining the performance of tree-based algorithms (Gioachin et al. 2007). Modern SMP-based supercomputers offer several levels at which data sharing can be effective. One possibility is that requests for the same remote elements from two traversals on a core can be merged. The fetched data can then be reused by all traversals on the core. Similarly, cores in the same SMP domain can share remotely fetched data. In the following we describe a two-level caching scheme that enables the data reuse across traversals on a core, as well as across cores on an SMP processor. This caching mechanism is transparent to the traversal code.

Each core on the SMP has a *private* cache, which stores pointers to remotely fetched data. There also exists one cache at the level of the SMP that is *shared* by all cores in the SMP. The shared cache contains the union of all the entries in the private caches of these PEs.

Briefly, the algorithm funnels all requests for remote data through the cache. If the data are found in the private cache, then they are immediately passed into the requesting traversal's visitor code. If the data are not found on the PE, we check whether some other piece on the PE has requested them previously. If so, a lightweight continuation

is created to resume the traversal at the requested node upon its receipt. Otherwise, the more expensive, SMP-wide table lookup is performed.

We devised a scheme to manage concurrent accesses of the shared, SMP-wide cache table, where all requests for remote data generated by traversals on the SMP processor are funneled through a single core, which is termed the *fetcher* for that SMP processor. Cheap, intra-node messaging between PEs is used for efficiency.

8.5 Domain decomposition

Simulations of datasets with nonuniform distributions are characterized by extensive movement of particles across tree piece boundaries over time. When unchecked, this leads to an increasingly nonuniform distribution of particles across tree pieces and eventually precludes a good balance of load across processors. In such scenarios, it becomes useful to repeat the full domain decomposition more frequently.

The first stage of domain decomposition, as described in Section 4, involves a series of histogramming steps to determine a set of splitters that partition the simulation domain into tree pieces of roughly uniform particle count. This is implemented in terms of broadcasts from a single sorter object, which refines the splitters, to the tree pieces, and reductions of particle counts for each bin back to the sorter process. In strong scaling scenarios for highly clustered datasets, domain decomposition may become a performance bottleneck, as the number of splitters generally depends on the number of processors used in the run. Therefore, we implemented a number of optimizations aimed at improving the SFC domain decomposition performance. First, we replaced the broadcast of SFC keys from the sorter object with the broadcast of a bit vector indicating which of the bins evaluated in the previous step need further refinement. From the bit vector, the set of splitters to evaluate is determined once at each SMP node, and delivered to all tree pieces at that node for evaluation. This optimization greatly reduced the size of the buffers being broadcast. Secondly, we noticed that some histogramming steps were much more expensive than others, due to involving more splitters. This was particularly true for the first and last steps. The first histogramming step involved a full set of splitters due to none having been finalized yet. For this step, we were able to remove the broadcast of splitters by having tree pieces reuse the splitters determined the last time domain decomposition was done. We were also able to eliminate the last histogramming step in the original algorithm, in which the final set of splitters was broadcast to the tree pieces to collect a full histogram of particle counts. Instead, we modified the sorter object to preserve particle counts for all previously finalized splitters, so as to have the full set of counts at the end.

These optimizations significantly improved domain decomposition performance. For runs of the *cosmo25* dataset on Blue Waters, the time for a full domain decomposition was reduced from 3.22 s to 1.52 s on 1,024 nodes, a speedup of 2.1.

8.6 Hierarchical multistep load balancer

Even if domain decomposition assigns almost equal number of particles to tree pieces, density variations in different regions of the simulated space can result in load imbalance. We experimented with domain decomposition based on load, but the basic approach was not ideal for multi-stepping simulations as it led to large jumps in boundaries and significant movement of particles. Since execution time is determined by the most loaded processor, it becomes important to address the load imbalance problem without significant additional overhead.

