Towards Petascale Cosmological Simulations with ChaNGa

Filippo Gioachin Pritish Jetley Celso L. Mendes Laxmikant V. Kalé
Thomas Quinn

Abstract

Cosmological simulators are currently 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. Other optimizations described in the paper include a particle caching mechanism and a recently added multi-timestepping scheme. The paper shows experimental results demonstrating scalable performance for simulations on machines with thousands of processors, including IBM-BlueGene/L and Cray-XT3. We conclude by pointing to our current work, which focuses on developing new load balancing strategies to further improve ChaNGa's parallel efficiency. To our knowledge, ChaNGa is presently the most scalable cosmological simulator available.

1 Introduction

Cosmological simulators are currently an important component in the study of the formation of galaxies and large scale structure. Galaxies are the most distinctive objects in the universe, containing almost all the luminous material. They are remarkable dynamical systems, formed by non-linear collapse and a drawn-out series of mergers and encounters. Galaxy formation is indeed a challenging computational problem, requiring high resolutions and dynamic timescales. For example, to form a stable Milky Way-like galaxy, tens of millions of resolution elements must be simulated to the current epoch. 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. No existing N-body/Hydro solver can handle

this regime efficiently.

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

To address these issues, various cosmological simulators have been created recently. PKDGRAV [1], developed at the University of Washington, can be considered among the state-of-the-art in that area. However, 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 to compute gravitational forces. Our recently released simulator, named ChaNGa, is based on the Charm++ runtime system [2]. We leverage the object based virtualization [3] inherent in the Charm++ runtime system to obtain automatic overlapping of communication and computation time, as well as to perform automatic runtime 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 with moderate programming effort. In addition, as confirmed by our efficiently scale on large machine configurations.

The remainder of this paper is organized as follows. In Section 2 we present 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 describes some of the advanced features recently introduced in the code. Finally, Section 6 contains our conclusions and the future directions of our work.

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 [4]. 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 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.

Hierarchical methods for N-Body simulations have been adopted for quite some time by astronomers [5, 6]. A widely used code in that area is PKDGRAV [1], 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 [7], developed in Germany, and falcON [8], originally developed at the University of Maryland. falcON has shown good scalability with the number of particles, but it is a serial simulator. GADGET, which is now is its second version (GADGET-2), has a rich modeling capability, but its reported scalability results have been limited to 128 processors or less [9]. Meanwhile, some special purpose machines have been developed in the past to simulate cosmological N-body problems. Notable examples were Grape-5 [10], which had 32 pipeline processors specialized for the gravitational force calculation, and an FPGA-based system [11] constructed as an add-in card connected to a host PC. Both systems achieved great price/performance points, but they had severe memory constraints: the largest datasets that one could use on them were limited to two million particles.

3 Structure of the ChaNGa Code

ChaNGa is implemented as a Charm++ application. As such, it leverages 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. We also describe the multistepping and advanced load-balancing capabilities, which we recently introduced in the code.

3.1 Charm++ Infrastructure

Charm++ is a parallel software infrastructure that implements the concept of *processor virtualization* [2]. 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 decomposition) independent of the number of available processors, and enables execution of the same code on different machine configurations. With this separation between logical and physical abstractions, Charm++ provides higher programmer productivity. This scheme has allowed the creation of parallel applications that

scale efficiently to thousands of processors, such as the molecular dynamics NAMD code [12], a winner of a Gordon Bell award in 2002.

Besides mapping and scheduling chares on available processors, the CHARM++ runtime system also has the ability to dynamically migrate chares across processors. This capability is used to implement a powerful measurement-based load balancing mechanism in CHARM++ [13]. Chares can be migrated based on observed values of various metrics, such as computation loads or communication patterns. Hence, as the execution proceeds, load can be removed from overloaded processors and redistributed to underutilized ones. This dynamic redistribution is critical to achieving 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 described in [14], 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*. The various TreePieces are distributed to the available processors for parallel computation of the gravitational forces. ChaNGa allows various distribution schemes, including 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 value corresponding to all particles under that node is used to calculate forces; otherwise, it 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 in fact essential to achieve acceptable performance [14].

