Massively Parallel Cosmological Simulations with ChaNGa

Pritish Jetley, Filippo Gioachin, Celso Mendes, Laxmikant V. Kalé,
Dep. of Computer Science, University of Illinois at Urbana-Champaign, USA
{pjetley2, gioachin, cmendes, kale}@uiuc.edu

Thomas Quinn
Dep. of Astronomy, University of Washington, USA
trq@astro.washington.edu

Abstract

Cosmological simulators are an important component in the study of the formation of galaxies and large scale structures, and can help answer many important questions about the universe. Despite their utility, existing parallel simulators do not scale effectively on modern machines containing thousands of processors. In this paper we present ChaNGa, a recently released production simulator based on the Charm++ *infrastructure. To achieve scalable performance, ChaNGa employs various optimizations that maximize the overlap between computation and communication. We present experimental results of ChaNGa simulations on machines with thousands of processors, including the IBM Blue Gene/L and the Cray XT3. The paper goes on to highlight efforts toward even more efficient and scalable cosmological simulations. In particular, novel load balancing schemes that base decisions on certain characteristics of tree-based particle codes are discussed. Further, the multistepping capabilities of ChaNGa are presented, as are solutions to the load imbalance that such* multiphase *simulations face. We outline key requirements for an effective practical implementation and conclude by discussing preliminary results from simulations run with our multiphase load balancer.*

1 Introduction

The scientific case for performing cosmological simulations is compelling. Theories of structure formation built upon the nature of Dark Matter and Dark Energy need to be constrained with comparisons to the observed structure and distribution of galaxies. However, galaxies are extremely non-linear objects, hence the connection between the theory and the observations requires following the non-linear dynamics using numerical simulations. Thus, cosmological simulators are important components for studying the formation of galaxies and planets.

The simulation of galaxy formation is a challenging computational problem, requiring high resolutions and dynamic timescales. As an example, to form a stable Milky Way-like galaxy, tens of millions of resolution elements must be simulated to the current epoch [5]. Locally adaptive timesteps may reduce the CPU work by orders of magnitude, but not evenly throughout the computational volume, thus posing a considerable challenge for parallel load balancing.

To address these issues, various cosmological simulators have been created recently. PKDGRAV [3], developed at the University of Washington, can be considered among the state-of-the-art in that area. However, due to load imbalance, PKDGRAV does not scale efficiently on the newer machines with thousands of processors. In this work, we present a new N-body cosmological simulator that (like other simulators) utilizes the Barnes-Hut algorithm [1] to compute gravitational forces. Our recently released simulator, named ChaNGa, is based on the Charm++ infrastructure [9]. We leverage the object-based virtualization [8] and the data-driven style of computation, inherent in Charm++, to obtain adaptive overlap of communication and computation, as well as to perform auto-

matic measurement-based load balancing. ChaNGa advances the state-of-the-art in N-Body simulations by allowing the programmer to achieve higher levels of resource utilization on large systems.

The remainder of this paper is organized as follows. Section 2 presents an overview of previous work in the development of parallel simulators for cosmology. Section 3 describes the major components of ChaNGa. Section 4 presents scalability results obtained with ChaNGa on various parallel platforms, and Section 5 illustrates some of the more advanced features in ChaNGa. Finally, Section 6 has our conclusions and future work directions.

2 Related Work

The study of the evolution of interacting particles under the effects of Newtonian gravitational forces, also known as the N-Body problem, has been extensively reported in the literature. A popular method to simulate such problems was proposed by Barnes and Hut [1]. That method associates particles to a hierarchical structure comprising a tree. By properly traversing the tree, one can compute the forces between a pair of particles or between a particle and a whole branch of the tree that is sufficiently far away. By using such an approximation, this method reduces the complexity of the problem from $O(N^2)$ to $O(N \log N)$, where N is the number of particles. $O(N)$ methods for calculating the gravitational force have also been developed, e.g. the Fast Multipole Method [6, 2]. However, the highly clustered nature of typical astrophysical datasets requires the organization of the data into some hierarchical structure. This is essentially a "sort", thus the overall complexity is still $O(N \log N)$.

Hierarchical methods for N-Body simulations have been adopted for quite some time by astronomers [12, 18]. A pioneering parallel tree code is the HOT code [19], which won the Gordon Bell prize in 1992 and 1997. This code does not have locally adaptive timesteps. Another widely used code in this area is PKDGRAV [3], a tree-based simulator that works on both shared-memory and distributed-memory parallel systems. However, despite its success in the past, PKDGRAV has shown limited potential to scale to larger machine configurations, due to load balancing constraints. This limitation makes PKDGRAV's effective use on future petascale systems very questionable.

Other recent cosmological simulators are GADGET [17], developed at the Max-Planck-Institut für Astrophysik, Garching and falcON [2], developed at the Max-Planck-Institut für Astronomie, Heidelberg. falcON has shown good scalability with the number of particles, but it is a serial simulator, and does not have locally adaptive timesteps. Attempts have been made to parallelize the algorithm, but only to 16 processors [13]. GADGET, now in its second version (GADGET-2), has a rich modeling capability. The published scalability results have been limited to 128 processors [15], but it has been used on at least 512 processors with 10 billion particles for published scientific results [16]. Direct comparisons with GADGET will be in our future work.

In addition, some special purpose machines have been developed in the past to simulate cosmological N-body problems. Notable examples are Grape-5 [11], which has 32 pipeline processors specialized for the gravitational force calculation, and an FPGA-based system [10] constructed as an add-in card connected to a host PC. Both systems achieve great price/performance points, but have severe memory constraints: the largest datasets that one can use on them are limited to two million particles.