Load balancing in Charm++ applications like ChaNGa is normally achieved by over-decomposing the problem into many more objects than processors and letting the Charm++ dynamic load balancing framework balance the load by mapping the objects to processors (Zheng 2005). The framework can automatically instrument the computation load and communication pattern of tree pieces and other objects and store it in a distributed database. This information is then used by the load balancing strategies, which we optimized for ChaNGa, to map the objects to processors. Once the decision has been made, the load balancing framework migrates the objects to newly assigned processors. Alternatively, the load of the objects and their communication pattern can be determined using a model based on *a priori* knowledge. But for ChaNGa, we find that determining the load based on a heuristic called the *principle of persistence* is more accurate. Based on this heuristic we use recent history to determine the load of near-future iterations. This scheme works well for single-stepping simulations at a relatively small scale. However, multi-stepping simulations at very large scale impose several new challenges.

First, multi-stepped execution introduces some challenges in the measurement based load balancing to obtain accurate load information. Substeps within a big step in a multi-step run have selected number of active particles. Predicting the load of a tree piece based on the preceding substep will result in discrepancy between the expected load and the actual load. Therefore, we instrument and store the load of the tree pieces for different substeps/rungs separately. Whenever particles migrate from one tree piece to another, they carry a fraction of their load for the corresponding rungs for which they were active and contribute that to the new tree piece. This enables us to achieve very accurate prediction of the load of a tree piece for each substep even with migrations and multi-stepping.

Secondly, it is very challenging to collect communication pattern information in ChaNGa, even at small core

count, due to a very large number of messages in the simulation, which may incur significant overhead on memory when performing load balancing. Therefore, we used an alternate strategy to implicitly take communication into account during load balancing by using an ORB-based (Orthogonal Recursive Bipartitioning) strategy, which preserves the communication locality.

Lastly, in extremely large scale simulations, load balancing itself becomes a severe bottleneck. The original centralized load balancing strategies, where load balancing decision is made on one central processor, do not scale beyond a few hundred processors, which makes them unfeasible for large scale simulations. To overcome this challenge, we implemented a scalable load balancing strategy suitable for multi-stepped execution based on the hierarchical load balancing framework (Zheng 2005; Mei et al. 2011) in the Charm++ runtime. This new load balancing strategy performs ORB to distribute the tree pieces among processors. The processors are divided into independent groups organized in a hierarchical fashion. Each group consists of 512 processors. At each level of the hierarchy, the root performs the load balancing strategy for the processors in its sub-tree. We found that two levels of the hierarchy is enough to achieve good load balance with little overhead. At higher levels of the hierarchy refinement based load balancing strategy, which minimizes the migration by considering the current assignment of tasks, is used. At the lowest level of the hierarchy we use ORB to partition the tree pieces among the processors in that subgroup. The load balancer collects the centroid information of tree pieces along with their load. Taking the centroids into account, the tree pieces are spatially partitioned into two sets along the longest dimension. Similarly, at each stage of partitioning, the processors are also partitioned. During partitioning, tree pieces are divided into two partitions such that the loads of the partitions are almost equal. This is done recursively until one processor remains which is assigned the corresponding partition containing the tree pieces.

Another optimization to further reduce the overhead of load balancing is to combine the node level global load balancing with the intra-node load balancing strategies described in Section 8.1. We implemented such a two-level load balancing strategy, where the load is first balanced across SMP nodes, and then balanced inside each SMP node. The ORB algorithm described above is done for nodes rather than processors. Once the tree pieces are assigned to SMP nodes, they are further distributed among the PEs in the SMP node using a *greedy* strategy. This ensures that the load is equally distributed among the SMP nodes. We perform an additional step of refinement to further improve the load balance for the rare cases when the load is not evenly balanced.

8.7 Performance evaluation

We now present the scaling performance of the *cosmo25* simulation. Figure 11(a) shows the average time per iteration for this simulation with single-stepping and Figure 11(b) shows the average time per iteration with multi-stepping. In a multi-stepping run, 16 substeps constitute a big step. To compare the time for single-stepping and multi-stepping, a single big multi-step covers the same dynamical time as 16 single steps. Table 3 gives a break down of the time taken for different phases for single-stepping and multi-stepping. We can see that at $8K$ cores the single-stepping simulation takes more than 3 times the time taken by multi-stepping and at $128K$ it takes twice as long. Note that the gravity time for multi-stepping is 4.5 times faster than single stepping. Due to sufficient sequential work to overlap communication and relatively balanced tree pieces, we are able to achieve 80% efficiency for