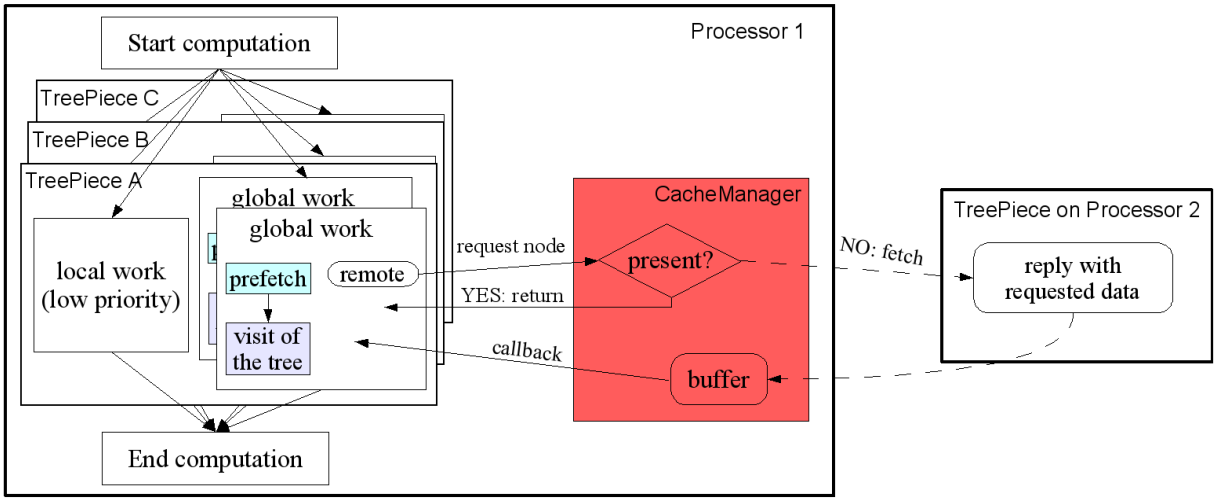


Figure 1: Components for parallel calculation of gravitational forces in ChaNGa

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 [14]):

- **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 determine the remote data that will be needed in the regular tree traversal. Such data is prefetched into the local cache before the corresponding computation is started.

- **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 makes use of 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 provides a good tradeoff between responsiveness to communication requests and processor utilization.

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
BlueGene/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

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

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 [15], and is slightly more uniform than *lambs*.

dwarf-50M : This dataset is the 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 This dataset is the same physical model as *lambs* except that it is realized with 80 million particles.

drgas This dataset is 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.

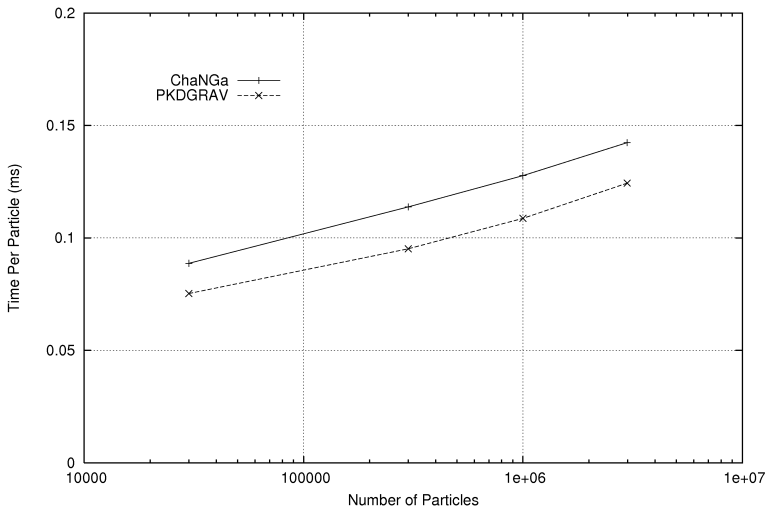


Figure 2: Scalability of ChaNGa with the dataset size, in serial mode

4.1 Serial Performance

We started our tests by addressing the issue of scalability with the number of particles. To assess this, we conducted serial executions of ChaNGa and PKDGRAV on NCSA's Tungsten, under varying dataset sizes. Figure 2 shows the comparison between ChaNGa and PKDGRAV on these tests. Notice that the horizontal axis in this figure (number of particles) is represented in log scale. 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 the similarities in the computational structures of the two codes, PKDGRAV's serial performance is slightly better (about 20 %) than ChaNGa's performance. We found that this is due to a compiler issue. While both codes were compiled by the Intel *icc* compiler, PKDGRAV is entirely a C code, whereas ChaNGa is written mostly in C++. Hence, the codes are handled by different parts of *icc* (in particular, *icc* invokes *icpc* for C++ codes). The net result is that similar fragments in the two codes may end up producing different object code. Nevertheless, for the two codes the performance obtained with Intel's *icc* is significantly superior to what is achievable with the GNU compiler.