3 Structure of the ChaNGa Code

ChaNGa is implemented as a Charm++ application, leveraging all the advanced features existing in the Charm++ runtime system. In this section, after reviewing the basic Charm++ characteristics, we describe ChaNGa's internal organization and point to its major optimizing features.

3.1 Charm++ Infrastructure

Charm++ is a parallel software infrastructure that implements the concept of *processor virtualization* [9]. In this abstraction, an application programmer decomposes the underlying problem into a large number of objects and the interactions among those objects. The Charm++ runtime system automatically maps objects, also called *chares*, to physical processors. Typically, the number of chares is much greater than the number of processors. This virtualization technique makes the number of chares (and therefore the problem of decomposition) independent of the number of available processors. With this separation between logical and physical abstractions, Charm++ provides higher programmer productivity. This scheme has allowed the cre-

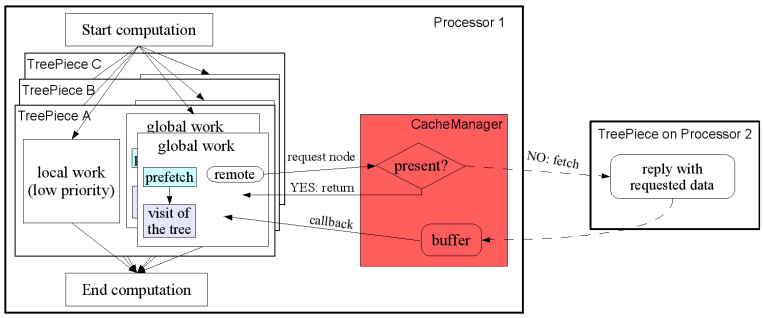


Figure 1. Components for parallel calculation of gravitational forces in ChaNGa

ation of parallel applications that scale efficiently to thousands of processors, such as the molecular dynamics NAMD code [14], a winner of a Gordon Bell award in 2002.

Besides mapping and scheduling chares on available processors, the CHARM++ runtime system can also dynamically migrate chares across processors. This capability is used to implement a powerful measurement-based load balancing mechanism in CHARM++ [20]. Chares can be migrated based on observed values of various metrics, such as computation loads or communication patterns. This dynamic redistribution is critical to achieve good overall processor utilization.

3.2 ChaNGa Organization

The major task in any cosmological simulator is to compute gravitational forces generated by the interaction between particles and integrating those forces over time to compute the movement of each particle. As we previously described in [4], ChaNGa accomplishes this task with an algorithm that uses a tree to represent the space. This tree is constructed globally over all particles in the dataset, and segmented into elements named *TreePieces*. These TreePieces are distributed by the CHARM++ runtime system to the available processors for parallel computation of the gravitational forces. ChaNGa allows various distribution schemes, including Morton and Peano-Hilbert Space-Filling-Curve (SFC) and Oct-tree-based (Oct) methods. Each TreePiece is implemented in ChaNGa as a CHARM++ chare.

At the lowest level of the tree, particles corresponding to tree leaves are grouped into *buckets* of a predefined maximal size, according to spatial proximity. To compute the forces on particles of a given bucket, one must traverse the tree to collect force contributions from all tree nodes. If a certain node is sufficiently far in space from that bucket, an aggregated contribution corresponding to all particles under that node is used; otherwise, the node is *opened* and recursively traversed.

The parallel implementation of gravity calculation in ChaNGa is represented in Figure 1. Each processor contains a group of TreePieces. To process a certain bucket, the processor needs to collect information from the entire tree. This might correspond to interactions with TreePieces in the same processor (*local* work) or with TreePieces from remote processors (*global* work). To optimize the access to identical remote information by buckets in the same processor, ChaNGa employs a particle caching mechanism. This cache is essential to achieve acceptable performance [4].

3.3 Major Optimizations in ChaNGa

Besides the particle caching mechanism mentioned in the previous subsection, ChaNGa implements various optimizations that contribute to its parallel scalability. Some of the major optimizations are the following (for more details, see [4]):

Data Prefetching: Given the availability of the particle caching mechanism, ChaNGa uses prefetching to further optimize the access to remote TreePieces. Before starting the gravity calculation for a certain bucket, a full tree traversal is done, to estimate the remote data that will be needed in the regular tree traversal. The prefetch of this remote data overlaps with local computation, thereby masking latency.

Tree-in-Cache: To avoid the overhead of communication between chares when accessing data from a different TreePiece in the same processor, ChaNGa employs another optimization that ex-

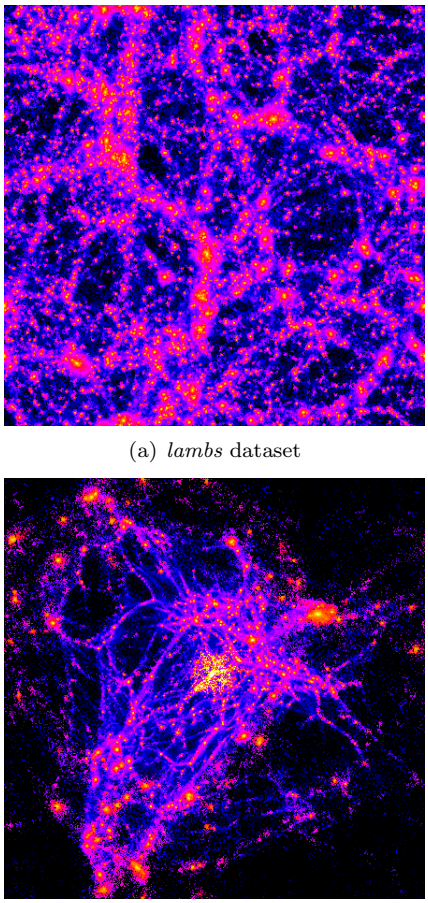
ploits the cache. Before computing forces, each TreePiece registers its data with the software cache. Thus, a larger tree, corresponding to the union of all local TreePieces, is assembled in the local cache. When any piece of that tree is needed during force computation, it is immediately retrieved.