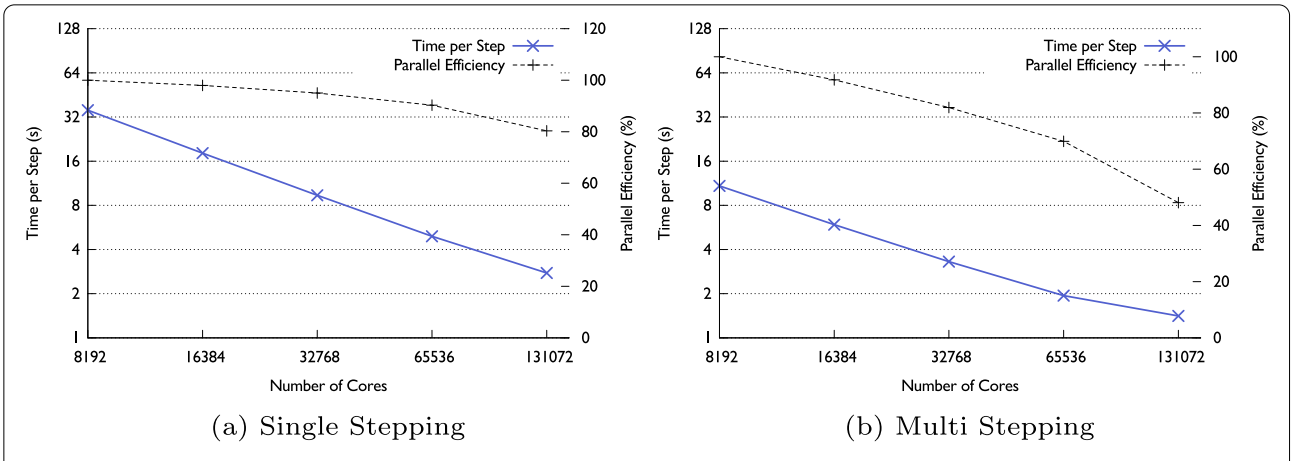


Figure 11 Performance comparison of single stepping and multi stepping. Time per step and parallel efficiency for the *cosmo25* dataset on Blue Waters.

Table 3 Breakdown of time for 1 step in seconds for *cosmo25* dataset with single-stepping (top half) and multi-stepping (bottom half) on Blue Waters

#cores	Gravity	DD	TB	LB	Step time	16 Step time
8,192	33.433	0.441	0.292	1.423	35.589	569.424
16,384	16.952	0.210	0.148	0.851	18.161	290.576
32,768	8.643	0.132	0.091	0.496	9.362	149.792
65,536	4.395	0.163	0.073	0.295	4.926	78.816
131,072	2.353	0.134	0.066	0.216	2.769	44.304
8,192	7.45	0.83	0.47	2.1	10.85	173.6
16,384	3.73	0.79	0.32	1.07	5.91	94.56
32,768	2.1	0.46	0.2	0.55	3.31	52.96
65,536	1.1	0.35	0.12	0.37	1.94	31.04
131,072	0.77	0.24	0.07	0.33	1.41	22.56

Table 4 Breakdown of time for 1 step in seconds for *cosmo25* dataset with PKDGRAV2 on Blue Waters

#cores	Gravity	DD	TB	Step time
8,192	17.90	1.50	0.57	19.97
16,384	10.10	1.40	0.84	12.34
32,768	6.10	0.97	1.50	8.57
49,152	6.60	0.99	13.30	20.89
65,536	8.60	1.00	17.80	27.40
98,304	16.10	1.30	25.80	43.20

single-stepping at 128K cores with an average step time of 2.7 seconds. As described in Section 8.1 the multi-stepping run has many challenges due to irregular distribution of particles in faster rungs. Incorporating the improvements mentioned above, we are able to scale to 128K cores with an efficiency of 48% with respect to 8K cores with a time step of 1.4 seconds. Note that if we consider the gravity force calculation time, we achieve an efficiency of 60% and the gravity time is 3 times faster in multi-stepping in comparison to the single-stepping run.

