2016 IEEE 23rd International Conference on High Performance Computing Workshops

Towards Understanding Optimal Load-Balancing of Heterogeneous Short-Range Molecular Dynamics

Steffen Hirschmann, Dirk Pflüger

Institute for Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
{*Steffen.Hirschmann, Dirk.Pflueger*}*@ipvs.uni-stuttgart.de*

Colin W. Glass

High Performance Computing Center Stuttgart
University of Stuttgart
Stuttgart, Germany
glass@hlrs.de

***Abstract*—For heterogeneous dynamic short-range molecular dynamics simulations it is critical to employ suitable load-balancing methods to minimize the time to solution. However, designing, selecting and parametrizing the optimal load-balancing method is a complex task which depends on detailed properties of the simulation scenario. The main challenge in balancing the load of molecular dynamics simulations is the extreme difference in load for scenarios with a heterogeneous particle density, which can easily reach 4-6 orders of magnitude. Therefore, heterogeneity is deemed to be a relevant property.**

In this paper, we formulate a suitable metric to reliably quantify heterogeneity. We apply this metric, which is based on the binning of particles and the evaluation of statistical moments, to example scenarios, and we correlate the results to the performance of five load balancing methods. Furthermore, we identify the load dynamics as a second relevant property. It quantifies how rapidly the load varies over time, and we introduce corresponding metrics. We show that the load dynamics can be used to determine how long the benefits of a specific partitioning are expected to last.

The results indicate that these metrics are useful to differentiate between scenarios, and that they facilitate reasoning over the complex relationship between particle simulation scenarios and optimal load balancing methods. This work is a first step towards understanding this relationship, and it introduces key concepts that we regard as crucial in this process.

***Keywords*-short-range molecular dynamics, parallelization, spatial decomposition, distributed memory, linked-cell, load-balancing, heterogeneity, load dynamics**

I. Introduction

Molecular dynamics (MD) [1]–[3] is an important method of simulation for many fields. We use short-range MD in conjunction with dynamic binding of particles to form agglomerates [4]. These models are used, e.g., to study soot particles [5]. The force evaluation for short-range potential fields can be reduced from $\mathcal{O}(n^2)$ for the naive method to $\mathcal{O}(n)$ for the so-called linked-cell method [6], [7]. Motivated by the massively parallel architectures of current and future high performance computers, MD simulations can be parallelized for scalability via spatial decomposition [8], [9].

However, in heterogeneous MD scenarios one needs to explicitly adapt this decomposition to minimize the time to solution. This process is called load-balancing—finding an optimal partitioning to fully balance the load between all available distributed resources—and is NP-hard [10]. Faced with this complexity class, various methods based on approximation algorithms have been proposed. We present five of them and empirically investigate their suitability for heterogeneous agglomeration scenarios in detail. We do this not by implementing them in a particular MD application but rather "offline" using a load model to quantify the quality of a partitioning.

Our main contribution is two-fold: First, we use an example scenario and different load-balancing methods to show that finding a good partitioning depends on the specific simulation scenario. Second, we propose metrics to evaluate the heterogeneity and the load dynamics of a scenario based on a binned particle distribution.

The paper is organized as follows: In Section II we report on work of other research groups on load-balancing, modelling and the comparison of different methods. In Section III we present a simple load model to quantify the effects of load-balancing. In Section IV we present the scenarios we use for evaluation and the main properties of MD scenarios, namely heterogeneity and load dynamics. In Section V we propose metrics to quantify these properties. In Section VI we present five different load-balancing methods and some enhancements which we propose for them. In Section VII we evaluate the metrics and the load-balancing methods on one sample scenario and show the resulting speedups of load-balancing. Finally, in Section VIII we summarize our work and conclude with a note on further research topics.

II. Related Work

Load-balancing methods have been subject to intensive studies. These include load-balancers based on space-filling curves (SFC) [11], [12], which are used, e.g., by Xiaolin and Zeyao [13]. They order cells along an SFC and use

978-1-5090-5773-3/16 $31.00 © 2016 IEEE
DOI 10.1109/HiPCW.2016.24

a 1D bisection scheme to partition the curve. Bisection can also be applied without an SFC. Recursively bisecting a domain is known as the orthogonal recursive bisection algorithm (ORB) [14]. This is used in the software packages ls1-mardyn [15] and NAMD [9]. The latter even allows to distribute the workload with respect to ghost layer sizes [16] or the process topology [17]. Simon and Teng [18] discuss in general the quality of ORB for solving p-way partitioning problems. Fleissner and Eberhard [19] present an ORB approach which can adapt the partitioning in a local manner. GROMACS [20] uses another sectioning approach. It staggers the dimensions and does all necessary sections in one dimension at once. Nakano and Campbell [21] present a decomposition based on a uniform grid in curvilinear coordinates and simulated annealing. Another well-known partitioning algorithm is graph partitioning. General information can be found in the work of Bichot and Sarry [22]. Hendrickson and Kolda [23] describe the problem of modeling communication cost using graph partitioning. Catalyurek et al. [24] describe a repartitioning model based on hypergraphs with fixed vertices. A diffusive load-balancing scheme is described in [25], [26]. Deng et al. [27] show its relation to the iterative solution of a Poisson equation in two dimensions. Hu and Blake [28] approach the problem more generally formulating the process of optimally migrating load as a quadratic program. Another local method that uses a grid to determine the partitioning is proposed by Deng et al. [29], [30]. We call it the grid-based scheme. Begau and Sutman [31] extend this scheme to three dimensions and report on its implementation in IMD [32].

