



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Interaction-Based Load Balancing in N-body Simulations

O. Pearce, T. Gamblin, M. Schulz, B. R. de
Supinski, N. M. Amato

April 25, 2013

Supercomputing
Denver, CO, United States
November 17, 2013 through November 22, 2013

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Interaction-Based Load Balancing in N-body Simulations *

Olga Pearce^{*†}, Todd Gamblin[†], Bronis R. de Supinski[†],
Martin Schulz[†], Nancy M. Amato^{*}

^{*}Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA
{olga,amato}@cse.tamu.edu

[†]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA, USA
{olga,tgamblin,bronis,schulzm}@llnl.gov

ABSTRACT

N-body methods compute interactions between particles. They represent an important class of simulations and are used in a wide range of applications. A rich body of work focuses on scaling N-body methods and typically includes load balancing mechanisms that assign load approximately by assigning particles; we demonstrate that they do not perform well at scale. We propose a decomposition method that directly assigns the computation units in N-body applications, *particle interactions*, to processes. We characterize the distribution of interactions in space and use an adaptive sampling approach to make the interaction-based decomposition accurate and affordable while minimizing the communication required. We implement and evaluate our approach on a Barnes-Hut algorithm and show that our method achieves a significant improvement in load balance and consequently overall performance.

Categories and Subject Descriptors

C.4 [Performance of Systems]: Performance attributes, Modeling techniques; I.6.8 [Simulation and Modeling]: Types of Simulation—*Parallel*; D.1.3 [Programming Techniques]: Concurrent Programming—*Parallel programming*

Keywords

load balance, performance, modeling, simulation, framework

1. INTRODUCTION

Problems in a variety of areas rely on N-body methods, including astrophysics (galaxy formation, large-scale structure [38]), computational biology (protein folding [32]), chemistry (molecular structure, thermodynamics [23]), and molecular dynamics [33]. An N-body application simulates the dynamic evolution of a system of particles or bodies under the influence of physical forces. The simulation computes the forces between the particles, or *particle-particle interactions*. Because the direct-sum algorithm is $O(n^2)$ in the number of particles, many large scale N-body simulations use optimizations that may consider nearby particles individually but treat distant particles as a single large particle; examples include the Barnes-Hut method [4] or multipole expansion [34]. This can dramatically reduce the number of pairwise particle interactions that must be computed.

As problems grow larger, simulations are scaled to larger supercomputers, and load imbalance becomes a larger impediment to performance. While it may be reasonable to leave a handful of cores idle, idling hundreds of thousands of processes results in severe performance degradation, necessitating more precise load balancing mechanisms at scale. Currently, many N-body simulations use domain decomposition to divide the simulated particles between processes; *particle interactions* are then computed by the processes owning the particles. The typically used geometric space decomposition provides an approximation for dividing work; we show that this approximation becomes prohibitively imprecise at scale.

We propose a load balancing that directly balances the computation in the simulation, not the space. *Particle interactions* are the unit of computation in N-body applications, whether computed directly or through approximation methods, and therefore the units that need to be balanced. Current approaches do not balance the particle interactions directly because it would be prohibitively expensive to balance $O(n^2)$ interactions within radius of interaction. However, interaction density is variable and cannot be approximated by the particle density, and physical proximity of the interacting particles is important as each interaction computation requires knowledge of these particles and, in turn, manipulates their positions and other attributes.

Our approach is to use hypergraph partitioning to partition particle interactions while also directly considering particle proximity. We use interaction sampling and nearest-neighbor assignment to control the cost/accuracy tradeoff of our approach through adaptive

^{*}This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-635761).

sample size and refinement.

We evaluate our approach by balancing a Barnes-Hut algorithm and demonstrate significantly improved load balance and therefore overall performance.

This paper makes the following contributions:

1. A new interaction-based decomposition method for N-body simulations, using hypergraph partitioning to assign interactions and particles to processes to balance the load and optimize required communication;
2. Characterization of distribution of interactions in space as power law;
3. A sampling approach exploiting nearest-neighbor assignment to reduce cost;
4. An adaptive (hierarchical) sampling strategy to mitigate the power law distribution of interactions in N-body simulations.

Section 2 summarizes traditional particle-based domain decomposition and load balance correction methods for N-body applications. Section 3 describes our interaction-based domain decomposition and load balancing method. Section 4 describes our adaptive sampling approach to interactions. Section 5 will outline the algorithm for load balancing interactions. We evaluate performance of our approach in Section 6.

2. LIMITATIONS OF CONVENTIONAL N-BODY LOAD BALANCING TECHNIQUES

Traditional domain decomposition methods in N-body applications either distribute particles to processes, or decompose the space geometrically to assign all particles in a geometrical region to a process. Techniques in this area include orthogonal recursive bisection [6, 38], oct-trees [31, 39], and fractiling [3]. Orthogonal recursive bisection applies a binary decomposition recursively on the domain to partition it into rectangles requiring equal computational effort. Oct-tree methods use a spatial tree data structure to represent a system of N bodies in a hierarchical manner, and rely on different methods to determine which nodes of the tree are close enough to interact. Fractiling is a probabilistic analysis-based dynamic scheduling scheme that exploits the self-similarity properties of fractals.

Barnes-Hut [4] simulations [20, 40], a classical example of an N-body algorithm, assigns individual particles to processes; to achieve locality, the particles are partitioned geometrically or sorted by a space-filling curve (e.g., Hilbert [12]) prior to being partitioned. The interactions, the computationally intensive part of any N-body simulation, are computed by the processes owning the particles involved; a tiebreaker is employed to determine the computation owner if the two particles involved in one interaction belong to different processes. Load balance is controlled by moving the particles between processes and with this indirectly affecting where interactions are computed, rather than being able to control interactions directly. Since the particles may move after each decomposition, a new spatial partitioning or sorting of the particles may be required prior to new rebalancing.

Other parallel N-body algorithms use spatial methods to decompose the domain between processes; they assign a space defined

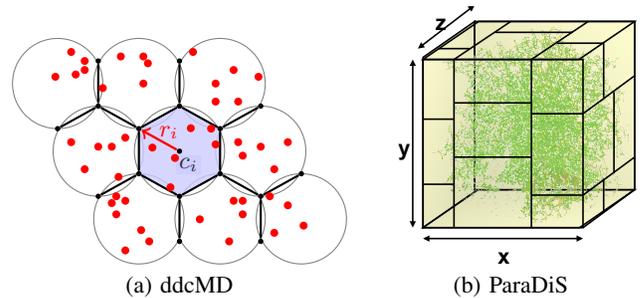


Figure 1: Geometric Particle-Based Domain Decomposition

by a Voronoi cell (e.g., ddcMD [33], Figure 1(a)) or a prism (e.g., ParaDiS [10], Figure 1(b)) to each process. Each process is then responsible for the subset of particles located in that space, and for computing the associated interactions. As with Barnes-Hut, a tie-breaker is used to compute interactions between particles spanning partitions.