4.2 Parallel Performance on a Commodity Cluster

We conducted tests with the parallel version of ChaNGa on NCSA's Tungsten cluster, which is a typical representative of current Linux-based commodity clusters. We compared ChaNGa's performance on the gravity calculation phase to the performance of PKDGRAV. In these tests, we used the original *lambs* dataset, with 3 million particles. This experiment consisted of a strong scaling test, since we used the same

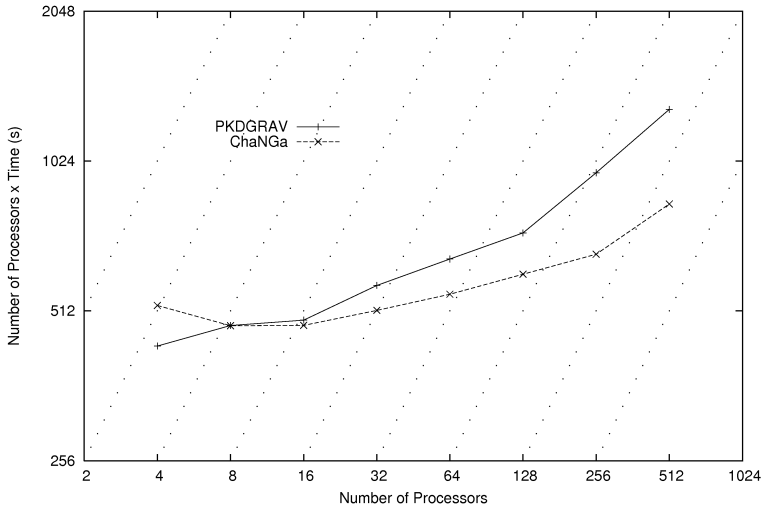


Figure 3: Parallel performance of ChaNGa and PKDGRAV on NCSA's Tungsten with 3 million particles

dataset size with a varying number of processors. Figure 3 shows the observed results. In this kind of plot, horizontal lines represent perfect scalability, while diagonal lines correspond to no scalability.

Figure 3 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 2. 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 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.

4.3 Parallel Performance on BlueGene/L

In addition to commodity clusters, we considered the scalability on high end parallel machines. In Figure 4 we report the scalability on the IBM BlueGene/L machine, repeated in table 2. These results include a large variety of datasets, from the smallest *dwarf* (5 million particles) to the largest *drgas* (730 million particles). Due to BlueGene/L memory limitations, larger datasets require larger minimum configurations to allow

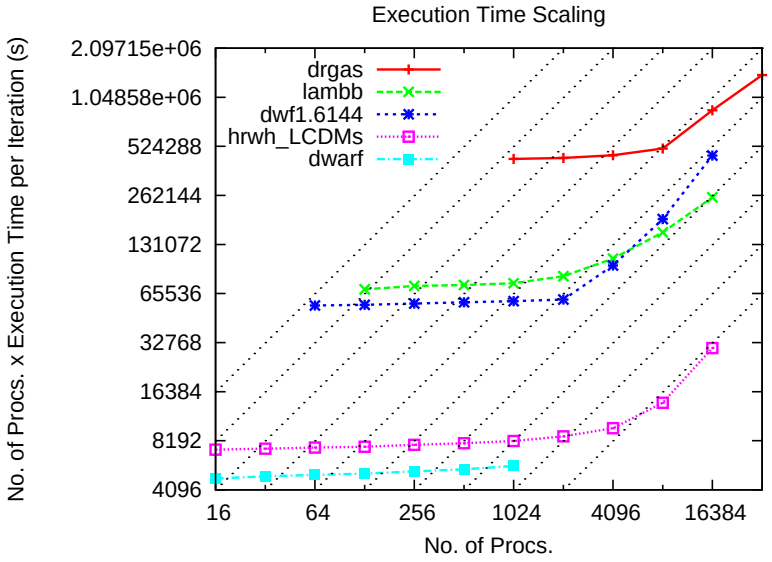


Figure 4: Parallel performance of ChaNGa on IBM's BlueGene/L with various datasets

	32	64	128	256	512	1024	2048	4096	8192	16384	32768
dwarf	154.64	78.97	40.20	20.74	10.69	5.60					
hwrh_lcdms	228.73	116.19	58.79	30.17	15.43	7.97	4.26	2.39	1.71	1.85	
dwarf-50M		864.37	436.54	222.18	113.19	57.42	29.42	23.78	22.90	28.10	
lambb			543.97	285.22	144.55	74.09	40.88	26.21	19.00	15.58	
drgas						428.49	217.01	112.91	62.08	53.20	43.80

Table 2: Parallel performance of ChaNGa on IBM's BlueGene/L with various datasets in table format

loading all the input. Given the difficulty in securing large numbers of BlueGene/L processors for extended experimentation, we focused on running only the largest of the datasets on 32,768 processors. Near that level, the other datasets have already started to show degraded scalability.

Over the entire range of simulation, we can see an excellent scalability. Naturally, larger datasets scale better, having more computation to parallelize. In particular, on 8,192 processors, *lambb* performs better than *dwarf-50M*, despite its 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 specific load balancing actions to improve performance in these tests. We will address this issue in the next Section.