Selectable Computation Granularity: ChaNGa accepts an input parameter that defines how much computation is performed before the processor is allowed to handle requests from remote processors. This enables a good tradeoff between responsiveness to communication requests and processor utilization.

4 Scalability Experiments

To evaluate ChaNGa's effectiveness as a production simulator, we conducted a series of tests with real cosmological datasets. These tests intended both to assess the code's portability across different systems and to measure its performance scalability in each particular type of system. We used the three systems described in Table 1, and ran tests with the following datasets:

lambs: Final state of a simulation of a $71Mpc^3$ volume of the Universe with 30% dark matter and 70% dark energy. Nearly three million particles are used. This dataset is highly clustered on scales less than 5 Mpc, but becomes uniform on scales approaching the total volume. Three subsets of this dataset are obtained by taking random subsamples of size thirty thousand, three hundred thousand, and one million particles, respectively.
dwarf: Snapshot at $z = .3$ of a multi-resolution simulation of a dwarf galaxy forming in a $28.5Mpc^3$ volume of the Universe with 30% dark matter and 70% dark energy. Although the *mass* distribution in this dataset is uniform on scales approaching the volume size, the *particle* distribution is very centrally concentrated and therefore highly clustered on all scales above the resolution limit. The total dataset size is nearly five million particles, but the central regions have a resolution equivalent to 2048^3 particles in the entire volume.
hrwh_lcdms: Final state of a $90Mpc^3$ volume of the Universe with 31% dark matter and 69% dark energy realized with 16 million particles. This dataset is used in [7], and is slightly more uniform than *lambs*.
dwarf-50M: Same physical model as *dwarf* except that it is realized with 50 million particles. The central regions have a resolution equivalent to 6144^3 particles in the entire volume.
lambb: Same physical model as *lambs* except that it is realized with 80 million particles.
drgas: Similar to *lambs* and *lambb* except that it is the high redshift ($z = 99$) state of the simulation, and it is realized with 730 million particles. The particle distribution is very uniform.



(a) *lambs* dataset

(b) *dwarf* dataset

Figure 2. Pictorial view of datasets

To illustrate some of the features in these datasets, Figure 2(a) presents a pictorial view of *lambs*, which has a reasonably uniform particle distribution, whereas Figure 2(b) presents *dwarf*, containing a much more clustered distribution. In these pictures the color scale indicates the log of the mass density and covers six orders of magnitude.

We conducted serial executions of ChaNGa and PKDGRAV on NCSA's Tungsten to compare scala-

Table 1. Characteristics of the parallel systems used in the experiments

System Name	Location	Number of Processors	Processors per Node	CPU Type	CPU Clock	Memory per Node	Type of Network
Tungsten	NCSA	2,560	2	Xeon	3.2 GHz	3 GB	Myrinet
Blue Gene/L	IBM-Watson	40,960	2	Power440	700 MHz	512 MB	Torus
Cray XT3	Pittsburgh	4,136	2	Opteron	2.6 GHz	2 GB	Torus

bility with varying numbers of particles. Figure 3(a) shows the results of this comparison using subsamples of *lambs*. As expected, the performance for both systems follows a linear aspect in this figure, because of the $O(N \log N)$ algorithm employed by both codes.

Despite their structural similarities PKDGRAV's serial performance is about 20% better than ChaNGa's. PKDGRAV is written in C whereas ChaNGa is a C++ code. The Intel *icc* compiler used for the tests uses different components to compile C and C++. As a result, similar code fragments in PKDGRAV and ChaNGa lead to differing object codes. This difference in object code creates the observed performance gap.

4.1 Parallel Performance on a Commodity Cluster

We conducted tests with parallel executions of ChaNGa on NCSA's Tungsten cluster, a typical representative of current Linux-based commodity clusters. We compared ChaNGa's performance on the gravity calculation phase to the performance of PKDGRAV. We used the *lambs* dataset, with 3 million particles, in a strong scaling test. Figure 3(b) shows the observed results. In this kind of plot, horizontal lines represent perfect scalability, while diagonal lines correspond to no scalability.

Figure 3(b) shows that ChaNGa's performance is lower than PKDGRAV's performance when the number of processors is small. This is due to the superior serial performance of PKDGRAV, as we had observed in Figure 3(a). However, for 8 or 16 processors, the two codes already reach similar performance levels. Beyond that range, ChaNGa clearly has superior performance. Since we did not use any load balancers with ChaNGa in this test, the main causes for the better performance of ChaNGa are its various optimizations that overlap computation and communication.

We can also observe from Figure 3(b) that ChaNGa's scalability starts to degrade at the right extreme of the horizontal axis. This happens because the *lambs* dataset is not large enough to effectively use 256 or more Tungsten processors. However, a similar degradation occurs for a much smaller number of processors in PKDGRAV. Hence, even for this relatively small problem size, ChaNGa presents significantly better parallel scalability than PKDGRAV.