Load balancing methods for spatially decomposed N-body simulations adjust the spatial decomposition in the simulation. They estimate the load of a process based on the number of owned particles or timing calipers; process space boundaries are then shifted to give each process less or more work. Algorithms with a geometric decomposition employ many approximations; they can assume uniform work distribution within the space assigned to each process, ignoring features in the simulation that may cause higher density of work in some regions of the simulation space. They typically do not consider individual units of work as they contribute to the load and how the load changes when the boundaries are shifted. Further, they do not take into account which of the interactions are computed directly (using force calculation) vs. which are approximated using a multipole method aggregating distant particles [21], which is determined by proximity of particles in simulation space. Geometric decomposition may also have additional implementation-specific spatial decomposition limitations (the space can only be decomposed into spatial prisms or Voronoi cells), resulting in additional approximations.

In summary, both particle-based and geometric approximations inherently contain inaccuracies in their methods to control and thereby balance the actual computational load. This can lead to significant load imbalances, which will have a drastic impact on the scaling properties of the algorithms.

3. A NEW APPROACH: INTERACTION-BASED DECOMPOSITION

In order to overcome the limitations of traditional particle-based or geometric approaches and to achieve a more precise load balance that will allow us to scale to larger number of nodes efficiently, we need to focus our load balance efforts on the *interaction* between particles, since they are the unit of work in N-body simulations. We therefore propose to assign interactions to processes directly to have full control over the load balance of work units in the application. For correctness of the simulation, interactions must be assigned to processes uniquely (i.e., each interactions must be assigned to exactly one process), so our goal is twofold: devise a method that assigns interactions (1) uniquely and (2) in a balanced manner.

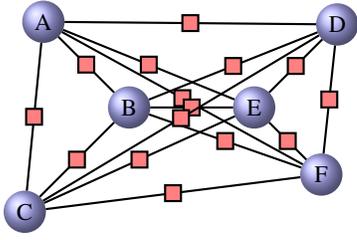


Figure 2: Particles {A–F} Shown as Spheres; n^2 Interactions Shown as Squares

To accomplish this goal we start by representing the computation in N-body simulations as a *hypergraph* with *vertices* representing interactions (work units) and *hyperedges* representing particles (storage units). A hyperedge connects one or more vertices (interactions) that need information about the particle for computation. Figure 2 shows an example of such a hypergraph representing particles as spheres and interactions between them as squares. This turns the load balancing problem into a hypergraph partitioning problem with communication required for all cut hyperedges, i.e., all particles that are required by two or more particles.

Parallel applications frequently utilize *ghost nodes* when a copy of remote data, in our case data of a particle, is needed; ghost nodes require communication to update the ghost node when its original has changed and/or to update the original when the ghost node has been changed (*pre-* and *post-*timestep communication). While the graph model counts each cut edge, the hypergraph model gives a more accurate representation of communication cost (volume) of the ghost nodes by counting a cut hyperedge to several vertices in the same partition as a single cut; this is important for irregular graphs [16] with non-uniform vertex degrees throughout the graph. The hypergraph model works well in our case because the number of interactions a particle is involved in is highly variable in N-body problems, as we will show in Section 4.

As mentioned above, we use hypergraph partitioning to balance the assignment of interactions to processes. Several hypergraph partitioning libraries are available [14, 30]; in this work, we use Zoltan’s hypergraph partitioner [15]. Formally, given a weighted *hypergraph* $H = (V, E^H)$ where V denotes a set of vertices and E^H denotes a set of hyperedges, hypergraph partitioning divides vertices into k sets based on the following two objectives:

1. **equal partitions:** vertices are divided among processes such that the sum of vertex weights on a process is approximately equal for all processes.
2. **minimized hyperedge cut:** minimize the number of ghost nodes required in the application by minimizing edge-cut, the total weight of the edges, i.e., shared particles, cut by the partitions.

After the partitioning process, we compute the assignment of particles to processes by assigning them where majority of their interactions are assigned; this helps minimize the number of ghost nodes required for the particles. Balanced assignment of particles in addition to interactions could be another goal, but the problem of balancing both vertices and hyperedges is a slightly different problem than the standard hypergraph partitioning problem and a good solution is not yet available.

Algorithm 1 Interaction Sampling

Input. $H = (V, E^H)$ (graph of particles and interactions)
1: **for** \forall Particle $e_i \in$ particles **do**
2: $H'.insert(e_i)$
3: numSampled = $\max(1, s \times |V_i|)$, where $V_i \leftarrow$ interactions of e_i
4: **for** $0 \leq j < \text{numSampled}$ **do**
5: $v'_j \leftarrow$ sample $v_j \in V_i$
6: $H'.insert(v'_j)$
7: **end for**
8: **end for**

4. SAMPLING INTERACTION GRAPH

An exact partitioning approach would consider a hypergraph of $O(n)$ particles and $O(n^2)$ interactions within the interaction radius, which would lead a prohibitively large graph both in terms of storage and time to partition. However, considering each individual interaction is unnecessary, as the distribution of computation load is tied directly to the variation in interaction density. We can therefore use a potentially small subset of the data as input to our partitioner, as long as we can maintain the density variations in the simulated space. We achieve this by applying a sampling approach.

In particular, we create a sampled hypergraph $H' = (V', E^{H'})$ as a subgraph of the interaction hypergraph H maintaining the following properties:

1. H' contains all hyperedges (particles) from H , i.e., all particles remain represented ($V' = V$);
2. $\forall h' \in E^{H'}, \exists i$ vertices $v' \in H'$ such that v' is connected to h' , where $i \geq 1$, i.e., the sampled graph contains at least one vertex (interaction) per hyperedge (particle), which ensures preservation of spatial proximity information in the graph;
3. $\forall e' \in E^{H'}, \exists i$ and a vertex $v' \in H'$ such that v' is connected to e' with $i = |v'| = s \times |v|$ (with s being the sampling ratio), i.e., the number of sampled vertices v' is proportional to the number of vertices v connected to h in the original graph; this ensures that the work (interactions) in the application is fairly represented and the partitioner is tasked with partitioning work units of similar granularity.

Algorithm 1 outlines the steps of our sampling approach.

Once we complete the sampled subgraph we use the same hypergraph partitioning approach as described in Section 3 to assign the sampled interactions to processes, followed by a reconstruction of the complete, deterministic assignment of non-sampled interactions. The latter is achieved by assigning each interaction that was not part of the sample to the same process as its nearest sampled interaction. For this to be possible we assign each *interaction* a *coordinate* in space matching the *centroid* of all of the involved (two or more) particles, as shown in Figure 2.