8.8 Comparison with PKDGRAV

To give a sense of the absolute performance of ChaNGa compared to other available N-body codes, we ran the *cosmo25* dataset with PKDGRAV2.[c] Table 4 gives the step time for PKDGRAV2 for the *cosmo25* dataset on Blue Waters. Comparing Table 4, the timings for PKDGRAV2, and Table 3, the timings for ChaNGa, for the single stepping benchmark, PKDGRAV2 is faster than ChaNGa on up to 32K cores but ChaNGa continues to scale until 131K cores to a time per step of 2.7 s. The Multi-stepping run of ChaNGa performs consistently better than PKDGRAV2.

9 Conclusion

In this paper, we have described the design and features of our highly scalable parallel gravity code ChaNGa and went into the details of scaling challenges for clustered multiple time-stepping datasets. We have presented strong scaling results for uniform datasets on up to 512K cores on Blue Waters evolving 12 and 24 billion particles. We also present strong scaling results for *cosmo25* and *dwarf* datasets, which are more challenging due to their highly clustered nature. We obtain good performance on up to 128K cores of Blue Waters and also show up to a 3 fold improvement in time with multi-stepping over single-stepping.

Many features of the Charm++ runtime system were used to achieve these results. Starting with the standard load balancing and overlap of communication and computation enabled by the over-decomposition strategy, we employed a number of Charm++'s features. Of particular importance were features that allowed us to replace parts of our algorithm that scaled as the number of cores, such as quiescence detection for particle movement and the hierarchical load balancer. Also of importance were features such as CkLoop, SMP Cache and node level load balancing, that exploited SMP features of almost all modern supercomputers. With these features, we can bring to bear the computational resources of many 100s of thousands of processor cores on the highly clustered, large dynamic range simulations that are necessary for understanding the formation of galaxies in the context of large scale structure.

While the focus of the work presented here was the performance of the gravity calculation, these techniques are applicable to other parts of cosmological simulations. Above, we summarized the SPH implementation in ChaNGa. To give an indication of the performance of our implementation, we used the *cosmo25* dataset which actually has 23 percent of the particles labeled as 'gas'. Benchmarking this dataset with an adiabatic equation of state for the gas on 8K cores, we find that the SPH component of the force calculation alone takes on average 37.9 seconds compared to an average 39.6 seconds needed for the gravity calculation. However, as mentioned above, the SPH calculation is dominated by the communication, and when we overlap the SPH with the gravity calculation it adds only 15.2 seconds over a gravity calculation alone. While this result nicely demonstrates the ability of the Charm++ runtime system to overlap communication and computation, it also indicates that there may be room for optimization of the neighbor finding algorithm. Neighbor finding is also a useful algorithm for implementing other hydrodynamic techniques. We expect that the recently developed Meshless Finite Mass and Meshless Finite Volume methods (Hopkins 2014) will scale better than SPH since they require fewer neighbors and the inter-neighbor calculations require more computation. Moving mesh methods (Springel 2010) can require the construction of a Voronoi mesh which, in turn, requires algorithms to quickly find all particles within a sphere of given radius. Again, the neighbor finding algorithm used in ChaNGa can perform this task. The implementation of some of these algorithms will be the subject of future work.

Future work is also planned to improve the scaling on hybrid architectures like those of many current leadership class machines. We have had some success in getting good performance and scaling on up to 896 cores and 256 GPUs with earlier generation GPUs (Jetley et al. 2010). The Charm++ paradigm for overlapping computing and computation also works for the overlap of data transfer from the host to the GPU, GPU gravity kernel work, and tree walk work done on the host CPUs. However, we have not addressed the balance of work, either between GPU and host CPU or among GPUs. Also, the increased performance of individual nodes enabled by GPUs or other accelerators will increase the need to optimize and hide communication costs.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
TQ is the primary researcher and supervisor of the ChaNGa project. TQ, PJ, along with various contributors developed the code. TQ and FG, and others verified the code for cosmological simulations. HM, LW, TQ and LK came up with the techniques mentioned in the paper for scaling the application. H.M. developed the dynamic load balancing techniques and optimizations for the various phases of the simulation. GZ and HM developed the hierarchical load balancer. LW worked on the domain decomposition optimizations. PJ developed the SMP cache optimization. GZ optimized the performance for the Blue Waters hardware. HM performed the scaling experiments with help from TQ, GZ and LW. All the authors discussed the results and contributed extensively to the writing of the paper.