4.4 Parallel Performance on the Cray XT3

In this subsection, we analyze the parallel performance of ChaNGa on the Cray XT3. The experiments detailed herein were carried out on the upgraded 2068-node XT3 at PSC, `bigben`, so that there were 2GB

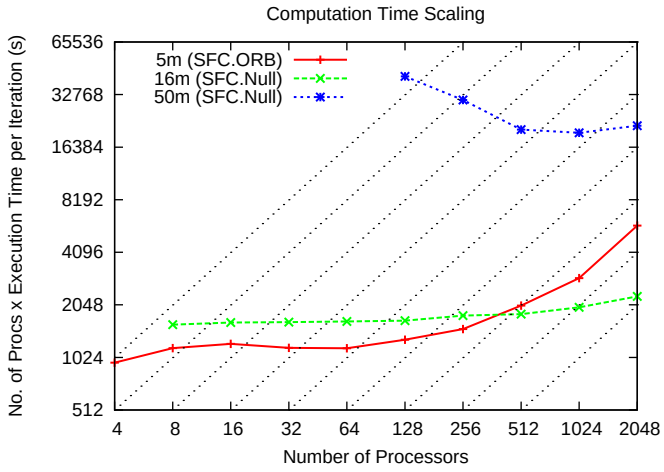


Figure 5: Parallel performance of ChaNGa on the Cray XT3 with distinct datasets

	8	16	32	64	128	256	512	1024	2048
dwarf	144.78	76.54	36.31	18.06	10.11	5.81	3.96	2.85	2.84
hwrh_lcdms	197.2	101.44	51.0	25.69	12.99	6.95	3.54	1.94	1.12
dwarf-50M					325.03	119.22	40.3	19.32	10.62

Table 3: Parallel performance of ChaNGa on the Cray XT3 with distinct datasets in table format

of memory and one dual-core 2.6 GHz AMD processor on each of the 2068 nodes. Note that the results do not include executions with 4096 processors, due to unavailability of the entire machine for our tests.

Figure 3 illustrates the results of ChaNGa executions with three distinct datasets: *dwarf* (5 million particles), *hrwh_lcdms* (16 million particles) and *dwarf-50M* (50 million particles). The same data is replicated in table **??**. With the smallest dataset (*dwarf*), 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 upto 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, leading us to suspect that memory issues are causing these aberrations. Nevertheless, these results show that ChaNGa can achieve very good scalability on the Cray XT3 when proper dataset sizes and machine configurations are employed.

5 Recent Extensions in ChaNGa

In this section, we describe two new features that we have recently introduced in ChaNGa. The first feature consists in specialized load balancers that take into account the characteristics of particle codes. The second feature, multistepping, allows a better modeling of the particles' movement when the velocities of individual particles are spread across a wide range of values.

5.1 Specialized Load Balancing

In all the experimental results reported in the previous section, the issue of load balancing was not considered explicitly. The only distribution of work among processors was the division of the particles among the TreePieces. These TreePieces were then assigned by Charm++ uniformly to each processor. Due to the property that different regions of the simulated space have different density of particles, different particles may require a significantly different amount of work to compute the force they are subject to. This implies that even SFC-based domain decompositions, which assign an equal number of particles to each TreePiece, fail to obtain a good load balance. An example can be seen in the first iteration represented in Figure 7, and described in more detail later in this subsection.

As in every application facing load balancing issues, the main problem is to distribute the work, TreePieces in our case, such that there is no overloaded processor that would delay the entire execution. Among the Charm++ suite of load balancers, GreedyLB is a load balancer whose purpose is exactly this one. Nevertheless, in preliminary tests that we conducted (not reported here), the results obtained with GreedyLB were not successful: in many cases, the execution time increased after load balancing, especially in large processor configurations. The reason for this comes from the CPU overhead to handle communication. In ChaNGa, due to the processor-level optimizations introduced, TreePieces residing on the same processor do not need to exchange messages, and will reuse the same data fetched previously. Given that TreePieces containing particles closer in the simulation space need to exchange more data, if these TreePieces are all placed in different processors, a higher volume of communication will be generated than in the case where some of them are in coresident.