Comparisons of different load-balancers can, e.g., be found in [33] and [34]. Both compare graph partitioning to SFCs. The main message is that graph partitioning gives a better partitioning quality, while SFCs are faster and consume less memory. A comparison including diffusive methods can be found in [26]. Hendrickson and Devine [23] review several static and dynamic load-balancing algorithms for their quality, memory usage, speed, etc. Particularly for MD, Buchholz [35] compares several different partitioners and describes their implementation in ls1-mardyn. Buchholz also presents a heuristic cost-model similar to the one used in this work. He uses it to estimate the cost as input to load-balancers especially to those that cannot rely on measurements of execution time. Nyland et al. [36] model the cost of an MD application using the Bulk Synchronous Parallel (BSP) model [37] in order to see how parallel performance is affected by computation and communication in their simulations. They conclude that for small simulations a good decomposition of the workload is far more important than communication cost. For a general overview over the BSP model and how to combine different cost terms, see [38].

Similar to Buchholz we use the cost-model as a cost estimator for load-balancing methods. But in contrast to Buchholz and Nyland et al., our main focus is on the theoretical assessment of to properties of different simulation scenarios. It is based on the distribution of cost across the linked-cells and prepares the path for a general classification of load balancing methods with respect to classes of simulation scenarios.

III. Modelling a Short-range MD Application

To quantify the effects of load-balancing one typically simply measures the simulation time. However, to study different combinations of load-balancing methods and scenarios, we choose to model the load using a computation and a communication term. This facilitates collecting data significantly and at the same time renders us independent from a specific implementation of MD.

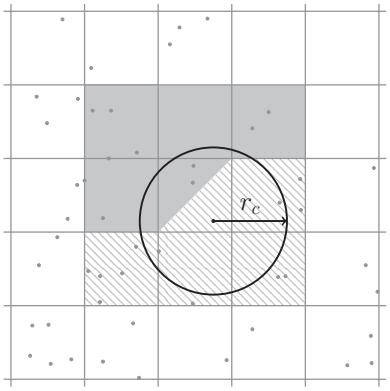
Typically, the overwhelming amount of work in a short-range MD simulation is force calculation. Therefore, we limit the modelling to this inner-most part. As mentioned before, we assume that the linked-cell algorithm is used. In addition to computational cost, there are communication costs related to so-called ghost layer particles. These are necessary to compute interactions between particles residing in different partitions. So, for a process p we have

$$cost(p) = compcost(p) + commcost(p).$$

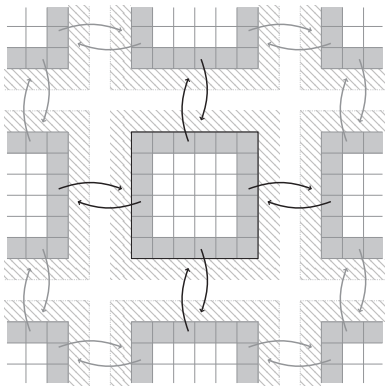
Since we seek to partition the discrete set of all linked-cells it is natural to quantify computational cost on the level of a single cell. The linked-cell algorithm can be described as follows: The particles are sorted into cells of size r_c, which is exactly the cutoff radius for the short-range forces. During the force calculation only particles from neighboring cells are considered, see Figure 1(a). According to measurements in [35] with argon and carbon dioxide, neglecting the actual number of force calculations in the cost term induces only an negligible error in the partitioning. Therefore, we base the computational cost model on distance calculations only. The considered cell neighborhood can be limited further to the *half-shell neighborhood*, using Newton's third law. We do not apply Newton's third law at ghost cells since this would require an additional communication phase to pass back forces. We calculate cross-boundary interactions on both involved processes.

$$cc(i) = \begin{cases} icc(i), & \text{if } i \text{ is an inner cell} \\ icc(i) + bcost(i), & \text{if } i \text{ is a boundary cell,} \end{cases}$$

where $icc(i) = \alpha\, dpairs(i)$ is the cost of an inner cell with $dpairs$ being the number of distance pairs in the half-shell neighborhood. $bcost$ is defined like icc but sums over cross-boundary pairs which are not included in the half-shell neighborhood. Finally, the computational cost term is given by summing over all local cells i: $compcost(i) = \sum_i cc(i)$. The parameter α is the time it takes to calculate a single distance. It depends on the machine and the intermolecular potential. If one was to also consider the cost of force pairs, they could simply be included as another term in $icc(i)$.



(a) A particle with its cell and the neighborhood in the linked-cell algorithm. To find interaction partners, one searches in the gray area (half-shell neighborhood) and writes calculated forces also back to its interaction partners. This saves interaction calculation in the hatched area.