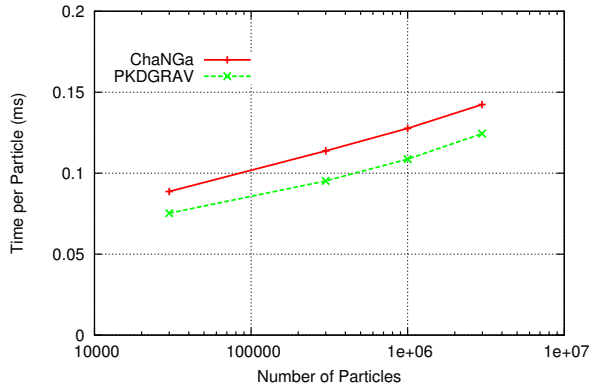
4.2 Parallel Performance on Blue Gene/L

In addition to commodity clusters, we considered the scalability of ChaNGa on high end parallel machines. Figure 4(a) reports scalability on the IBM Blue Gene/L system These results include a large variety of datasets, from the smallest *dwarf* (5 million particles) to the largest *drgas* (730 million particles). Due to Blue Gene/L memory limitations, larger datasets require larger minimal configurations. Given the difficulty in securing large numbers of Blue Gene/L processors for extended experimentation, we focused on running only the largest dataset on 32,768 processors. Near that level, the other datasets already start to show degraded scalability.

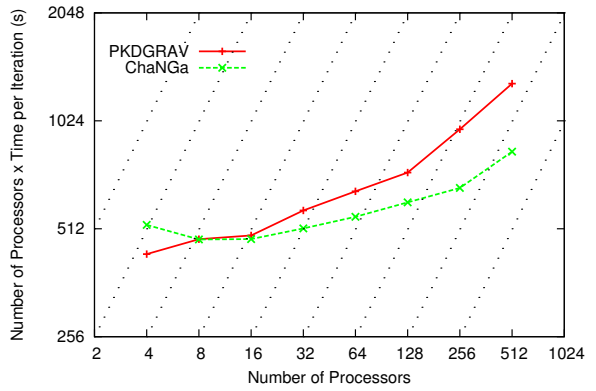
Over the entire range of simulation, we can see good scalability. Naturally, larger datasets scale better, having more computation to parallelize. In particular, on 8,192 processors, *lambb* performs better than *dwarf-50M*, despite *lambb*'s bigger size. This occurs because *lambb* has a more uniform cosmological particle distribution. Like in the case of NCSA's Tungsten, we did not consider advanced load balancing actions to improve performance in these tests. We will address this issue in section 5.1

4.3 Parallel Performance on the Cray XT3

We conducted executions of ChaNGa on the upgraded 2068-node Cray XT3 at PSC (`bigben`). Due to unavailability of the full machine for our tests,

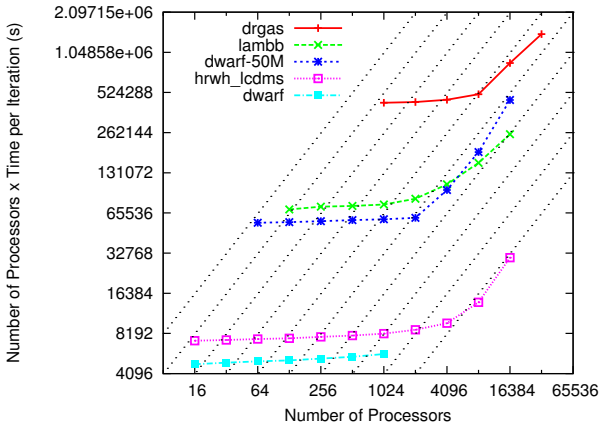


(a) Scalability with the dataset size, in serial mode, showing time per particle against number of particles.

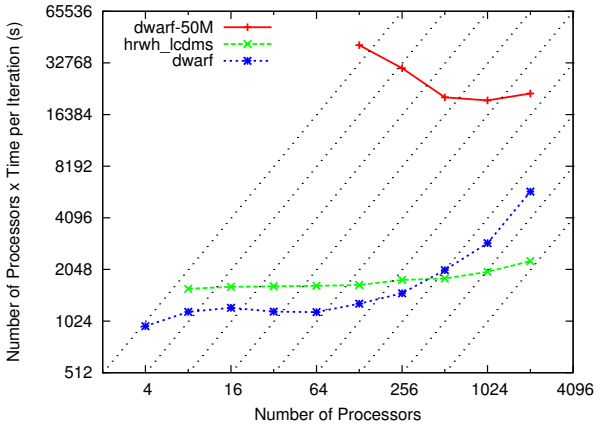


(b) Scalability with the number of processors (*lambs* dataset, 3 million particles)

Figure 3. Comparisons between ChaNGa and PKDGRAV on NCSA's Tungsten



(a) IBM Blue Gene/L



(b) Cray XT3

Figure 4. Parallel performance of ChaNGa on the IBM Blue Gene/L and Cray XT3.

we could not use more than 2048 processors. Figure 4(b) illustrates the results of ChaNGa with three distinct datasets: *dwarf* (5 million particles), *hrwh_lcdms* (16 million particles) and *dwarf-50M* (50 million particles). With the smallest dataset, we observe good scalability up to 256 processors. Beyond that point, performance degrades because the problem size is not large enough to utilize processors effectively. For the 16 million particle dataset, there is good scalability across the entire range of configurations used in the test. Remember that *hrwh_lcdms* is a uniform dataset, so that the load is even across processors and is sufficiently high to keep processors busy in all configurations tested.

Meanwhile, in the case of the largest dataset (*dwarf-50M*), the scalability is good up to 2048 processors and the data shows superlinear speedups from 128 to 512 processors. The program does not run on 64 or fewer processors due to memory limitations. We are still studying in detail the memory behavior of ChaNGa to understand and correct these anomalies. Nevertheless, these results show that ChaNGa achieves very good scalability on the Cray XT3 when adequate machine configurations are used for the various datasets.

5 Towards Greater Scalability

The results from the previous section demonstrate that scaling simulations beyond a few thousand processors remains a challenge in some cases,

even with the optimizations described in Section 3. In this section, we describe techniques that can further improve execution efficiency, especially on large system configurations. The first technique consists of specialized load balancers that take into account certain characteristics of particle codes. The second, multistepping, reduces operation count.