To analyze those issues, we executed ChaNGa with the *dwarf* dataset on 1.024 BlueGene/L processors. Table 4 summarizes observed number of messages, amount of data transfered during one iteration of ChaNGa, as well as the iteration time itself. This table includes values both before and after load balancing, with GreedyLB and other load balancers described later in this subsection. As the table shows, the communication

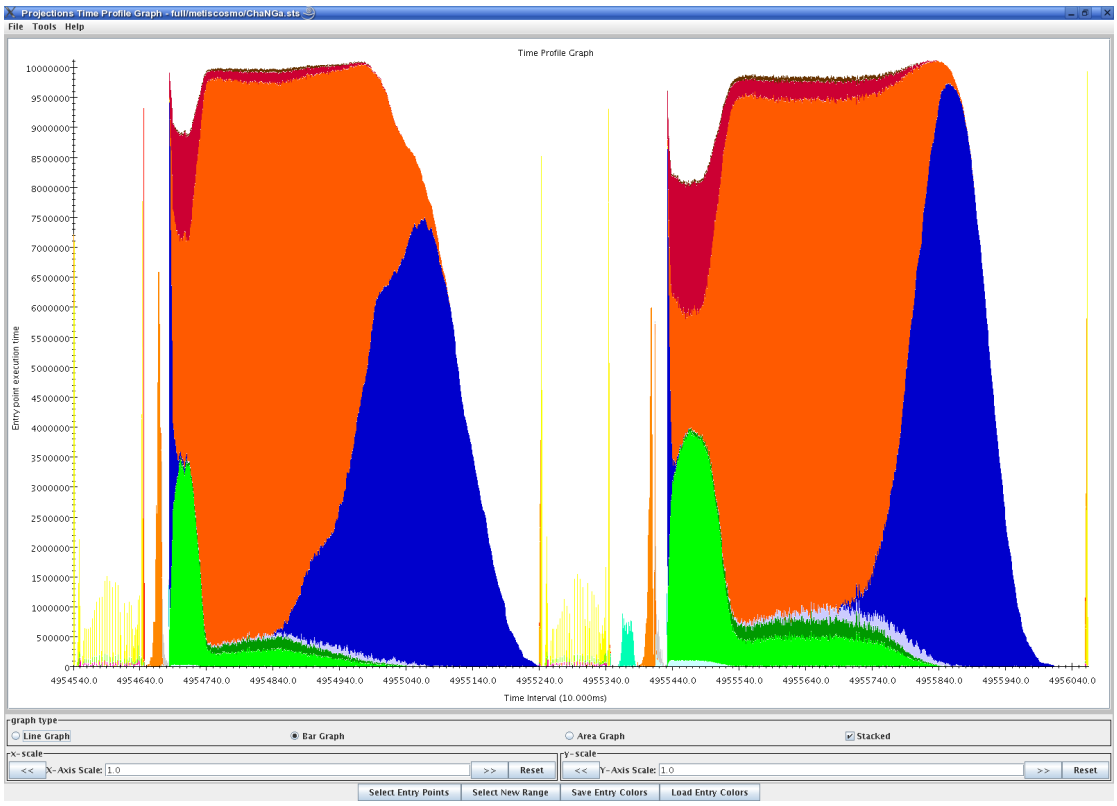


Figure 6: PROJECTIONS time profile of two iterations of *dwarf* simulation on 1,024 BlueGene/L processors. Horizontal axis is time and vertical axis is combined processor utilization. Colors represent different activities.

volume after GreedyLB is nearly three times larger than the original volume.

	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

Table 4: Communication volume and execution time with *dwarf* on 1,024 BlueGene/L processors using different load balancing strategies

The connection between this increase in communication volume and the reduced performance is shown graphically in Figure 6. Here, the horizontal axis is time and the vertical axis corresponds to processor utilization. The different colors represent different operations. Orange and blue (the two biggest areas) refer to the force computation, global and local respectively. The other colors represent overhead for communication. It can be easily seen that the second iteration after load balancing contains a higher CPU overhead for communication, as the number in table 4 showed. Moreover, in correspondence with this overhead, the processors are not fully utilized by the application, as some work will be performed by the underlying runtime