Author details
[1]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, USA. [2]Department of Astronomy, University of Washington, Seattle, USA.

Acknowledgements
ChaNGa was initially developed under NSF ITR award 0205413. Contributors to the development of the code include Graeme Lufkin, Sayantan Chakravorty, Amit Sharma, and Filippo Gioachin. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois. HM was supported by NSF award AST-1312913. TQ and FG where supported by NSF award AST-1311956. Use of Bluewaters was supported by NSF PRAC Award 1144357. We made use of pynbody (https://github.com/pynbody/pynbody) to create Figure 1, and we thank Andrew Pontzen for assistance in creating that figure. We also thank the referees for helpful comments that improved the manuscript.

Endnotes
[a] A public version of ChaNGa is publicly available at http://hpcc.astro.washington.edu/tools/changa.html.
[b] Using a geometric density mean in the SPH force expression: $(P_i + P_j)/(\rho_i \rho_j)$ in place of $P_i/\rho_i^2 + P_j/\rho_j^2$ where P_i and ρ_i are particle pressures and densities respectively.
[c] We downloaded version 2.2.15.3.1 in June 2014 from https://hpcforge.org/projects/pkdgrav2/.

Received: 3 September 2014 Accepted: 17 February 2015
Published online: 28 March 2015

References
Planck Collaboration: Ade, PAR, Aghanim, N, Armitage-Caplan, C, Arnaud, M, Ashdown, M, Atrio-Barandela, F, Aumont, J, Baccigalupi, C, Banday, AJ, et al.: Planck 2013 results. XVI. Cosmological parameters (2013). arXiv:1303.5076
Angulo, RE, Springel, V, White, SDM, Jenkins, A, Baugh, CM, Frenk, CS: Scaling relations for galaxy clusters in the millennium-XXL simulation. Mon. Not. R. Astron. Soc. **426**, 2046-2062 (2012). doi:10.1111/j.1365-2966.2012.21830.x
Barnes, J, Hut, P: A hierarchical O(NlogN) force-calculation algorithm. Nature **324**, 446-449 (1986)
Brooks, A: Re-examining astrophysical constraints on the Dark Matter model (2014). arXiv:1407.7544
Brooks, AM, Governato, F, Booth, CM, Willman, B, Gardner, JP, Wadsley, J, Stinson, G, Quinn, T: The origin and evolution of the mass-metallicity relationship for galaxies: results from cosmological N-body simulations. Astrophys. J. Lett. **655**, 17-20 (2007). doi:10.1086/511765
Christensen, C, Quinn, T, Governato, F, Stilp, A, Shen, S, Wadsley, J: Implementing molecular hydrogen in hydrodynamic simulations of galaxy formation. Mon. Not. R. Astron. Soc. **425**, 3058-3076 (2012). doi:10.1111/j.1365-2966.2012.21628.x
Collins, DC, Xu, H, Norman, ML, Li, H, Li, S: Cosmological adaptive mesh refinement magnetohydrodynamics with Enzo. Astrophys. J. Suppl. Ser. **186**, 308-333 (2010). doi:10.1088/0067-0049/186/2/308
Davis, M, Efstathiou, G, Frenk, CS, White, SDM: The evolution of large-scale structure in a universe dominated by cold dark matter. Astrophys. J. **292**, 371-394 (1985). doi:10.1086/163168
Dehnen, W: Towards optimal softening in three-dimensional N-body codes - I. Minimizing the force error. Mon. Not. R. Astron. Soc. **324**, 273-291 (2001). doi:10.1046/j.1365-8711.2001.04237.x