5.1 Specialized Load Balancing

The results reported in the previous section used a very simple load balancing strategy. The only distribution of work among processors was that of particles among TreePieces, which were then assigned by Charm++ uniformly to each processor. Since density can vary vastly over the simulated space, different particles may require significantly different amounts of work to compute the force they are subject to. Thus, even SFC-based domain decompositions, which assign an equal number of particles to each TreePiece, fail to obtain good load balance. Figure 5 shows an example of this imbalance. The shaded regions in the figure represent processor activity during different iterations of execution. In the first iteration there is significant imbalance of load across processors.

Among the CHARM++ suite of load balancers, GreedyLB is one that attempts to minimize load imbalance by continually assigning the heaviest remaining object to the least loaded processor. However, in our initial tests (not reported here), the results obtained with GreedyLB were not encouraging: in many cases, the execution time increased after load balancing, especially in large processor configurations. The reason for this lies in the CPU overhead to handle increased communication. In ChaNGa, due to the processor-level optimizations, TreePieces residing on the same processor do not need to exchange messages. Given that TreePieces containing particles close together in the simulation space need to exchange more data, a higher volume of communication will be generated if these TreePieces are all placed in different processors than in the case where some of them are coresident.

To verify our reasoning, we executed ChaNGa with the *dwarf* dataset on 1,024 Blue Gene/L processors. Table 2 summarizes the observed number of messages, amount of data transfered during one iteration, as well as the iteration time, including values both before and after load balancing. In addition to GreedyLB, we used other load balancing schemes that we describe later in this section. As

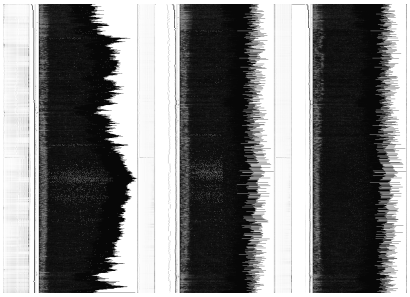


Figure 5. Overview of three iterations of *dwarf* simulation on 1,024 Blue Gene/L processors with OrbLB. Horizontal axis is time and each horizontal line represents activity on a processor.

shown, the communication volume after GreedyLB grows nearly three times.

The connection between this increase in communication volume and the reduced performance is shown graphically in the processor utilization view of Figure 6. Here, orange and blue regions (the two largest) constitute the global and local force computations, respectively. The different components of communication overhead are shown in red and green. Notice that the second iteration, which begins after GreedyLB load balancing, suffers a higher CPU overhead for communication, as evidenced by the larger red and green regions. The numbers in Table 2 confirm this fact. Due to this overhead, processors are not fully utilized by the application, as communication is handled by the runtime system. On the other hand, the steeply sloped profile of iteration two reflects better load balance.

These results demonstrate that an effective load balancer must take communication into account. However, basing decisions on communication between objects is very difficult, since the particle cache, acting as a proxy between TreePieces, obscures this information. Furthermore, because communication depends on the location of the other TreePieces in the system, the communication data can only be estimated. To complicate matters further, creating a precise communication graph is expensive, and requires a large memory system.

Instead of estimating the point-to-point communication, we use SFC domain decomposition and the OrbLB load balancer in tandem to arrive at a heuristic for object placement. The SFC decomposition procedure assigns particles to TreePieces

Table 2. Communication volume and execution time with *dwarf* on 1,024 Blue Gene/L processors using different load balancing strategies

Perf. parameter (per iteration)	Original	GreedyLB	OrbLB	OrbRefineLB
Messages exchanged (x 1,000)	5,480	16,233	5,590	8,370
Bytes transfered (MB)	7,125	23,891	7,227	9,873
Execution time (s)	5.6	6.1	5.3	5.0

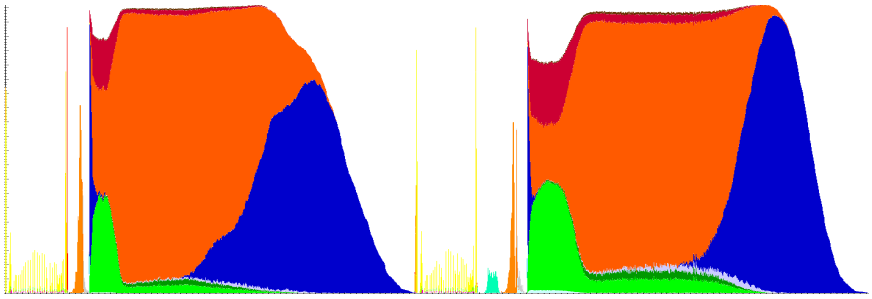
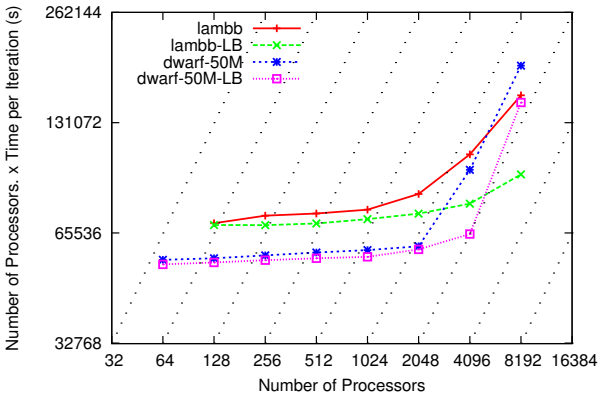


Figure 6. Two iterations of *dwarf* simulation on 1,024 Blue Gene/L processors, separated by a GreedyLB load balancing phase. Horizontal axis is time and vertical axis is combined processor utilization. Colors represent different activities.