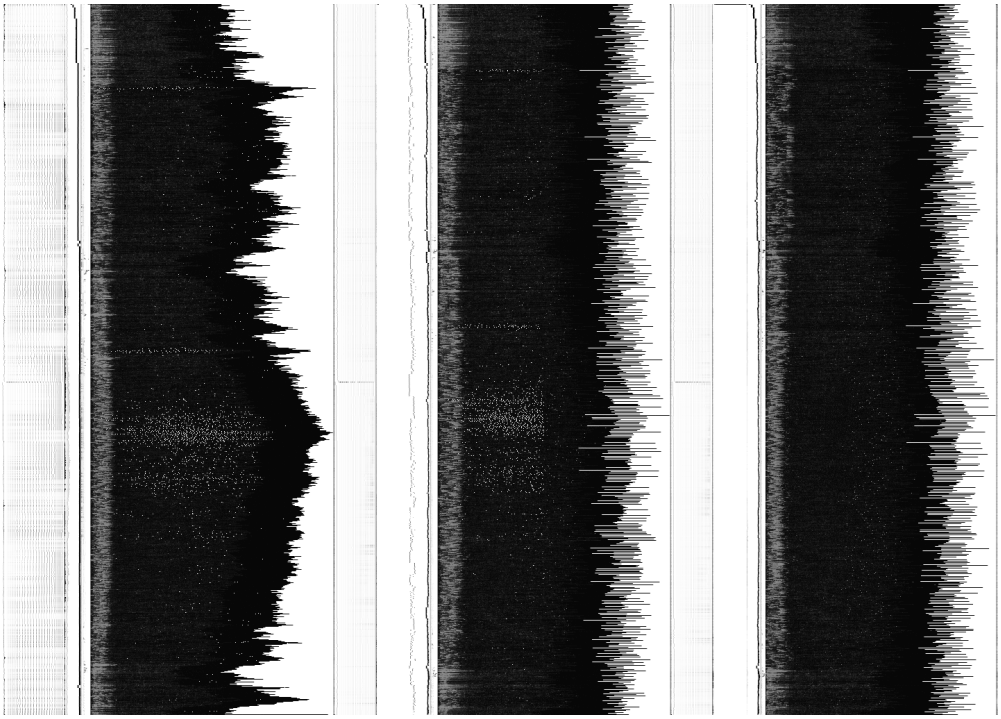


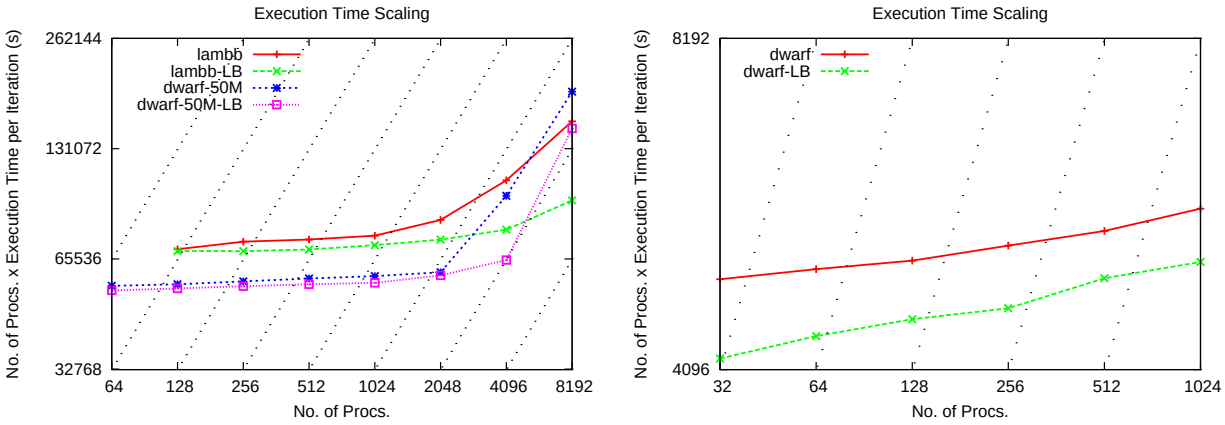
Figure 7: PROJECTIONS overview of three iterations of the *dwarf* simulation on 1,024 BlueGene/L processors. Horizontal axis is time and each horizontal line represents a processor.

system. A positive aspect of this load balancer is that the slope at the end of the iteration is steeper. This means that the work among processors is better balanced than before.

These results indicate that any effective load balancer for ChaNGa will have to take into consideration the effect of communication. Looking at the point-to-point communication between TreePieces is very difficult, since the particle cache will be hiding most of it. Furthermore, the communication depends on the location of the other TreePieces in the system. Hence, the communication data can only be estimated. Moreover, creating a precise communication graph would be expensive, and require a large memory system.

Instead of estimating the point-to-point communication, given that we already noted that TreePieces closer in space will communicate more, we can simply assume this spatial positioning as heuristic for communication volume and try to place TreePieces accordingly. We used another of the CHARM++ load balancers: OrbLB. This load balancer will divide the TreePieces according to their position in the decomposition line. Unfortunately, this is only a unidimensional decomposition, and will loose the other two dimensions of the simulation volume. The results from OrbLB are also shown in Table 4, where we can see that the communication volume remains the same and we are able to improve performance.

Analyzing in more detail the load of each processor with PROJECTIONS' overview (figure 7), it appears



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

(b) *dwarf* dataset

Figure 8: Scalability of ChaNGa before and after load balancing on BlueGene/L.

clear that while the load distribution is improved, processors still have very different loads. Therefore, we decided to utilize a hybrid approach: the high level distribution of the TreePieces to processors is performed by OrbLB, but neighbouring processors will exchange their objects with a Greedy algorithm. This strategy, which we call OrbRefineLB, led to the values in the last column of table 4. Here we can see that this hybrid approach based on spacial locality is winning, even though it slightly increases the communication volume.