Ding, H-Q, Karasawa, N, Goddard, WA III: The reduced cell multipole method for Coulomb interactions in periodic systems with million-atom unit cells. Chem. Phys. Lett. **196**, 6-10 (1992). doi:10.1016/0009-2614(92)85920-6
Durier, F, Dalla Vecchia, C: Implementation of feedback in smoothed particle hydrodynamics: towards concordance of methods. Mon. Not. R. Astron. Soc. **419**, 465-478 (2012). doi:10.1111/j.1365-2966.2011.19712.x
Ewald, PP: Die Berechnung optischer und elektrostatischer Gitterpotentiale. Ann. Phys. **369**, 253-287 (1921). doi:10.1002/andp.19213690304
Frenk, CS, White, SDM: Dark matter and cosmic structure. Ann. Phys. **524**, 507-534 (2012). doi:10.1002/andp.201200212
Gioachin, F, Sharma, A, Chakravorty, S, Mendes, C, Kale, LV, Quinn, TR: Scalable cosmology simulations on parallel machines. In: VECPAR 2006. LNCS, vol. 4395, pp. 476-489 (2007)
Governato, F, Willman, B, Mayer, L, Brooks, A, Stinson, G, Valenzuela, O, Wadsley, J, Quinn, T: Forming disc galaxies in ΛCDM simulations. Mon. Not. R. Astron. Soc. **374**, 1479-1494 (2007). doi:10.1111/j.1365-2966.2006.11266.x
Governato, F, Zolotov, A, Pontzen, A, Christensen, C, Oh, SH, Brooks, AM, Quinn, T, Shen, S, Wadsley, J: Cuspy no more: how outflows affect the central dark matter and baryon distribution in Λ cold dark matter galaxies. Mon. Not. R. Astron. Soc. **422**, 1231-1240 (2012). doi:10.1111/j.1365-2966.2012.20696.x
Governato, F, Weisz, D, Pontzen, A, Loebman, S, Reed, D, Brooks, AM, Behroozi, P, Christensen, C, Madau, P, Mayer, L, Shen, S, Walker, M, Quinn, T, Wadsley, J: Faint dwarfs as a test of DM models: WDM vs. CDM (2014). arXiv:1407.0022
Guedes, J, Callegari, S, Madau, P, Mayer, L: Forming realistic late-type spirals in a ΛCDM universe: the Eris simulation. Astrophys. J. **742**, 76 (2011). doi:10.1088/0004-637X/742/2/76
Habib, S, Morozov, V, Frontiere, N, Finkel, H, Pope, A, Heitmann, K: Hacc: extreme scaling and performance across diverse architectures. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. SC '13, pp. 6-1610. ACM, New York (2013). doi:10.1145/2503210.2504566
Hopkins, PF: GIZMO: a new class of accurate. mesh-free hydrodynamic simulation methods (2014). arXiv:1409.7395
Hopkins, PF, Keres, D, Onorbe, J, Faucher-Giguere, C-A, Quataert, E, Murray, N, Bullock, JS: Galaxies on FIRE (Feedback In Realistic Environments): stellar feedback explains cosmologically inefficient star formation (2013). arXiv:1311.2073
Ishiyama, T, Nitadori, K, Makino, J: 4.45 Pflops astrophysical N-body simulation on K computer - the gravitational trillion-body problem (2012). arXiv:1211.4406
Jetley, P, Gioachin, F, Mendes, C, Kale, LV, Quinn, TR: Massively parallel cosmological simulations with ChaNGa. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008 (2008)
Jetley, P, Wesolowski, L, Gioachin, F, Kalé, LV, Quinn, TR: Scaling hierarchical N-body simulations on GPU clusters. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing,