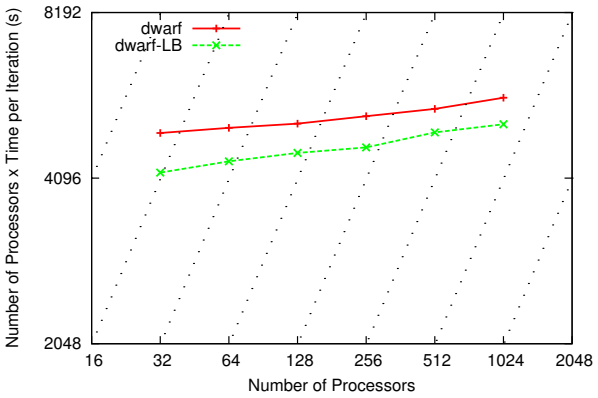
preserving an interesting property: it ensures that contiguous regions of the simulation space are represented by TreePieces having adjacent Charm++ identifiers (`objId`'s). Further, OrbLB begins by sorting objects according to their `objId`'s. It then proceeds recursively by assigning roughly balanced 'halves' of this sorted set to groups of processors, one 'half' to each. Therefore, when applied to TreePieces generated by such a decomposition, OrbLB would implicitly base its placement decision on the simulation space coordinates of a TreePiece. In other words, OrbLB makes use of the admittedly rough correspondence between a TreePiece's `objId` and its location in the simulation space. In this manner, TreePieces likely to exchange large amounts of information are assigned to contiguously ranked processors. Our placement heuristic is a coarse one, since we use the position of TreePieces along a unidimensional SFC line, whereas the simulation space is three dimensional. The results from OrbLB in Table 2 show that the communication volume remains about the same, while the overall performance is improved. Figure 5 shows processor load in more detail, with execution iterations separated by OrbLB phases.

From Figure 5, it is clear that while global load distribution is improved by OrbLB, processors still have very different loads. This observation motivates the use of a hybrid approach, which we name OrbRefineLB: the high level distribution of TreePieces among processors is performed by OrbLB, but neighboring processors (i.e. processors with contiguous ranks) exchange objects according to a greedy algorithm. This strategy yielded the results in the last column of Table 2. As can be seen, a small increase in communication volume results from the 'smoothing,' but the gains in load balance offset this slight disadvantage. With this hybrid load balancer, we re-ran three simulations, *dwarf*, *dwarf-50M* and *lambb*, on Blue Gene/L. Figure 7 shows the corresponding reductions in execution times due to OrbRefineLB.

Although we obtain a reduction in execution time by using OrbRefineLB, the total communication volume increases. For this reason, we continue to look into more advanced load balancing techniques that will allow us to balance work across the system without disrupting the application's communication pattern. In particular, we are looking at load balancers that are aware of the spatial layout of objects in all three dimensions. In another direction, we are looking to reduce the cost per message exchanged by using the communication optimization framework of Charm++.



(a) *dwarf-50M* and *lambb* datasets



(b) *dwarf* dataset

Figure 7. Scalability of ChaNGa before and after load balancing on Blue Gene/L.

5.2 Multistepping

Gravitational instabilities in the context of cosmology lead to a large dynamic range of timescales: from many gigayears on the largest scales to less than a million years in molecular clouds. In self gravitating systems this dynamical time scales roughly as the square root of the reciprocal of the local density. Hence density contrasts of a million or more from the centers of galaxies to the mean density of the Universe correspond to a factor of 1000 in dynamical times. Indeed, large density contrasts are the primary reason to use tree-based gravity algorithms in cosmology.

Typically, only a small fraction of the particles in a cosmological simulation are in the densest environments; therefore, the majority of the particles do not need their forces evaluated on the smallest timestep. Large efficiencies can be gained by a *multistep* integration algorithm which on the smallest timestep updates only those particles in dense regions, and only infrequently calculates the forces on all the particles. This type of algorithm poses an obvious challenge for load balancing. Much of the time, work from a small spatial region of the simulation must be distributed across all the processors, but on the occasional large timestep, the work involves all the particles.

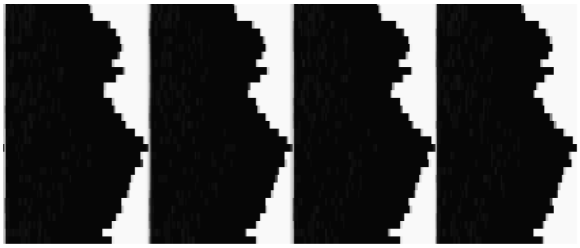
Table 3. Ratio between singlestepped and multistepped gravity calculation times for *dwarf* simulations on NCSA's Tungsten

Processors	1	4	8	16	32	64
Ratio	3.29	2.65	2.11	1.79	1.39	1.55

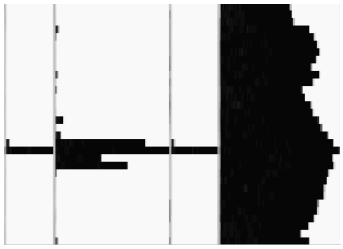
To study the influence of multistepping on ChaNGa's performance, we ran simulations of the *dwarf* dataset on NCSA's Tungsten. We started our tests with two sequential executions, selecting appropriate values of the simulated timestep such that one execution would be in singlestepping mode and the other in multistepping mode. Both executions simulated the same amount of cosmological time. For our chosen parameters, four timesteps in the singlestepping case corresponded to one big step in the multistepping case. Next, we repeated the pair of executions, varying the number of Tungsten processors used in each case. Table 3 compares the execution times obtained under those two modes.