(b) Ghost layer exchange for a spatial decomposition in 2D. In gray the boundary layer which has to be sent to the ghost layers (hatched) of all neighbors. The communication is split into two phases: Sending north and south and then sending east and west.

Figure 1. Basic algorithms comprising the cost: On the left the linked-cell algorithm and on the right the ghost layer exchange for a spatial decomposition.

In order to calculate the forces on boundary particles a subdomain needs information about the boundary particles of neighbors. Exchanging this information is called *ghost layer exchange*. To reduce the number of neighbors to communicate with, we employ the staggered communication scheme of Plimpton [8], see Figure 1(b). Since each process needs to send its boundary layer and receive the neighboring one, we model the cost as

$$commcost(p) = \gamma \left(Npart_{gl} + Npart_{bndry} \right),$$

where $Npart_{gl}$ and $Npart_{bndry}$ are the numbers of particles in the ghost layer and the boundary cell layer of process p, respectively. The parameter γ is the time it takes to communicate one particle and is machine-dependent. We measured the typical cost for a 2D ghost-layer exchange on "Hazel Hen" at the High Performance Computing Center Stuttgart, using one process per node, averaging over different numbers of total processes and each of these over 100 communication steps. In each communication we sent at least 8192 bytes which is the default limit to switch to the Rendezvous protocol. These results were again averaged over ten different node allocations. Finally, we did a linear regression and obtained $\gamma \approx 1.6 \cdot 10^{-8}\,\mathrm{s}$. The linked-cell calculation of distance and force takes approximately $1.3 \cdot 10^{-8}\,\mathrm{s}$ per pair. This number was measured on a local machine with an Intel Xeon E7-8880v3. While the computation parameter will stay the same, the communication parameter will increase when going to 3D.

IV. Scenario Properties

We base this investigation on three example scenarios. In the main scenario, we simulate 10,000 particles as a two-dimensional Lennard-Jones fluid using ESPResSo [39] with dynamic bonding in a setup similar to [5]. However, we use ESPResSo's three particle bond feature to bind particles together via three pairwise harmonic and angular bonds. We also increase the volume density to 0.16 in order to get larger clusters. (Later on, Figure 3(a) will plot an exemplary particle distribution at timestep 10^6.) Most agglomerates have a fractal dimension Df [40] of 1.5 to 1.8. Over time, agglomerates tend towards higher Df. A fractal dimension of 2 means that the agglomerate is disc-shaped whereas perfect chains have a value of 1. On a high level, the scenario basically consists of two parts: In the first part, medium- to large-sized agglomerated form. Here, a significant speedup drop happens. The magnitude and the length of time of the drop depend on the scenario and the partitioning. In the second part, large agglomerates move around due to Brownian motion.

The second scenario we investigate is nucleation. It has the same configuration as the agglomeration scenario but without dynamic binding. The third one is an artificial scenario where only movement occurs: particles are ordered in a grid in a square block and move with the same speed in one direction. To classify the scenarios we define two main properties: heterogeneity and load dynamics.

A. Heterogeneity

The distribution of particles, i.e. the density, varies heavily in all scenarios. They are *heterogeneous*. We have presented that the cost of calculating one particle depends on the number of neighbor particles within a fixed neighborhood (compare Figure 1(a) once more). Therefore, it scales with the local density ρ. The cost of calculating all particles in an area therefore scales with ρ^2. This fact explains why the speedup will degrade for the example scenario and it points

towards the challenge of load balancing heterogeneous MD simulations. As an example, consider a fluid droplet in coexisting vapour: the density of a fluid is typically two to three orders of magnitude higher than the density of the vapour. Therefore the load per volume is four to six orders of magnitude higher.

B. Load Dynamics

Load dynamics is the second important property of a simulation scenario. With the term "load dynamics" we refer to the change of load over time. Such a change can have several causes, mainly the clustering of particles and the movement of particle clusters. Note, however, that in a completely homogeneous particle distribution, Brownian motion may actually not cause any load dynamics as referred to here. This definition of dynamics has the advantage that it directly correlates to the need for rebalancing the load.

C. Exemplary reasoning

Load balancing scenarios with a high degree of heterogeneity will most likely lead to subdomains of unequal sizes, i.e., small subdomains in regions with high cost density and large subdomains elsewhere. However, large subdomains can lead to a rapid deterioration of the load distribution for dynamic scenarios, because load can migrate into them. The rate at which load will migrate depends on the load dynamics of the scenario and on the surface area of the subdomain. Therefore, with increasing load dynamics, the maximum size and surface of subdomains should decrease. This redefines what is to be considered the optimal load balancing.

Even from the simple reasoning above it quickly becomes obvious that optimal load balancing of specific scenarios is a highly complex task. And, as we will see in Section VI, there are plenty of load-balancing methods available which can be further adapted or hybridized and on top of that have various parameters to tune. Our ansatz to cope with this complexity hinges on a crucial component: we introduce metrics to quantify the heterogeneity and load dynamics of scenarios and thereby render them classifiable. We present these metrics in the following section.