All particle’s sampled interactions are designated as centers of Voronoi cells (shown as black dots in Figure 3(a)). Each sampled interaction is the representative for all the interactions in its neighborhood, defined as the sample’s Voronoi cell; the sample’s weight is the number of interactions the sample represents. Since the *weighted sample* is a vertex in our hypergraph, when a partitioner assigns a weighted sample to a process, all of the interactions in the corresponding Voronoi cell are also assigned to that process. Multiple

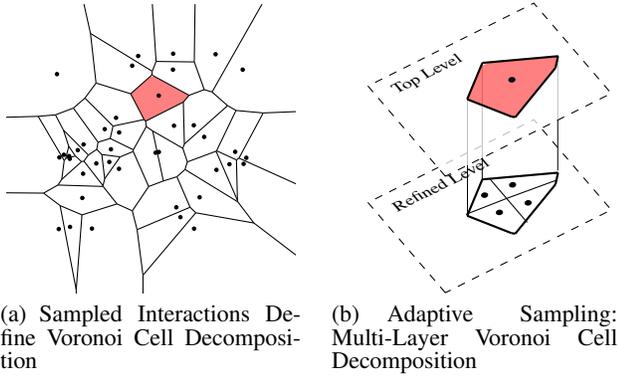


Figure 3: Interaction Sampling Strategy

Voronoi cells can be assigned to each process, and the sample rate becomes a direct method for controlling the accuracy/cost tradeoff of the granularity of the decomposition. To assign a non-sampled interaction to a process, we first find its nearest neighbor among the sampled interactions, and assign the non-sampled interaction to the same process as the sampled one. Note that if a single sample represents the interactions of a particle, all of the particle’s interactions effectively belong to a single Voronoi cell, allowing us to skip the nearest neighbor calculation for this common case.

It has been shown that in complex systems consisting of many interacting elements, the dynamical process results in power law distribution of activity of the elements [17]. Specifically, in N-body simulations, interaction density asymptotically follows a power law distribution. Thus our random sampling leads to Voronoi cells whose density follows a power law distribution and can be fitted to a gamma distribution:

$$\mathbb{P}(x) = \frac{1}{\Theta^k} \frac{1}{\Gamma(k)} x^{k-1} e^{-\frac{x}{\Theta}} \quad (1)$$

where k is a *shape* parameter, Θ is a *scale* parameter, and $\Gamma(k)$ is the *gamma function* evaluated at k . While the parameters of the gamma distribution representing Voronoi cell density (or weights of sampled interactions) vary in the examples we have considered, the important characteristic to note is the thin long tail of the distribution as shown in Figure 4, which implies that a very small number of the samples we take may represent a large portion of the interactions. These samples become the disproportionately heavy vertices in the graph we partition, and are treated as indivisible units of work by the partitioner. This limits the partitioner’s ability to load balance effectively.

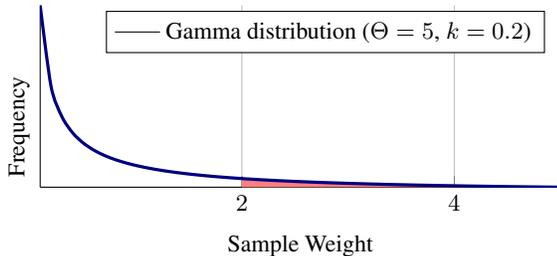


Figure 4: Gamma Distribution Probability Density Function

Algorithm 2 Sampling-based Interaction Load Balancer

$p \leftarrow$ # processes, $n \leftarrow$ # particles, $m \leftarrow$ # interactions,
 $s \leftarrow$ # sampled interactions per particle ($s \approx \frac{m}{n} \times \% \text{ sampled}$),
 $HG \leftarrow$ hypergraph ($V \leftarrow$ interactions, $HE \leftarrow$ particles)

Step	Cost
1: Add particles as hyperedges to hypergraph HG	$O(\frac{n}{p})$
2: Build list of interactions per particle	incurred
3: Sample vertices (interactions) and add to HG	$O(s \frac{n}{p})$
4: Per particle, build kd-tree \leftarrow samples	$O(\frac{n}{p} s \log(s))$
5: Use kd-tree: assign interactions to samples	$O(\frac{m}{p} \log(s))$
6: Partition HG: assign samples to processes	$O(\frac{p}{n} \log \frac{sn}{p})$
7: Redistribute particles, samples, ghosts	incurred
8: Build list of interactions per particle	incurred
9: Per particle, build kd-tree \leftarrow samples	$O(\frac{n}{p} s \log(s))$
10: Use kd-tree: interaction \rightarrow sample \rightarrow process	$O(\frac{m}{p} \log(s))$

This problem can be thought of as under-sampling. However, uniformly increasing the sample rate will not correct the power law distribution of sample weights. Therefore, we apply an adaptive sampling strategy to increase the number of samples in areas with high density. Voronoi cells representing heavy samples are subsequently split into more Voronoi cells, as shown in Figure 3(b). To not affect the Voronoi decomposition of the surrounding cells, we have chosen a layered approach, where only the space in the cell being refined is considered when applying a new Voronoi decomposition.

Note that because the number of heavy samples is small, our adaptive sampling strategy allows us to achieve a more uniform distribution of sample weights while only increasing the number of samples slightly. The (red) shaded area under the tail of the gamma distribution pdf in Figure 4 depicts the number of samples that are larger than 2μ ; the size of samples that should be refined is a parameter in our implementation, and we have studied its impact on the cost/accuracy tradeoff. In Section 6, we will discuss the impact of the sample size and the adaptive sampling strategy on the sample weight distribution and balancing quality.

5. AN INTERACTION-BASED LOAD BALANCER FOR BARNES-HUT

Using the interaction-based decomposition described in Section 3 and the sampling approach described in Section 4, we have a complete approach to load balancing interactions in N-body applications, which we implement in a Barnes-Hut simulation code to demonstrate the efficiency of our techniques. The approach is fully general, though, and could be equally integrated into any other N-body simulation framework.

5.1 Putting all Steps Together

Algorithm 2 outlines our complete approach. First, we add all particles to the hypergraph as hyperedges (line 1), and then build a list of interactions and assign a *coordinate* to each interaction, either by choosing one of the involved particles’s coordinates, or taking the *centroid* of all of the involved particles (two or more particles) (line 2). Second, we take a uniform random sample of these interaction (line 3) and build a nearest-neighbor tree (kd-tree) per particle from the samples (line 4); each sample defines a Voronoi cell. Third, we use the nearest-neighbor tree to count the non-sampled interactions that fall into each Voronoi cell (line 5), adaptively refine the sample if necessary, and add them to the hypergraph, weighted with their corresponding counts. We utilize a partitioner to assign the sampled

interactions to processes in a balanced manner (line 6) and assign particles to processes by majority rule. The application redistributes the particles and the sampled interactions, sets up the ghost nodes (line 7), and builds the interaction list again (line 8). For each particle, we build a nearest-neighbor tree of the local samples (line 9) and use it to assign the non-sampled interactions (line 10).

Table 2 also lists the costs associated with each step. ‘Incurred’ cost means that the application will already do this step regardless of the load balancing approach, the other costs are additional to the application and must be carefully considered. The costs associated with the nearest-neighbor tree and the partitioner are determined by the size and the variability of the sample set, as is the quality of the resulting partitioning.

As we will show in Section 6, the cost of our scheme can be divided into the cost of sampling the graph and computing the sample weights, and the cost of partitioning. It is important to note that the cost of sampling is relative to the number of interactions each particle is involved in; if a particle is only involved in a small number of interactions which are then represented by a single sample, the cost of building the nearest-neighbor tree and classifying the non-sampled interactions is $O(1)$.