It becomes clear from Table 3 that multistepping can greatly accelerate a simulation. On a single processor, we observe more than a three-fold savings in execution time when multistepping is used. The serial execution performance gain comes exclusively from the algorithm used for multistepping. This result confirms the importance of multistepping capability for the computational performance of a cosmological simulator.

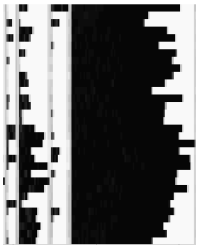
Table 3 indicates that multistepping provides performance gains across the entire range of processors tested. Although the gain with 64 processors is still quite significant, the table reveals a trend of slight decrease in the gain as the number of processors increases. We investigated this issue and found that load imbalance was the cause of that effect, as we explain in the next subsection.



(a) Duration: 613.3 s



(b) Duration: 428.7 s



(c) Duration: 228 s

Figure 8. Comparison of phase durations among (a) singlestepped, (b) multistepped and (c) load balanced multistepped simulations of the *dwarf* dataset on 32 Blue Gene/L processors.

5.3 Balancing Load in Multistepped Executions

To better understand the load imbalance problem in multistepped executions, we studied ChaNGa executions on 32 processors of Blue Gene/L. Figures 8(a) and 8(b) correspond to parts of the execution of the singlestepping and multistepping cases, respectively. In both figures, the horizontal axis represents time, the vertical axis corresponds to processors (32 in both cases). Shaded horizontal lines represent processor activity during the simulation.

In Figure 8(a), where singlestepping was used, there are four steps displayed. Figure 8(b) represents a big step, corresponding to the same cosmological duration as the four steps of singlestepping. The computation for this big step is divided into four substeps, and only select particles are active in each. In turn, only the corresponding TreePieces are active, leaving most with no work to do at all! As Figures 8(a) and 8(b) show, multistepping affords significant performance gains.

However, this scheme is not without its drawbacks. The first three substeps of Figure 8(b) demonstrate severe load imbalance – only a few processors are busy, while the others remain mostly idle. Such imbalance is stressed by the characteristics of the *dwarf* dataset, which has a highly non-uniform particle distribution. As the number of processors grows, this imbalance increases, making parallelism less effective than in singlestepping.

From our discussion, it is clear that measurement based load balancing in multistepped executions is significantly more challenging than for the singlestepped case. Since different numbers of particles are active in each substep, load information from an immediately preceding substep is unusable in the current one. We must, therefore, correlate load across *instances* of corresponding substeps.

Each substep can be viewed as a different *phase* of execution. In Figure 8(b), the first and third substeps correspond to phase 0, since they involve only the fastest particles. Similarly, substep two constitutes phase 1 and the last represents phase 2. While the principle of persistence does not apply to neighboring substeps, it does hold across different instances of any phase. That said, the set of particles involved in the force computation of a certain phase is not a static entity. It changes during the course of the simulation, with variations in particle velocities. Clearly, a load balancer must take the multiphase nature of the simulation into account when considering a placement strategy.

5.3.1 Balancing Techniques

We now outline a few techniques we devised in meeting the challenges posed by the problem of severe load imbalance during multiphase simulations.

Different strategies for different phases. Different phases of a computation present the load balancer with vastly differing load profiles. In Figure 8(b), the utilization profile for the first substep shows that only a few TreePieces were active during this phase. On the other hand, the last substep involves all the TreePieces. In such an event, significant gains in performance can be achieved by using fast and simple schemes for lower ranked phases, i.e. those involving fewer TreePieces. The time taken by the load balancer to compute placement decisions for such phases is critical, since they occur very often. Using a fast, greedy load balancing algorithm for the smaller phases, and the OrbRefineLB strategy (Section 5.1) when all particles are active, we obtain promising reductions in execution time, as a comparison between Figures 8(b) and 8(c) illustrates. There are marked improvements both be-

Table 4. (a) Comparison between execution times for one big step of single- and multistepped runs on 512 and 1024 processors of Blue Gene/L, (b) Reduction in execution time with increase in number of TreePieces on 512 Blue Gene/L processors.

Number of Processors	512	1,024
Singlestepped time(s)	4,652.16	2,387.52
Multistepped time(s)	2,023.98	1,286.37

(a)

Number of TreePieces	4,096	8,192	16,384
Time (s)	2,524.97	2,198.91	2,023.98

(b)

tween corresponding phases and in the overall time.

Multiphase instrumentation. We measure and store object loads during different phases separately. This is required since certain TreePieces are active over multiple phases. Indeed, TreePieces with the quickest particles are active over all computation phases and might induce a different load in each.

Model-based load estimation. We noted earlier that the principle of persistence holds over multiple instances of the same phase. However, the quickest particles, in moving from one region of space to another, might switch across distinct TreePieces between iterations. This might lead to large discrepancies between estimated and actual loads of TreePieces, especially for lower-ranked phases. For this reason, we allow for a model-based estimation of phase load. In our multiphase load balancer, we estimate the load of a TreePiece during phase ϕ as the corresponding value from the immediately preceding instance of ϕ. If that information is unavailable, we use a fraction f of the load from the *first* step. This first step involves all particles and executes before any other. The fraction f is the ratio of active particles to total particles in a TreePiece.

5.3.2 Preliminary Results

We now describe some preliminary results obtained by using the principles outlined previously. We simulated the 80M particle dataset, *lambb*, using multitimestepping and our multiphase load balancer. For brevity, we only present results from 512 and 1024 processor runs on Blue Gene/L.

Table 4(a) illustrates the complementary effects of multistepping and load balancing. It compares execution times of a load balanced multistepped simulation and the same simulation when run in singlestepped mode, without load balancing. There are savings of nearly 50% on 1024 processors.