V. Metrics

In this section we present metrics for quantifying the main properties of a simulation scenario: heterogeneity and load dynamics. The first one provides a statement about the distribution of load and is based on moments of this distribution. The latter one provides a statement about the movement or migration of load over time. It is based on the difference in load between timesteps.

A. Heterogeneity

We first bin particles into cells corresponding to the cell grid of the linked-cell algorithm and calculate the full-shell load for each of the cells. Then, we successively coarsen the cell grid by summing over neighboring cells. We use a 2×2 neighborhood in 2D. This builds a hierarchy of grids with associated load values in each grid cell. We regard these cells as box-sized subdomains. The subdomain's size corresponds to the number of cells of the original grid which have been summed together. The load values serve as a discrete distribution, which we normalize by its average. We call a cost value normalized by the average an *imbalance*. Then, we take standardized moments of the distributions. These are variance, skewness and excess kurtosis.

The variance is the primary indicator for load heterogeneity as its statement is how strongly a distribution deviates from its mean. Empirically investigated over different scenarios, we find that a variance of over 0.05 is already problematic, as the corresponding speedups in most cases are down by 20–50%. A variance of 0.05 means that the standard deviation σ is approximately 0.22. Assuming a standard normal distribution this means that roughly 17 % of all cells on a certain level have a load of at least the average plus 22 %. However, the higher moments show us that in heterogeneous scenarios we do not deal with normal distributions. This motivates the second indicator of heterogeneity: Positive skewness. We interpret positive skewness as a few cells having a lot of load while the bulk of cells is cheaper. How much more expensive these cells are is indicated by the value of the excess kurtosis. If it is positive the "tails" of the distribution become fatter, i.e. more likely. In the example agglomeration scenario, the actual amount of cells which are more expensive than the average plus 1σ goes up to over 25 % at the skewness peak. Figure 2 shows this for a homogeneous example, the agglomeration scenario at MD timestep 10^7 and the "inverse" of it, meaning the density is inverted by subtracting it from the maximum density. This could correspond to cavitation: Gaseous regions in a fluid. For each scenario the load imbalance distribution on the linked-cells grid is shown and the imbalance metric for different coarsened versions of this grid. We see that the load distribution of the agglomeration scenario is right-skewed with positive excess kurtosis while having a large variance. We regard this scenario as hard to balance because the bulk of cells is cheap. As expected, the variance decreases with growing subdomain sizes. This is typical for particle clustering. Conversely, the inverse scenario is not inhomogeneous in its load at all—while still being inhomogeneous in its particle distribution—because the bulk of cells is expensive and, therefore, also the average load. We regard the latter scenario, as the variance suggests, as easier to balance than the agglomeration scenario.

B. Load Dynamics

To quantify the movement of load we have to look at multiple timesteps. We take the load imbalance distribution from the last section as starting point. Let d_1 and d_2 be two such distributions. We take the norm of the difference of

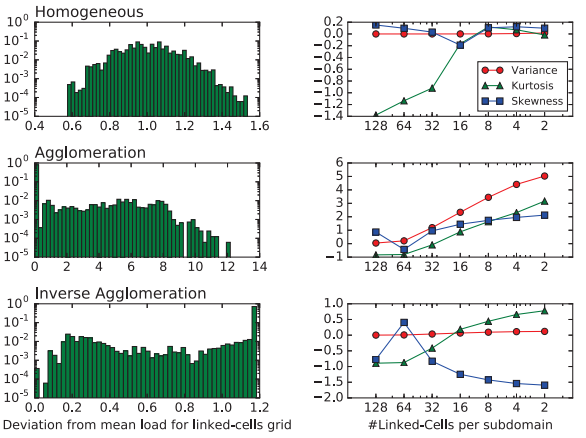


Figure 2. Comparison of the load imbalance distribution on the linked-cells grid on the left (different x-axis scales) and its moments for different levels of the hierarchy (subdomain sizes) on the right for three different scenarios: One homogeneous, one agglomeration and one agglomeration with inverse density, i.e., empty regions in a dense fluid. The load distributions are log-scaled to highlight their skewness. For the homogeneous distribution the variance is almost zero. The agglomeration scenario is highly inhomogeneous and right-skewed. The inverse scenario has a variance near zero.

these two distributions: $dyn(d_1, d_2) := ||d_1 - d_2||$. We use this metric in two forms. First, to quantify the dynamics over time with a fixed timestep, i.e. calculate $dyn(d_t, d_{t+\delta_t})$ for a fixed δ_t and varying t. Second, to quantify the change that happened since a particular timestep, i.e. calculate $dyn(d_t, d_{t+\delta_t})$ for fixed t and varying δ_t. We also apply the concept of different hierarchies which we presented for heterogeneity to this dynamics metric.

For a nucleation scenario, which does not involve movement of the nuclei, the value of $dyn(d_1, d_2)$ decreases over time. A pure movement scenario shows a mostly constant behavior. In a highly dynamic scenario, we can expect to see $dyn(d_t, d_{t+\delta_t})$ increase when starting from an arbitrary point in time.