We use the nearest-neighbor implementation from CGAL [2] for the nearest neighbor computation; other implementations or range queries are possible alternatives to consider in future work to reduce the cost. Further, when the number of samples in the tree is small, a linear traversal of an array of samples is faster than pointer jumping through the tree, despite being $O(s)$ rather than $O(\log(s))$.

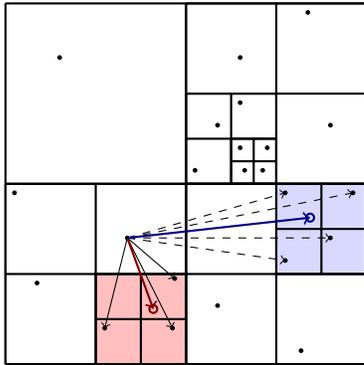


Figure 5: Octree in Barnes-Hut Benchmark

5.2 Interaction Partitioning to Barnes-Hut

We apply our method to a Barnes-Hut simulation code. Barnes-Hut’s force-calculation algorithm uses an octree to approximately compute the force that the n particles in the system have on each other (e.g., through gravity). The n leaves of the octree are the individual particles, while the internal nodes summarize information about the particles contained in the subtree (i.e., combined mass and center of gravity), which effectively partitions the volume hierarchically around the n particles into successively smaller cells. While a precise computation would have to consider $O(n^2)$ interactions, the Barnes-Hut algorithm uses the summary information contained at each level of the hierarchy to approximate interactions for far away particles: particles that interact with other particles in nearby cells are computed directly, while for interactions with

Algorithm 3 Pseudocode for Barnes-Hut

```

Input. particles  $\leftarrow$  /* read input */;
1: for int step = 0; step < maxTimestep; step++ do
2:   Octree octree = new Octree();
3:   for  $\forall$  particle  $p \in$  bodies do
4:     octree.Insert(p);
5:   end for
6:   for  $\forall$  Subtrees  $s \in$  octree do
7:     s.ComputeCombinedMass();
8:     s.ComputeCenterOfGravity();
9:   end for
10:  for  $\forall$  Particle  $p \in$  bodies do
11:    b.ComputeForce(octree);
12:  end for
13:  for  $\forall$  Particle  $p \in$  bodies do
14:    b.Advance();
15:  end for
16: end for

```

Algorithm 4 Pseudocode for Barnes-Hut ComputeForce()

```

Input. Particle p;
1: stack.PushBack(root);
2: while !stack.empty() do
3:   OctreeNode node = stack.PopBack();
4:   if distance(p,node) > threshold /* node is far */ then
5:     ComputeForce(p, node);
6:   else
7:     for  $\forall$  child  $\in$  node.children() do
8:       if child is leaf then
9:         ComputeForce(p, child);
10:      else
11:        stack.PushBack(child);
12:      end if
13:    end for
14:  end if
15: end while

```

cells that are sufficiently far away, it suffices to perform only one force computation with the cell instead of performing one calculation with each body in the cell. Overall, this results in $O(n \log(n))$ complexity. For example, consider the two-dimensional hierarchical subdivision of space in Figure 5. The algorithm will check the distance to the red cell’s center of gravity (red circle); because the distance is not large (red arrow), interactions with all the bodies in the red cell will be computed (black arrows). Because the center of the blue cell (blue circle) is far enough, only the interaction with the center will be computed (blue arrow, a single computation), and the actual bodies will not be considered (dashed arrows).

We created a distributed version of Barnes-Hut [4] based on a shared memory implementation from the Lonestar suite in Galois [11, 28]. Our code is in C++ and communicates with MPI; it allows redistribution between timesteps and enables assignment of each particle to any process. Pseudocode is presented in Algorithm 3. To integrate our approach, we extract the particle interactions from the octree data structure and use it to generate our hypergraph. Once partitioned, we use the result of the hypergraph partitioner to determine the new load distribution.

6. PERFORMANCE EVALUATION

For our experiments, we use a Linux cluster with 1,856 compute nodes, each consisting of two Hex-core Intel Xeon EP X5660 processors running at 2.8 GHz, for a total of twelve cores per node and 22,272 cores total. All nodes are connected by QDR Infiniband. We use gcc 4.4.7 and MVAPICH v0.99 on top of CHAOS [1], an HPC variant of RedHat Enterprise Linux (RHEL), running at Linux

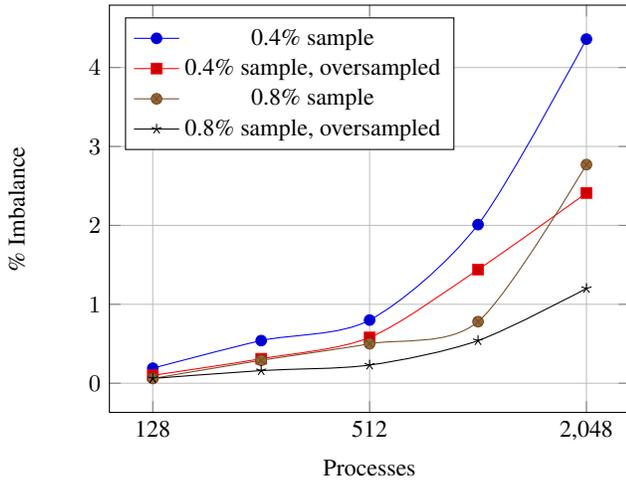


Figure 10: Effect of Interaction Oversampling on Resulting Partitioner Imbalance

kernel version 2.6.32.

All following experiments use a problem with 32K particles, which we strong scale from 8 to 2,048 processes (which represents the largest partition we could run on). We chose this problem since it is the largest problem that can fit into memory for 8 processes and hence provides us with the most realistic input at larger scales. Further, we chose strong scaling, since reproducing density variations for weak scaling is difficult. Only with strong scaling we can guarantee that the problem solved is the same one at any scale and all data points are comparable.

6.1 Sampling and the Power Law Distribution

In this Section, we demonstrate that density variations in N-body problems follow a power law distribution. We show how different approaches to sampling can lead to a more uniform representation of the sampled density, and therefore to better partitions of work.

The number of interactions computed per particle is highly variable, as seen in Figure 6. Additionally, the density of interactions follows a power law, as described in Section 4. If sampled uniformly, the distribution of the number of interactions each sample represents also follows a power law, as shown in Figure 7. This level of variability makes it extremely difficult to partition the samples; intuitively, a sample that represents many interactions will be placed on a single process, and may make that process overloaded.

To mitigate the density properties inherent in N-body simulations, we use a *proportional* per-particle sampling, where each particle is represented by at least one sample to ensure full connectivity in the sampled data, and particle interactions are sampled proportionally to the number of interactions each particle is involved in, as described in Section 4, Algorithm 1. Figure 8 shows that proportional sampling provides a better distribution, but still does not result in a power law distribution of sample weights, as the particles interacting most are now represented by proportionally more sampled interactions.