With this hybrid version of the load balancer, we re-ran two simulations, *dwarf-50M* and *lambb*, on BlueGene/L. Figures 8 shows the improvements of the execution time from before to after load balancing. There is a noticeable reduction in time, especially in the larger processor configurations.

Although we obtained a reduction of execution time by using OrbRefineLB, the total communication volume increased. For this reason, we are continuing to research for more advanced load balancing techniques that will allow us to balance the work without disrupting the communication pattern of the application. In particular, we are looking at load balancer that utilize all the three dimensions of the space. In another direction, we are also looking at reducing the cost per message exchanged by using the communication optimization framework of Charm++.

5.2 Multistepping

Gravitational instabilities in the context of cosmology leads 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

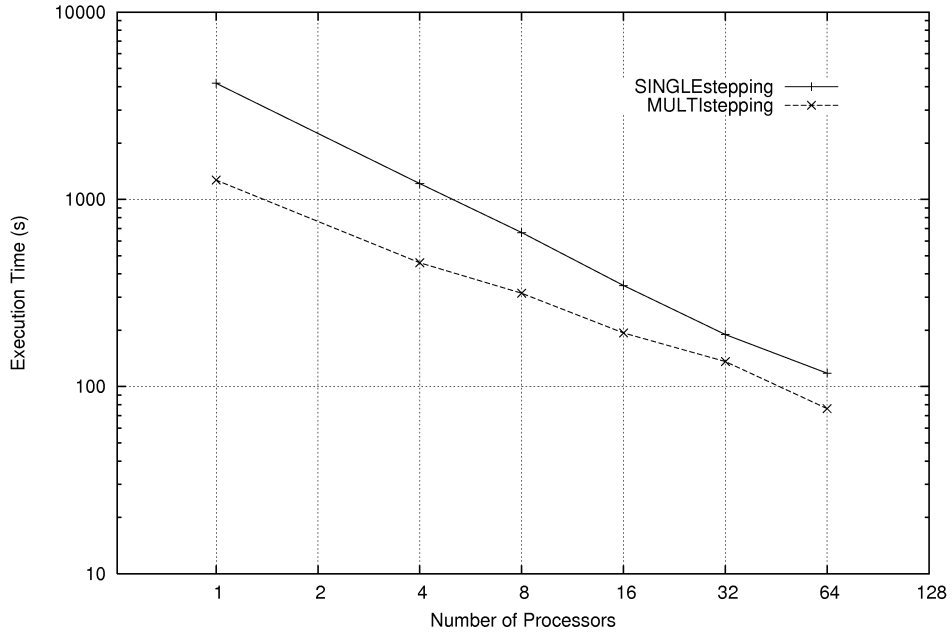


Figure 9: Effects of multistepping on performance of *dwarf* simulations on NCSA's Tungsten

correspond to a factor of 1000 in dynamical times. Indeed, the 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.

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 in each case the number of Tungsten processors used. Figure 9 shows the results of those tests.

It becomes clear from Figure 9 that multistepping can greatly accelerate a simulation. Starting with the

Figure 10: Timeline of singlestepping *dwarf* simulation on 32 Tungsten processors (duration=190 s)

case of one processor, and noticing the logarithmic scales in the plot, there is a gain of more than three times in execution time when multistepping is used. Since this case is a serial execution, the performance gain comes exclusively from the algorithm used for multistepping. This result confirms the importance of the multistepping capability for the computational performance of a cosmological simulator.

Figure 9 indicates that multistepping provides performance gains across the entire range of processors tested. Although the gain with 64 processors is still very significant, the figure reveals a trend of slight decrease in the gain as the number of processors increases. To understand why this happens, we consider in detail the particular executions on 32 processors. Figures 10 and 11 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), and the colors in a given horizontal line represent the utilization of that processor across the execution.

In Figure 10, where singlestepping was used, there are four steps displayed. Figure 11 represents an entire big step, which corresponds to the same cosmological duration of the four steps displayed for singlestepping. In this big step, the execution proceeds across four sub-steps. For each sub-step, only a group of particles is active, according to the selection of the multistepping algorithm. That is the reason for the different