Table 4(b) brings a related issue to light, that of the degree and grain of parallelism. We noticed a significant reduction in computation time as the number of TreePieces was increased for a given number of processors. With more TreePieces in the system, the load balancer is able to maintain a more even balance of work in the most frequently repeated phases. This prevents such phases from becoming the execution bottleneck. Indeed, the overhead of finer grained work distribution is offset by the prospect of greater balance in system load. For this reason multistepped executions stand to benefit even more from overdecomposition than their singlestepped counterparts.

These results motivate more detailed analyses of multiphase simulations in the context of load balance. In particular, we continue to develop more sophisticated load balancing strategies. For instance, the ORB assignment of TreePieces to processors in the OrbRefineLB strategy could be three dimensional. Further, the exchange of objects therein could be sensitive to machine topology. Finally, multiphase computations could benefit from better heuristics and more precise load estimation models.

6 Conclusions and Future Work

In this paper, we described our highly scalable parallel gravity code ChaNGa, based on the well-known Barnes-Hut algorithm. Scaling this computation to tens of thousands of processors requires a sophisticated load balancing strategy. To enable such automatic load balancing schemes, we developed ChaNGa using the Charm++ parallel programming system and its adaptive Run-Time System. Since Charm++ requires overdecomposition to empower its load balancing strategies, ChaNGa decomposes the particles into a large number of *TreePieces*. Multiple TreePieces assigned to a single processor share a software cache that stores nodes of the tree requested from remote processors as well as local nodes. Requests for remote data are effectively pipelined to mask any communication latencies.

Although using CHARM++ enables load balancing, an appropriate load balancer must be chosen or written for each application. A pure load-based balancer was not adequate for this problem. We showed that a combination of a coordinate-based balancer, along with a measurement based refinement strategy, led to good load balance without increasing communication overhead significantly. We demonstrated unprecedented speedups in cosmological simulations based on real datasets, scaling well to over 8,000 processors.

A new multiple time-stepping scheme was described that reduces operation count, but is even more challenging to load balance. We demonstrated a multi-phase load balancing scheme that meets this challenge, on several hundred processors.

Our code is currently being used for gravity computations by astrophysicists. We intend to add hydrodynamics to it in the future. Alternative decomposition schemes to further reduce communication, and runtime optimizations to mitigate the impact of communication costs are also planned.

Acknowledgments

This work was supported in part by NFS grant ITR-0205611 and by Teragrid allocations ASC050039N and ASC050040N. We thank Shawn Brown for help with the Cray XT3 at PSC and Glenn Martyna and Fred Mintzer at IBM Research for the use of BGW (BlueGene/L at Watson).

References

[1] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, December 1986.

[2] W. Dehnen. A hierarchical $O(N)$ force calculation algorithm. *Journal of Computational Physics*, 179:27–42, 2002.

[3] M. D. Dikaiakos and J. Stadel. A performance study of cosmological simulations on message-passing and shared-memory multiprocessors. In *Proceedings of the International Conference on Supercomputing - ICS'96*, pages 94–101, Philadelphia, PA, December 1996.

[4] F. Gioachin, A. Sharma, S. Chakravorty, C. Mendes, L. V. Kale, and T. R. Quinn. Scalable cosmology simulations on parallel machines. In *VECPAR 2006, LNCS 4395, pp. 476-489*, 2007.

[5] F. Governato, B. Willman, L. Mayer, A. Brooks, G. Stinson, O. Valenzuela, J. Wadsley, and T. Quinn. Forming disc galaxies in ΛCDM simulations. *MNRAS*, 374:1479–1494, Feb. 2007.

[6] L. Greengard and V. I. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73, 1987.

[7] K. Heitmann, P. M. Ricker, M. S. Warren, and S. Habib. Robustness of Cosmological Simulations. I. Large-Scale Structure. *ApJSup*, 160:28–58, Sept. 2005.

[8] L. V. Kalé. Performance and productivity in parallel programming via processor virtualization. In *Proc. of the First Intl. Workshop on Productivity and Performance in High-End Computing (at HPCA 10)*, Madrid, Spain, February 2004.

[9] L. V. Kale and S. Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In G. V. Wilson and P. Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.

[10] A. Kawai and T. Fukushige. $158/Gflops astrophysical N-body simulation with a reconfigurable add-in card and a hierarchical tree algorithm. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006. ACM Press.

[11] A. Kawai, T. Fukushige, and J. Makino. $7.0/Mflops astrophysical N-body simulation with treecode on GRAPE-5. In *SC '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 1999. ACM Press.

[12] G. Lake, N. Katz, and T. Quinn. Cosmological N-body simulation. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 307–312, Philadelphia, PA, February 1995.

[13] P. Londrillo, C. Nipoti, and L. Ciotti. A parallel implementation of a new fast algorithm for N-body simulations. *Memorie della Societa Astronomica Italiana Supplement*, 1:18–+, 2003.

[14] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kalé. NAMD: Biomolecular simulation on thousands of processors. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–18, Baltimore, MD, September 2002.

[15] V. Springel. The cosmological simulation code GADGET-2. *MNRAS*, 364:1105–1134, 2005.

[16] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature*, 435:629–636, June 2005.

[17] V. Springel, N. Yoshida, and S. White. GADGET: A code for collisionless and gasdynamical simulations. *New Astronomy*, 6:79–117, 2001.

[18] M. S. Warren and J. K. Salmon. Astrophysical n-body simulations using hierarchical tree data structures. In *Proceedings of Supercomputing 92*, Nov. 1992.

[19] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of Supercomputing 93*, Nov. 1993.

[20] G. Zheng. *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.