To compensate, we used a *oversampled proportional* per-particle sampling approach. In it, we took a larger sample and discarded the samples representing too few interactions; the proximity of the

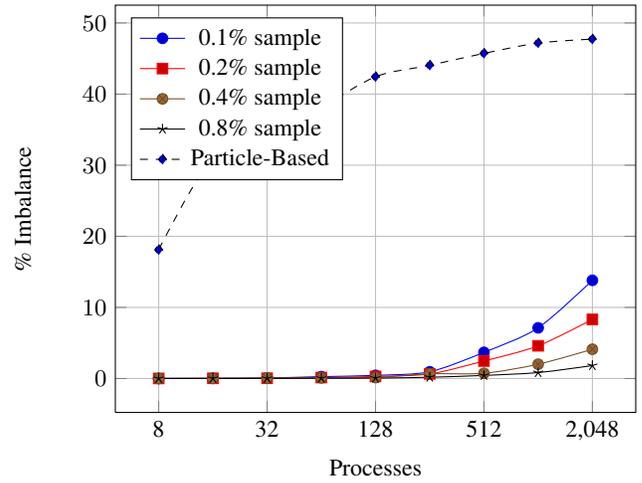


Figure 11: Imbalance (Strong Scaling, 32K Particles)

interactions formerly represented by the discarded samples was recalculated. Figure 9 shows that oversampling and selecting the better samples results in a tighter distribution of how many interactions each sample represents.

Figure 10 compares how the two approaches to sampling impact partition quality; because oversampling changes the distribution of sample weights, it results in better partition quality. Roughly, oversampling can achieve the same partitioning quality as doubling the sample size, and because it keeps the graph size the same, the load balancing cost remains the same.

Additionally, the imbalance goes up with the number of processes in Figure 10. This is due to strong scaling, and the fact that at larger scale each process has fewer interactions to compute and there are fewer samples to partition between processes. Since the number of samples directly impacts the cost of our load balancing method, using a sampling approach that results in more uniform distribution of sample weights becomes of increasing importance on higher process counts.

6.2 Load Balancing Quality

We first evaluate the quality of our load balancing approach for different sampling rates compared to a traditional particle based approach. Figure 11 shows the imbalance, i.e., maximum load minus average load divided by average load, for our experiments. We see that, while imbalance of the application using a particle-based method grows quickly, our direct interaction assignment scheme is able to achieve much lower levels of imbalance. Because our method is sampling based, accuracy becomes a function of number of samples, or the indivisible units assigned to processes. When the sample is too small, quality partitioning is difficult to achieve. However, we can see that even a very modest sample sizes of under 1% of all interactions allows for quality partitions.

6.3 Scaling Behavior of the Load Balancer

Next we study the performance or cost of our balancing approach itself. For this we split the time spent in the load balancer into its three main components, the time needed for the partitioner itself, for the counting of interactions, and the exchange of samples. The speedups of the three phases for various sample rates are shown in Figures 12, 13, and 13 respectively.

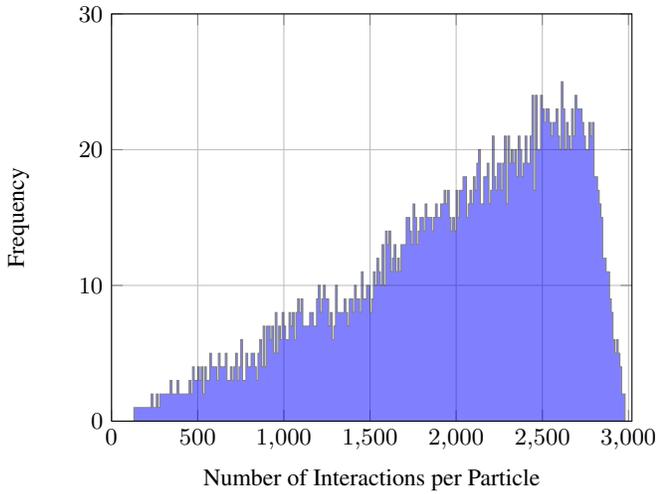


Figure 6: Distribution of Number of Interactions per Particle

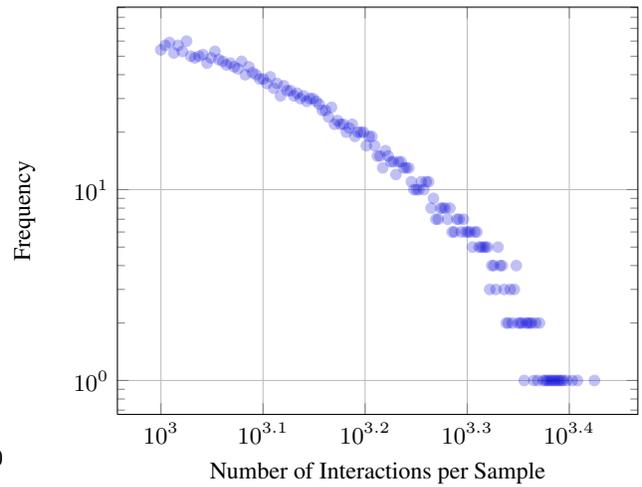


Figure 7: Distribution of Number of Interactions per Sample with Random Sampling

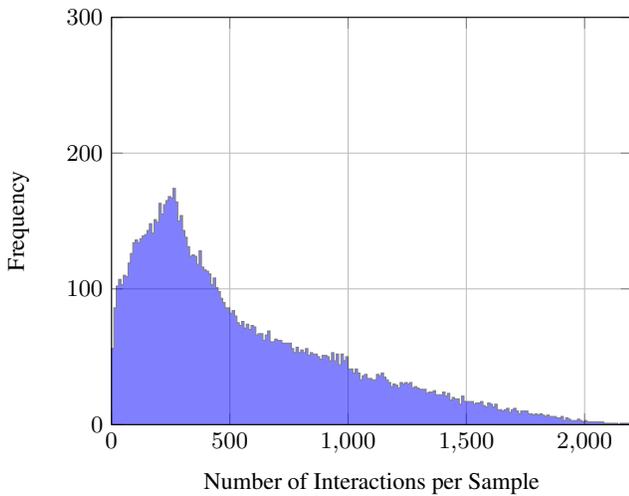


Figure 8: Distribution of Number of Interactions per Sample with Per-Particle Sampling

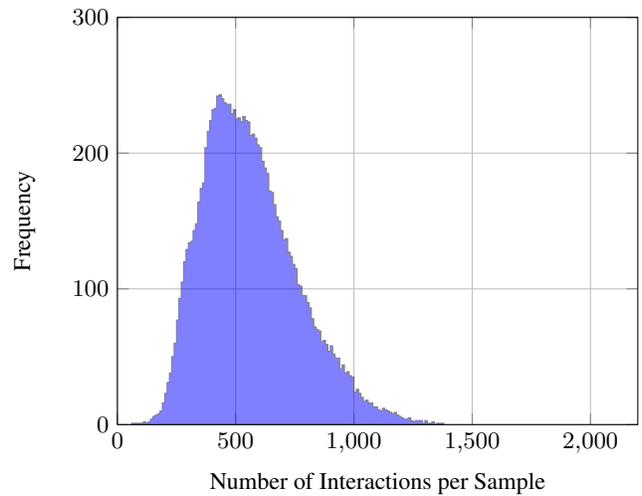


Figure 9: Distribution of Number of Interactions per Sample with Per-Particle Oversampling

Figure 12 shows that the graph partitioner we are using (Zoltan) does not scale well in our strong scaling case because graph partitioning at scale is a challenging problem. This is a well known problem, but can be remedied. Since our method is sampling based and our graph only contains a small portion of the interactions, it is feasible to gather this graph onto a smaller number of processes for partitioning, and then scatter the results. This allows us to pick the optimal partitioner runtime independent of the size of the application scale, resulting in overall reduced runtime. However, this scheme is not implemented, yet, and will be added in our final paper.