We present evaluations of both metrics on the agglomeration scenario in the results in Section VII and see their value for explaining the behavior of load-balancing methods. But first, we describe the load-balancing methods we consider and compare.

VI. Load-Balancing Algorithms

The main component of a load-balancer is a partitioning algorithm. Additionally, a load migration routine which, in our case, moves all reassigned linked-cells from one process to the respective new one, is necessary. The particles (and cells) to be moved to other processes are referred to as *migration volume*.

We use the two terms *static* load-balancing and *dynamic rebalancing* to refer to two different concepts of load-balancing. The first one refers to the static partitioning once at the beginning of a simulation while the latter refers to the repartitioning during runtime. To efficiently rebalance a simulation during runtime, not only the partitioning algorithm has to be fast, but it also has to keep the migration volume and the amount of processes to communicate with low.

This motivates what is sometimes called *load migration*: exchange cells only between neighbors. We call the load-balancers *local* which fulfill this criterion as they (mostly) communicate with their neighborhood only. One might allow for global communication even in local load-balancers simply to increase the rate of convergence if they are iterative. An equal categorization is that local load-balancers build on an existing partitioning and only adapt it slightly. Note, that local load-balancers are inherently distributed. In contrast to local ones, load-balancers which completely redistribute the domain are consequently called *global* and usually do not restrict the migration volume or need a dedicated load-balancing process.

Following this classification, we briefly sketch the implementational ideas of graph partitioning and orthogonal recursive bisection as static, global load-balancers and an SFC-based, a grid-based and a diffusion-based one as local load-balancers.

A. Graph partitioning

The main task is to define a mapping from the load-balancing problem to a graph and then let an external library do the partitioning. We use METIS [41]; ParMETIS [42] can be used to partition in parallel. Since we seek to partition the set of linked-cells, it is natural to use these as nodes of the graph with the corresponding half-shell cost of each cell as vertex cost. The set of edges is determined by the full-shell neighborhood relation. As edge cost we use the calculation cost of particle pairs between two adjacent cells. Then, a cut at a specific edge adds the cross-boundary cost of the respective cell pair to the overall cost.

This fact reveals that the objective function of load-balancing does not map correctly to graph partitioning: It minimizes the total cross-boundary interaction cost subject to equalizing the half-shell calculation cost in each subdomain. However, we rather seek for a partition minimizing the sum of both, local half-shell cost and local cross-boundary interaction cost. Despite this fact, graph partitioning gives the best results for static partitioning. Also, communication cost is not directly mappable because they incur only at boundary cells and are defined between subdomains rather than cells [23]. Therefore, we left them out in the explanation above. In [35] an example is given why communication cost should be included anyway on a cell-to-cell basis.

B. ORB

The idea of ORB is to approximate the p-way partitioning problem by successive bisections. A good description can be found in [35]. The basic procedure is to fix a dimension (or search in all dimensions and then take the best) and project

the 3D domain to this dimension by summing up the cost over all layers of cells. Then, we solve the 1D bisection problem by iterating through the chosen dimension and seeking the section layer which equalizes the two sections best. Also, we take communication and cross-boundary interaction cost at the chosen section layer into account. When a section layer has been chosen, we recursively do the same procedure again in the two newly created subdomains until there exists a one-to-one mapping between subdomains and processes. The partitioning can be stored in a k-d tree [43]. Note, that this is a sequential process which has to be executed on a distinguished node. This node beforehand has to collect all the necessary information and afterwards distribute the new k-d tree. One major problem of this scheme is that the subdomain cost at future section layers are not known during a bisection. These future cost might change the current subdomain's cost significantly.

C. SFC

The SFC-based load-balancer uses an even simpler approximation: Linearize the 3D domain using a space-filling curve and partition it in 1D. This can be done, again, by 1D bisectioning as in [13]. However, we employ a slightly different scheme because it is easier to see the parallelism in it. Let ℓ be the local cost of a subdomain. We calculate the target load $\bar{\ell}$ per subdomain via global reduction and the prefix load $\hat{\ell}$ of a subdomain along the space-filling curve. Then, in every subdomain we iterate over its cells calculating the prefix sum $\hat{\ell}_i$ of each cell. We calculate the new subdomain for each cell as $p_i = \lfloor \frac{\hat{\ell}_i}{\bar{\ell}} \rfloor$. The partitioning is given by the subdomain boundaries, i.e. where p_i changes, as pairs of process number and index on the space-filling curve. Using this load-balancer, the migration is most likely between a subdomain itself and the two neighboring processes along the SFC only if the repartitioning is done often enough. Also, non-local migration is trivial because every process knows all process boundaries. So it can easily determine its send and receive partners. As cost we simply take the half-shell interaction cost of a cell since the control over partition boundaries is fully handed over to the space-filling curve. We use the Hilbert curve [44] and the Z curve or Morton order [45]. In our 2D experiments we find that the two curves on average produce slightly less than 8 neighbors per subdomain. However, the maximum number of neighbors in heterogeneous scenarios is typically larger than 10.