Networking, Storage and Analysis. SC '10. IEEE Computer Society, Washington (2010)
Kalé, LV, Sinha, A: Projections: a scalable performance tool. In: Parallel Systems Fair, International Parallel Processing Symposium, pp. 108-114 (1993)
Keller, BW, Wadsley, J, Benincasa, SM, Couchman, HMP: A superbubble feedback model for galaxy simulations. Mon. Not. R. Astron. Soc. **442**, 3013-3025 (2014). doi:10.1093/mnras/stu1058
Kim, J-H, Abel, T, Agertz, O, Bryan, GL, Ceverino, D, Christensen, C, Conroy, C, Dekel, A, Gnedin, NY, Goldbaum, NJ, Guedes, J, Hahn, O, Hobbs, A, Hopkins, PF, Hummels, CB, Iannuzzi, F, Keres, D, Klypin, A, Kravtsov, AV, Krumholz, MR, Kuhlen, M, Leitner, SN, Madau, P, Mayer, L, Moody, CE, Nagamine, K, Norman, ML, Onorbe, J, O'Shea, BW, Pillepich, A, Primack, JR, Quinn, T, Read, JI, Robertson, BE, Rocha, M, Rudd, DH, Shen, S, Smith, BD, Szalay, AS, Teyssier, R, Thompson, R, Todoroki, K, Turk, MJ, Wadsley, JW, Wise, JH, Zolotov, A (for the AGORA Collaboration29): The AGORA high-resolution galaxy simulations comparison project. Astrophys. J. Suppl. Ser. **210**, 14 (2014). doi:10.1088/0067-0049/210/1/14
Mei, C, Zheng, G, Gioachin, F, Kalé, LV: Optimizing a parallel runtime system for multicore clusters: a case study. In: TeraGrid'10, Pittsburgh, PA, USA (2010)
Mei, C, Sun, Y, Zheng, G, Bohm, EJ, Kalé, LV, Phillips, JC, Harrison, C: Enabling and scaling biomolecular simulations of 100 million atoms on petascale machines with a multicore-optimized message-driven runtime. In: Proceedings of the 2011 ACM/IEEE Conference on Supercomputing, Seattle, WA (2011)
Menon, H, Jain, N, Zheng, G, Kalé, LV: Automated load balancing invocation based on application characteristics. In: IEEE Cluster, Beijing, China, vol. 12 (2012)
Munshi, F, Governato, F, Brooks, AM, Christensen, C, Shen, S, Loebman, S, Moster, B, Quinn, T, Wadsley, J: Reproducing the stellar mass/halo mass relation in simulated LCDM galaxies: theory vs observational estimates. Astrophys. J. **766**, 56 (2013). doi:10.1088/0004-637X/766/1/56
Pontzen, A, Governato, F: How supernova feedback turns dark matter cusps into cores. Mon. Not. R. Astron. Soc. **421**, 3464-3471 (2012). doi:10.1111/j.1365-2966.2012.20571.x
Pontzen, A, Governato, F: Cold dark matter heats up. Nature **506**, 171-178 (2014). doi:10.1038/nature12953
Pontzen, A, Roškar, R, Stinson, GS, Woods, R: pynbody: N-Body/SPH analysis for Python. Astrophysics Source Code Library (2013). ascl:1305.002
Power, C, Navarro, JF, Jenkins, A, Frenk, CS, White, SDM, Springel, V, Stadel, J, Quinn, T: The inner structure of ΛCDM haloes - I. A numerical convergence study. Mon. Not. R. Astron. Soc. **338**, 14-34 (2003). doi:10.1046/j.1365-8711.2003.05925.x
Purcell, CW, Bullock, JS, Tollerud, EJ, Rocha, M, Chakrabarti, S: The Sagittarius impact as an architect of spirality and outer rings in the Milky Way. Nature **477**, 301-303 (2011). doi:10.1038/nature10417
Quinn, T, Katz, N, Stadel, J, Lake, G: Time stepping N-body simulations (1997). arXiv:astro-ph/9710043
Quinn, TR, Jetley, P, Kale, LV, Gioachin, F: N-body simulations with ChaNGa. In: Kale, LV, Bhatele, A (eds.) Parallel Science and Engineering Applications: The Charm++ Approach. CRC Press, Boca Raton (2013)
Reed, D, Gardner, J, Quinn, T, Stadel, J, Fardal, M, Lake, G, Governato, F: Evolution of the mass function of dark matter haloes. Mon. Not. R. Astron. Soc. **346**, 565-572 (2003). doi:10.1046/j.1365-2966.2003.07113.x