The other two phases, which are direct parts of our new algorithm, on the other hand, show almost linear scaling. Figure 13 shows that the nearest-neighbor computation scales, except when the number of samples is so small that there is not much work left per process; this is not a concern as this coincides with a sample too small to achieve accuracy in balancing. Similar, in Figure 14, we see that the

exchange of non-local interactions scales well with only minimal degradations at large scale and for large sample sizes.

6.4 Application Performance

As the final set of experiments, we evaluate the impact of our load balancer onto application performance. Figure 15 shows percent improvement over particle-based balancing, where the times used for our method include the force calculation time as well as the cost of our algorithm. Note that while our method improves the performance in all cases, there is a degradation in percent improvement at scale; this is due mainly to the increasing cost of the partitioner which can be mitigated as previously discussed. Also, the lowest levels of sampling do not offer the balance accuracy to offset the cost of load balancing, while sample rates of 0.2% provides a reasonable overall balance between partitioning cost and load balance.

The total times of the same experiments are shown in Figure 16 as a log-log plot. It shows that our interaction-based balancer with

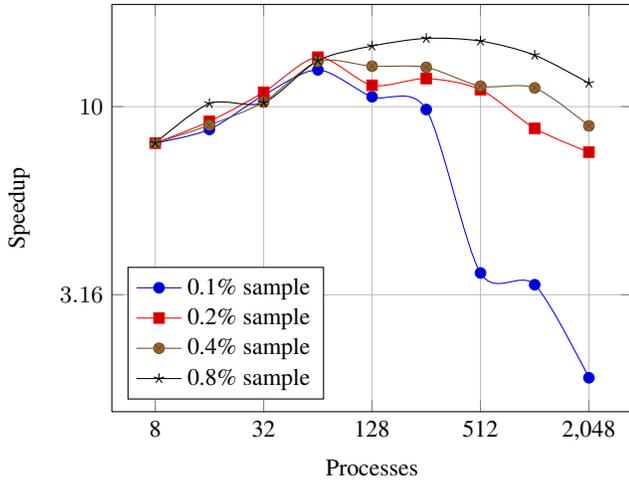


Figure 12: Performance of Hypergraph Partitioner

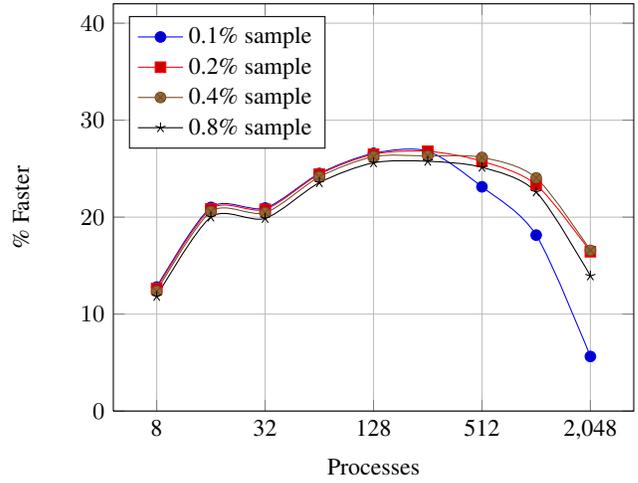


Figure 15: % Faster (Strong Scaling, 32K Particles)

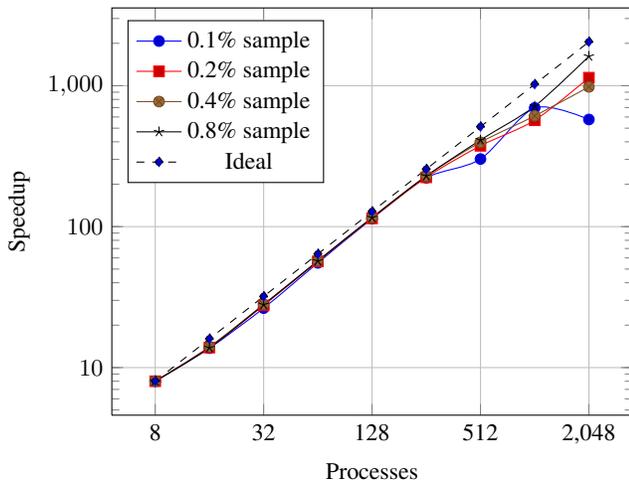


Figure 13: Performance of Interaction Counting

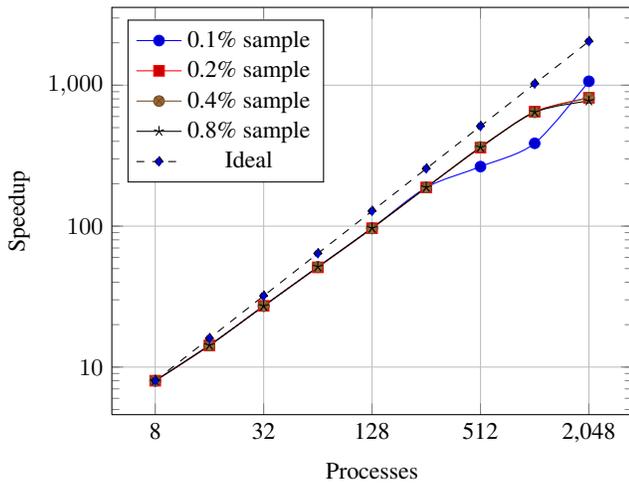


Figure 14: Performance of Sample Exchange

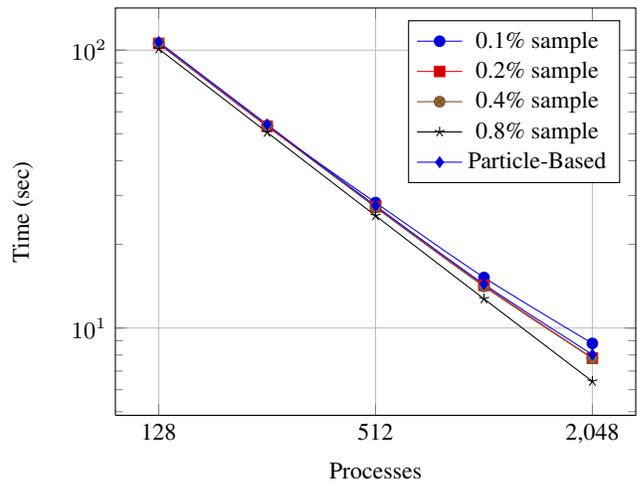


Figure 16: Total Time (Strong Scaling, 32K Particles)