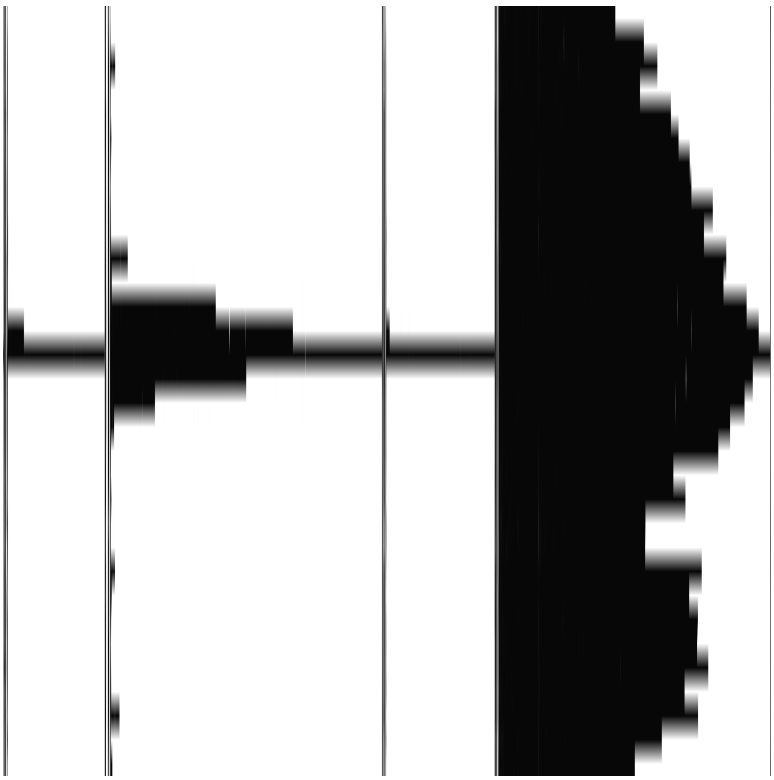


Figure 11: Timeline of multistepping *dwarf* simulation on 32 Tungsten processors (duration=136 s)

durations among the sub-steps in Figure 11. Overall, the total duration of the segment in Figure 10 is approximately 190 seconds; in contrast, the same simulation with multistepping in Figure 11 takes only 136 seconds. However, despite the gain from multistepping, some of the sub-steps in Figure 11 show a much more severe load imbalance. in fact, in the third sub-step of Figure 11 only one processor is 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 also increases, making parallelism less effective than in singlestepping.

5.3 Multistepping-Based Load Balancing

While we could have applied the load balancing techniques described in Subsection 5.1 to optimize the singlestepping execution of Figure 10, applying load balancing to a multistepping execution is much more challenging. As Figure 11 reveals, the fourth sub-step in our multistepping execution is similar to a regular step of singlestepping, because all particles are active. However, from the first to the third sub-step, only a small subset of particles are moving. A load balancer that is effective for the last sub-step may be completely

inadequate for the other sub-steps.

This analysis shows that particle codes with multistepping capabilities may need more advanced load balancers. We are currently developing new load balancers that work in concert with the multistepping scheme of Figure 11, and redistribute particles at sub-step boundaries. Such redistribution can be based on the observed processor loads in the corresponding sub-step of the previous big step in the execution. In addition, we are considering toplology-aware strategies to guide the load redistribution. Those strategies explore the interconnection topologies of the machines, and allow minimization of the communication overhead caused by the particle redistribution.

6 Conclusions and Future Work

We described a highly scalable parallel gravity code ChaNGa, based on the well-known Barnes-Hut algorithm. Based on an adaptive runtime system, ChaNGa decomposes the particles into many *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 the local nodes. Requests for remote data are effectively pipelined to mask any communication latencies. We demonstrated, for the first time, cosmological simulations based on real datasets scaling well to over 8,000 processors, and running with some additional speedups to over 32,000 processors.

We showed that existing dynamic load balancers are not quite adequate for the purpose of cosmological simulation, since they do not deal well with the peculiar nature of communication involved. We demonstrated the good performance of a new dynamic load balancer that works much better than the existing balancing methods. We also showed preliminary results on a multiple-time-stepping scheme that illustrates both its promise in reducing the operation count and execution time, and its dynamic load balancing challenge. We identified the need for a new class of load balancers that can deal with multiple phases of the computation separately.

Currently, our code is being used for gravity computations by scientists. We intend to add hydrodynamics to this code in future. Alternative decomposition schemes for further reducing communication, and runtime optimizations to mitigate the impact of communication costs are also planned.

References

[1] 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*, (Philadelphia, PA), pp. 94–101, December 1996.

[2] L. V. Kale and S. Krishnan, "Charm++: Parallel Programming with Message-Driven Objects," in *Parallel Programming using C++* (G. V. Wilson and P. Lu, eds.), pp. 175–213, MIT Press, 1996.

[3] 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.

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

[5] G. Lake, N. Katz, and T. Quinn, "Cosmological N-body simulation," in *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, (Philadelphia, PA), pp. 307–312, February 1995.

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

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

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

[9] V. Springel, "The cosmological simulation code GADGET-2," *MNRAS*, vol. 364, pp. 1105–1134, 2005.

[10] 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), ACM Press, 1999.

[11] 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), ACM Press, 2006.

[12] 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*, (Baltimore, MD), pp. 1–18, September 2002.

[13] 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.

[14] 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.

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