D. Grid-based

In [29] and [30], Deng et al. propose a dynamic 2D rebalancing scheme where the partitions are given as cells of a grid. To balance the load between the subdomains, the vertices of the grid are shifted while the structure of the grid—and therewith the process topology—stays fixed. The method can be sketched according to [31] as follows: We calculate the load ℓ_p of each subdomain and a center of load c_p (e.g. the center of mass). Let $\bar{\ell}$ be the load averaged over the neighborhood. Then we define a "virtual force" f shifting the vertex v as the sum over the neighborhood, $f := \sum_p (\frac{\ell_p}{\bar{\ell}} - 1)\, u_p$, where u_p is the normalized distance vector from a vertex v to the neighboring subdomain p's center of load c_p. We restrict the grid cells to being convex to avoid collapsing. If a subdomain is completely empty we do not consider it as a term in the virtual force calculation. The vertex movement is restricted to the length of one linked-cell, thus effectively limiting the migration volume of one iteration of this scheme. Like the space-filling curve load-balancer, only scalar costs can be considered, so we again take the half-shell interaction cost as the load metric.

E. Diffusion

Diffusion [25], [26] is arguably the most natural algorithm for load migration presented in this paper. The main idea is that overloaded processes select and send cells to underloaded processes. This is referred to as *sender initiated* in contrast to *receiver initiated* which would be a form of load stealing. We use the transfer volume calculation of Willebeek-LeMair and Reeves [26] and a cell selection scheme adapted from Buchholz [35]. Unlike Buchholz, we do not consider an overall gain for changing the ownership of a cell but we clearly distinguish between cost saved on the sending process and cost arising on the receiving process. Given a transfer volume $\delta_{p\to q}$ of load to be sent from process p to q, we calculate the cost s_i saved on process p owning a boundary cell i by sending it to q. Conversely, we calculate a_i which is the cost this reassignment adds to the receiving process q. These costs are determined counting the cross-boundary particle pairs in neighboring cells, subtracting saved communication and adding additional necessary communication costs. Then, we sort the boundary cells according to s_i, and, starting with the most promising one, we try to send each cell to a neighbor if $\delta_{p\to q} > a_i$. Afterwards, we decrease the admissible transfer volume by a_i. Note that there is no distinguished data structure holding the partitioning. It is implicit.

The transfer volume $\delta_{p\to q}$ is determined in the local neighborhood as follows: Let ℓ_p be the load of a subdomain p and $\bar{\ell}$ the load averaged over the neighborhood. Each process defines for each neighbor q an absolute deficiency value $h_q := \max\{\bar{\ell} - \ell_q, 0\}$ and a relative one, $\hat{h}_q := \frac{1}{H_p} h_q$, where H_p is summed over the neighborhood $H_p = \sum_q h_q$. Then, the send volume is given as $\delta_{p\to q} = (\ell_p - \bar{\ell})\, \hat{h}_q$.

F. Enhancements

We observe that in scenarios driven by the movement of large particle clusters it becomes crucial how to divide the (mostly) empty space around these clusters becomes crucial. Especially graph partitioning is prone to assign empty parts of the domain to a single process—because they do not cause much or any load at all. Considering a high load dynamics of

a scenario, the particle clusters will eventually move into the large empty parts which then causes a massive overload of one process. Therefore, we propose an enhancement to make load-balancing more robust. We convolve the cell load values before partitioning with a Gaussian kernel which makes cells near cells with a high load more expensive. This forces the partitioner to not only partition the cluster itself but also a "halo" region around it. We study this enhancement exemplarily with graph partitioning in the results section.

While local load-balancers are prone to get stuck in local minima, Begau and Sutman [31], e.g., mention that their grid-based method sometimes needs a "reset". Global load-balancers cannot be executed often enough to maintain a good load balance in highly dynamic scenarios. This motivates what we call *hybrid load-balancers* which consist of a fitting pair of global and local methods. The local method is used to maintain the load balance the global method has established. The global one is only executed, e.g., if there is a manifesting imbalance, if the partitioning is too suboptimal with respect to subdomain layouts or simply after a fixed number of timesteps.

VII. Results

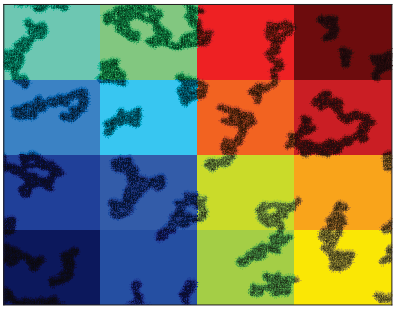
In this section we present results for the 2D agglomeration scenario presented in Section IV to see how the different load-balancing methods can cope with heterogeneity and load dynamics. We use snapshots of the scenario as input to a Python implementation of all the above-mentioned load-balancers. As the baseline, we employ a standard spatial decomposition by splitting the domain into 16 equally sized boxes on a 4×4 grid, see Figure 3(a). To assess the quality of any decomposition we evaluate it with the metric we presented in Section III. Let this parallel load be T_n. We also calculate the load T_1 a sequential simulation causes. Then, we calculate the speedup $S = \frac{T_1}{T_n}$. Note, that T_1 neither includes communication nor cross-boundary calculation cost.