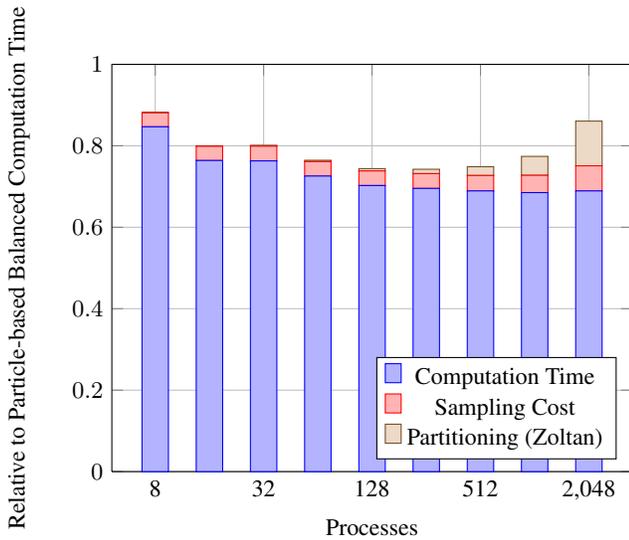


Figure 17: Performance of Computation and Interaction-Based Balancer Relative to Particle-Based Balanced Computation Using 0.8% Sample

sufficient sampling performs the best and clearly outperforms the particle-based balancer.

Figure 17 shows the cost of the interaction-based method and the computation time as compared to the particle-based-balanced computation time. Note that the cost of our algorithm stays relatively flat, while the increase is due to the graph partitioner. Also not that even more improvement of computation time is achievable as we scale up.

Finally we show the aggregated runtime across all processors split into compute time (Figure 18) and overhead of our load balancer 19 (same size bars show same overall compute time across all processes and hence perfect scaling). The overhead numbers show again the limited scalability of the partitioner itself, especially at larger sampling sizes, which we can eliminate as discussed above. For the compute times, however, we see that our load balancer achieves an almost even load distribution across all scales, resulting in only minimal increases in compute times at larger scales.

7. RELATED WORK

We have discussed related work on traditional N-body applications in Section 2. In addition, work related to our approach includes applications and frameworks using geometric and graph-based load balancing methods, partitioners, and work-stealing.

Other types of scientific applications employ geometric load balancers. SAMRAI [41] is a structured AMR application that orders boxes according to their spatial location by placing a Morton space filling curve [25] through the box centroids to increase the likelihood that the neighboring patches will reside on the same processor after load balancing; a geometric balancer in this case makes sense since the boxes are the work units to be explicitly balanced. PLUM [8, 26, 27] is a load balancing framework for adaptive grid applications; it is capable of using any partitioning algorithms and assists in efficient processor assignment and remapping of the computation. DRAMA [5] is a dynamic load balancing library for finite

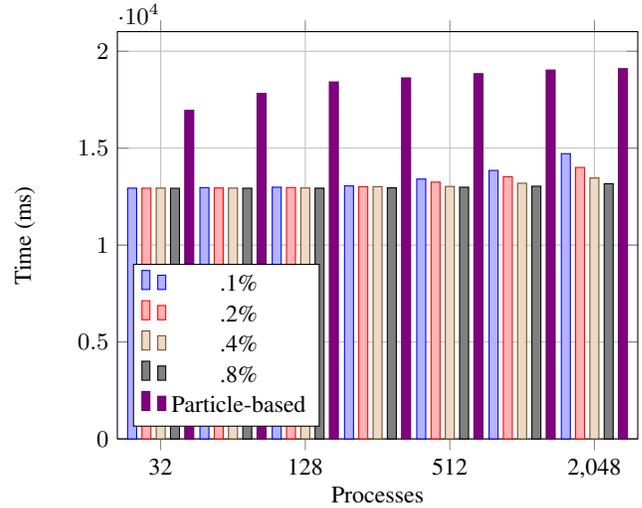


Figure 18: Aggregate Computation Time

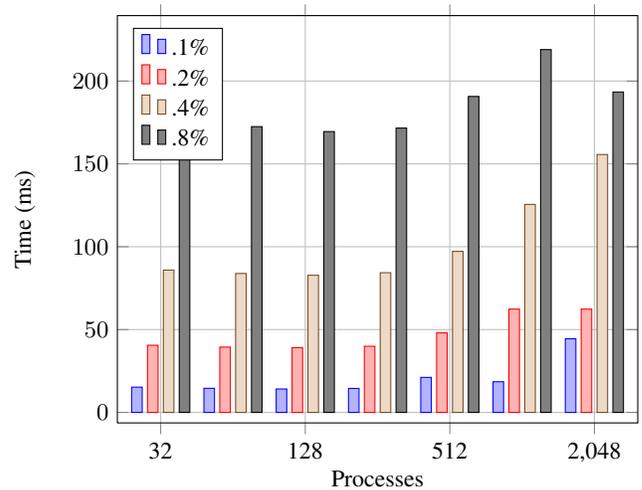


Figure 19: Aggregate Cost of Our Algorithm

element methods that includes geometric and graph partitioning algorithms. Its repartitioning modules include iterative pairwise load balancing, recursive coordinate bisection (RCB), and using ParMetis and Jostle underneath. Since DRAMA specializes on finite element methods, it includes cost functions to account for work and communication associated specifically with elements and nodes in the finite element mesh.

An alternative to geometric methods are partitioners that work with mesh or graph representation of computation, i.e., ParMetis [29, 30], Jostle [35, 36, 37], and Zoltan [14, 15], Zoltan also includes a suite of geometric and graph partitioning algorithms, such as recursive coordinate bisection, recursive inertial bisection, refinement tree-based partitioning, oct-tree partitioning, ParMetis and Jostle. In the graph representation, graph vertices represent computation while graph edges represent communication. In a hypergraph, vertices represent computation and hyperedges represent communication. While the tools in these libraries are powerful, a key responsibility placed on the user is coming up with the appropriate representation for their application to allow the partitioners to balance the units while being aware of the constraints. Hypergraph partitioning can be used for solving many balancing problems, i.e., LU factorization [22].

Overdecomposition and work-stealing are yet another way to load balance scientific applications. Charm++ [7, 9] requires the programmer to express the decomposition (of data and work) into a large number of *objects* and assigns tasks to processes. Charm++ migrates tasks between processor queues based on runtime measurements and a suite of greedy strategies, refinement strategies, and graph-based strategies. ADLB [24] is an Asynchronous Dynamic Load Balancing software library designed for instantaneous load balancing via work-stealing. Similarly, Cilk [13, 18, 19] implements work stealing in its runtime system for task load balancing. Overdecomposition, scheduling and work-stealing work only for the types of applications where the computation can be decomposed into independent objects; since many of the applications do not work this way, a more general approach is necessary.