Ruan, JJ, Quinn, TR, Babul, A: The observable thermal and kinetic Sunyaev-Zel'dovich effect in merging galaxy clusters. Mon. Not. R. Astron. Soc. **432**, 3508-3519 (2013). doi:10.1093/mnras/stt701
Schaye, J, Crain, RA, Bower, RG, Furlong, M, Schaller, M, Theuns, T, Dalla Vecchia, C, Frenk, CS, McCarthy, IG, Helly, JC, Jenkins, A, Rosas-Guevara, YM, White, SDM, Baes, M, Booth, CM, Camps, P, Navarro, JF, Qu, Y, Rahmati, A, Sawala, T, Thomas, PA, Trayford, J: The EAGLE project: simulating the evolution and assembly of galaxies and their environments (2014). arXiv:1407.7040
Sharma, A: Performance evaluation of tree structures and tree traversals for parallel n-body cosmological simulations. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (2006) http://charm.cs.uiuc.edu/papers/AmitMSThesis.html
Shen, S, Wadsley, J, Stinson, G: The enrichment of the intergalactic medium with adiabatic feedback - I. Metal cooling and metal diffusion. Mon. Not. R. Astron. Soc. **407**, 1581-1596 (2010). doi:10.1111/j.1365-2966.2010.17047.x
Sinha, AB, Kale, LV, Ramkumar, B: A dynamic and adaptive quiescence detection algorithm. Technical Report 93-11, Parallel Programming Laboratory, Department of Computer Science, University of Illinois, Urbana-Champaign (1993)
Solomonik, E, Kale, LV: Highly scalable parallel sorting. In: Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2010)
Springel, V: The cosmological simulation code GADGET-2. Mon. Not. R. Astron. Soc. **364**, 1105-1134 (2005). doi:10.1111/j.1365-2966.2005.09655.x
Springel, V: E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. Mon. Not. R. Astron. Soc. **401**, 791-851 (2010). doi:10.1111/j.1365-2966.2009.15715.x
Springel, V, White, SDM, Jenkins, A, Frenk, CS, Yoshida, N, Gao, L, Navarro, J, Thacker, R, Croton, D, Helly, J, Peacock, JA, Cole, S, Thomas, P, Couchman, H, Evrard, A, Colberg, J, Pearce, F: Simulations of the formation, evolution and clustering of galaxies and quasars. Nature **435**, 629-636 (2005). doi:10.1038/nature03597
Stadel, JG: Cosmological N-body simulations and their analysis. PhD thesis, Department of Astronomy, University of Washington (March 2001)
Stinson, G, Seth, A, Katz, N, Wadsley, J, Governato, F, Quinn, T: Star formation and feedback in smoothed particle hydrodynamic simulations - I. Isolated galaxies. Mon. Not. R. Astron. Soc. **373**, 1074-1090 (2006). doi:10.1111/j.1365-2966.2006.11097.x
Vogelsberger, M, Sijacki, D, Kereš, D, Springel, V, Hernquist, L: Moving mesh cosmology: numerical techniques and global statistics. Mon. Not. R. Astron. Soc. **425**, 3024-3057 (2012). doi:10.1111/j.1365-2966.2012.21590.x
Vogelsberger, M, Genel, S, Springel, V, Torrey, P, Sijacki, D, Xu, D, Snyder, GF, Nelson, D, Hernquist, L: Introducing the illustris project: simulating the coevolution of dark and visible matter in the Universe (2014). arXiv:1405.2921
Wadsley, JW, Stadel, J, Quinn, T: Gasoline: a flexible, parallel implementation of TreeSPH. New Astron. **9**, 137-158 (2004)
Wadsley, JW, Veeravalli, G, Couchman, HMP: On the treatment of entropy mixing in numerical cosmology. Mon. Not. R. Astron. Soc. **387**, 427-438 (2008). doi:10.1111/j.1365-2966.2008.13260.x
Warren, MS: 2HOT: an improved parallel hashed oct-tree N-body algorithm for cosmological simulation (2013). arXiv:1310.4502
Zheng, G: Achieving high performance on extremely large parallel machines: performance prediction and load balancing. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (2005)