The speedup for the first $2 \cdot 10^6$ timesteps of the mentioned decomposition corresponds to the blue curves in Figure 4. As one can see, the speedup deteriorates rapidly at the beginning of the simulation. Later in the simulation, starting after about $1.5 \cdot 10^7$ timesteps, we would observe another speedup drop, this time to 5, which is caused by two clusters agglomerating in a single subdomain. The overall drop height depends on various simulation parameters. To explain this deterioration we look at the heterogeneity of the scenario. The Figures 3(b), 3(c) and 3(d) show the moments for the upper part of the hierarchy of load imbalances as defined in Section V-A for the agglomeration scenario. Here, we show results for up to $4 \cdot 10^7$ timesteps, which includes both speedup drops. We observe that the variance, i.e. heterogeneity, increases during both drops. During the second drop also the skewness and kurtosis for 4×4 cells increase sharply which implies that expensive outliers get more common. We can also see that for 4×4 cells the variance gets bigger than 0.05 within the first timesteps while the speedup in this small time frame gets practically halved. Even a decomposition into 4 equally sized boxes would lose its efficiency in the last quarter of the simulation.

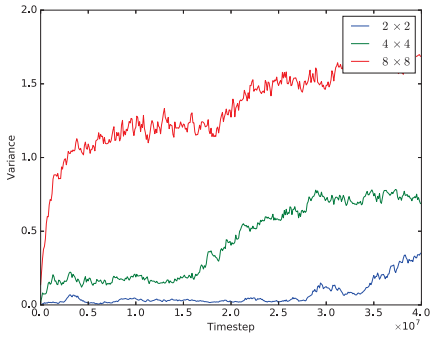
Now, we test how different load-balancers perform on the example scenario. We use 16 processes and partition at *fixed* points in time in order to see how the partitions behave over time. For static load-balancing we use 40,000 timesteps, for local load-balancing 4,000 and we evaluate the partitioning cost using the current particle positions every 1,000 timesteps. The results for graph partitioning, ORB and the SFC-based load-balancer using the Hilbert curve can be found in Figure 4(a) for the first two million timesteps. We did not measure a significant difference in partitioning quality between the Hilbert and the Z curve. The results of grid and diffusion-based load balancers are given in Figure 4(b). For the former we did 10 iterations per load-balancing step and for the latter a single one sufficed to get good results.

We observe that the quality of partitionings at the exact timestep of rebalancing does not vary much over time for any load-balancer—even beyond the time range plotted in Figure 4. Graph partitioning gives the best results followed by diffusion. The ORB partitioning varies most. Its major disadvantage are the axis-parallel section layers: The achievable partitioning quality depends on the position of the agglomerates. This can also be observed for the grid-based scheme. It seems to have problems when the change in variance is high. This can—at least partly—be remedied by investing more iterations. Therefore, we show the results for 10 iterations each. The formation of large particle clusters requires to cut through clusters to balance the load. These cuts increase the load since cross-boundary interactions in high density regions need to be calculated, and therefore the speedup with respect to the sequential simulation drops. This also holds true if we simply increase the number of processes. Thus, strong scaling becomes increasingly ineffective. But if we use 100 processes for example and do graph partitioning on the agglomeration scenario, we still get an average speedup of around 70 (in contrast to about 20 for the unbalanced simulation) at the partitioning timesteps.

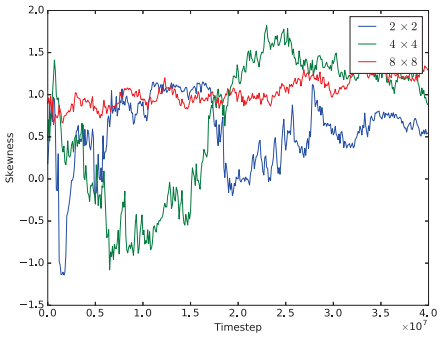
We get very different results when inspecting the quality of partitionings between rebalancing steps, i.e., when the simulation has progressed but the partitioning has not yet been adapted. This behavior depends strongly on the scenario dynamics. After the initial partitioning the speedup drops rapidly to the level of the non-balanced decomposition for the static load-balancers. We can conclude that an initial partitioning in the homogeneous state of the scenario does not make much sense. The dynamic load-balancers behave better, simply because they are cheaper and can therefore be applied with higher frequency. Using a lower load-balancing frequency, we spot another important fact: The



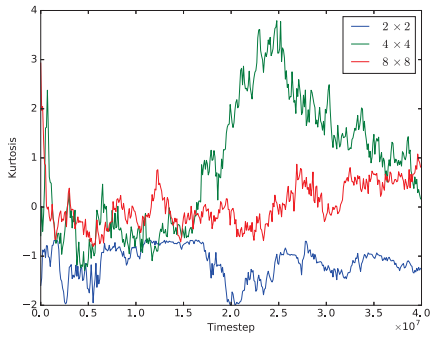
(a) The agglomeration scenario at timestep 10^6. Colors encode the different subdomains and the particle distribution is overlayed.