8. CONCLUSION

We have developed a methodology for explicitly balancing interactions in N-body applications. We characterized the distribution of interactions in N-body applications as power law, and presented a sampling approach that allows a deterministic representation of a large number of interactions both accurately and affordably. We applied hypergraph partitioning to achieve accuracy in load balancing while minimizing the communication required in the application. We evaluated our approach on a Barnes-Hut algorithm and showed significant performance improvement.

9. REFERENCES

- [1] chaos-release: Linux distribution for high performance computing. http://code.google.com/p/chaos-release/wiki/CHAOS_Description.
- [2] CGAL, Computational Geometry Algorithms Library. <http://www.cgal.org>.
- [3] I. Banicescu and S. Flynn Hummel. Balancing processor loads and exploiting data locality in N-body simulations. In *ACM/IEEE Conf. on Supercomputing*, 1995.
- [4] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, 1986.
- [5] A. Basermann, J. Clinckemaillie, T. Coupez, J. Fingberg, H. Dignonnet, R. Ducloux, J. M. Gratiem, U. Hartmann, G. Lonsdale, B. Maerten, D. Roose, and C. Walshaw. Dynamic load-balancing of finite element applications with the DRAMA library. *Applied Mathematical Modelling*, 25(2), 2000.
- [6] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, 1987.
- [7] A. Bhatel , L. V. Kal , and S. Kumar. Dynamic topology aware load balancing algorithms for MD applications. In *ACM SIGARCH Intl. Conf. on Supercomputing*, 2009.
- [8] R. Biswas, L. Oliker, S. K. Das, and D. Harvey. Portable parallel programming for the dynamic load balancing of unstructured grid applications. In *IEEE Intl. Symposium on Parallel Processing*, volume 0, 1999.
- [9] R. K. Brunner and L. V. Kal . Handling application-induced load imbalance using parallel objects. *Par. and Distr. Comp. for Symbolic and Irregular Applications*, 2000.
- [10] V. Bulatov, W. Cai, J. Fier, M. Hiratani, G. Hommes, T. Pierce, M. Tang, M. Rhee, K. Yates, and T. Arsenlis. Scalable line dynamics in ParaDiS. In *Supercomp.*, 2004.
- [11] M. Burtscher and K. Pingali. An efficient CUDA implementation of the tree-based Barnes Hut n-body algorithm. In *GPU Computing Gems Emerald Edition*, 2011.
- [12] A. R. Butz. Convergence with Hilbert’s space filling curve. *Journ. of Computer & System Sciences*, 3(2):128–146, 1969.
- [13] R. B. Christopher, C. F. Joerg, B. C. Kuzmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, 1995.
- [14] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, J. Teresco, J. Faik, J. Flaherty, and L. Gervasio. New challenges in dynamic load balancing. *Applied Numerical Mathematics*, 52(2-3), 2005.
- [15] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan. Design of dynamic load-balancing tools for parallel applications. In *ACM SIGARCH Intl. Conf. on Supercomputing*, 2000.
- [16] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and  . V.  ataly rek. Parallel hypergraph partitioning for scientific computing. In *IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
- [17] Z. Eisler, I. Bartos, and J. Kertesz. Fluctuation scaling in complex systems: Taylor’s law and beyond. *Advances in Physics*, 57(1):89–142, 2008.
- [18] M. Frigo, P. Halpern, C. E. Leiserson, and S. Lewin-Berlin. Reducers and other cilk++ hyperobjects. In *ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA ’09, 2009.
- [19] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation*, volume 33, 1998.
- [20] P. Gibbon, R. Speck, A. Karmakar, L. Arnold, W. Frings, B. Berberich, D. Reiter, and M. Mařandek. Progress in mesh-free plasma simulation with parallel tree codes. *IEEE Transactions on Plasma Science*, 38(9):2367–2376, 2010.
- [21] L. Greengard and V. Rokhlin. A Fast Algorithm for Particle Simulations. *Journal of Computational Physics*, 135:280–292, 1987.
- [22] L. Grigori, E. G. Boman, S. Donfack, and T. A. Davis.

- Hypergraph-based unsymmetric nested dissection ordering for sparse lu factorization. *SIAM J. Scientific Computing*, 32(6):3426–3446, 2010.
- [23] N. Komatsu, T. Kiwata, and S. Kimura. Thermodynamic properties of an evaporation process in self-gravitating n -body systems. *Phys. Rev. E*, 82, Aug 2010.
- [24] E. L. Lusk, S. C. Pieper, and R. M. Butler. More scalability, less pain: A simple programming model and its implementation for extreme computing. <http://www.cs.mtsu.edu/~rbutler/adlb/>.
- [25] G. Morton. A computer oriented geodetic data base and a new technique in file sequencing. *IBM tech report*, 1966.
- [26] L. Oliker and R. Biswas. Efficient load balancing and data remapping for adaptive grid calculations. In *ACM Symposium on Parallel Algorithms and Architectures*, 1997.
- [27] L. Oliker and R. Biswas. PLUM: parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distr. Computing*, 1998.
- [28] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. Méndez-Lojo, D. Proutzos, and X. Sui. The tao of parallelism in algorithms. In *Programming Language Design and Implementation (PLDI)*, 2011.
- [29] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel algorithms for multi-constraint graph partitioning. In *Intl. Euro-Par Conf. on Parallel Processing*, 2000.
- [30] K. Schloegel, G. Karypis, and V. Kumar. A unified algorithm for load-balancing adaptive scientific simulations. In *ACM/IEEE Conf. on Supercomputing*, 2000.
- [31] J. P. Singh, C. Holt, J. L. Hennessy, and A. Gupta. A parallel adaptive fast multipole method. In *Supercomputing*, 1993.
- [32] C. D. Snow, E. J. Sorin, Y. M. Rhee, and V. S. Pande. How well can simulation predict protein folding kinetics and thermodynamics? volume 34, pages 43–69, 2005.
- [33] F. Streitz, J. Glosli, M. Patel, B. Chan, R. Yates, B. de Supinski, J. Sexton, and J. Gunnels. Simulating solidification in metals at high pressure: The drive to petascale computing. *J. of Physics: Conf. Series*, 46, 2006.
- [34] K. S. Thorne. Multipole expansions of gravitational radiation. *Reviews of Modern Physics*, 52:299–340, 1980.
- [35] C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *Parallel Computing*, 2000.
- [36] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2), 1997.
- [37] C. Walshaw, M. Cross, and K. McManus. Multiphase mesh partitioning. *Applied Mathematical Modelling*, 25(2), 2000.
- [38] M. S. Warren and J. K. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. In *Conf. on Supercomputing*, 1992.
- [39] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree N-body algorithm. In *Conf. on Supercomputing*, 1993.
- [40] M. Winkel, R. Speck, H. Hübner, L. Arnold, R. Krause, and P. Gibbon. A massively parallel, multi-disciplinary Barnes-Hut tree code for extreme-scale N-body simulations. *Computer Physics Communications*, 183(4):880–889, 2012.
- [41] A. M. Wissink, D. Hysom, and R. D. Hornung. Enhancing scalability of parallel structured AMR calculations. In *ACM SIGARCH Intl. Conf. on Supercomputing*, 2003.