(b) The variance of the load distribution of the agglomeration scenario over time for different cell sizes. The increase for 4×4 cells is directly related to a drop of speedup.



(c) The skewness of the load distribution of the agglomeration scenario over time for different cell sizes. The change in the skewness for 4×4 means that more cells get empty.

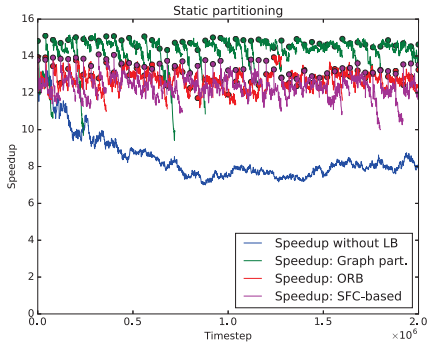


(d) The excess kurtosis of the load distribution of the agglomeration sceneario over time for different cell sizes. A positive kurtosis means outliers become more frequent.

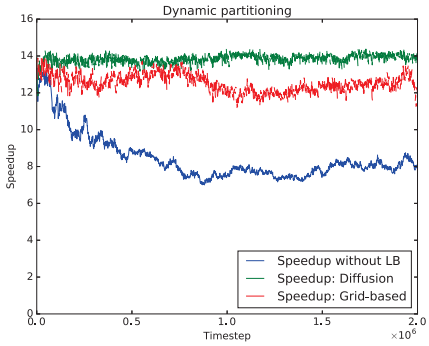
Figure 3. On the upper left the scenario with the basic domain decomposition. The other three pictures show the seconds (variance), third (skewness) and fourth (kurtosis) standardized moments of the binned particle load distribution on different levels for different levels of the hierarchy (cell count in the legends).

speedup of balanced simulations might actually drop *below* the unbalanced one, meaning that infrequent load-balancing may become harmful for overall performance. Comparing the static load-balancers, ORB seems to have the best consistency over time especially at later points in the simulation when large particle clusters move around. However, as we stated above, we use convolution to attempt to remedy this fact, see Figure 4(c). There, we only balance every 100,000 timesteps in order to see the effects of too infrequent load-balancing. The graph partitioning results drop multiple times to a level which is way smaller than its static partitioning quality. To remedy this problem, we convolve each load value with a Gaussian kernel of width 10. We can see that the static partitioning quality decreases but the behavior over time gets better. In Section VI-D we gave an explanation for this. Note that the static partitioning quality of the convolved scheme must not necessarily decrease because the graph partitioning libraries also do approximations of their own. To give an indicator of how much simulation cost according to the load model the partitionings actually save, we assume the simulation cost to be constant within one evaluation step (1,000 timesteps). Using this approximation, graph partitioning saves 42 % of the cost within the plotted range, ORB 35 % and the SFB-based method 32 %. We regard these numbers as an upper bound to the saved cost because of the approximations we mentioned and since we don't include the cost of load-balancing. However, for the overall simulation, the savings are higher, since the speedup of the unbalanced simulation drops further while the quality of the load-balancing methods stays roughly on the same level.

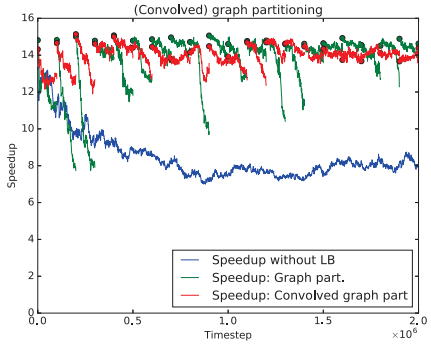
We now exemplarily take the partitioning from Figure 4(c) and try to explain the difference in the temporal suitability of the partitionings after timestep $t_0 = 200{,}000$ and $t_1 = 300{,}000$. After the first one, the speedup of graph partitioning degrades rapidly which is not the case after the partitioning at t_1. To explain this difference, we consider the load dynamics metric as described in Section V-B. We



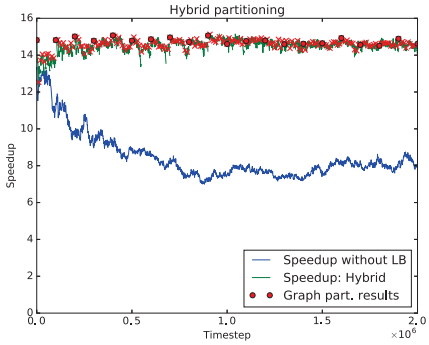
(a) Static partitioning every 40,000 timesteps. In green graph partitioning, ORB in red and the SFC-based in magenta. Graph partitioning performs best at this rate of partitioning.



(b) Dynamic partitioning every 4,000 timesteps. In green diffusion and in red the grid-based method. We don't plot points at partitioning timesteps in this figure. Diffusion performs better than the grid-based method at this rate of rebalancing.



(c) Effects of the convolution of load values before graph partitioning: In green unconvolved; in red convolved with a Gaussian kernel of width 10. Both are applied every 100,000 timesteps. Convolution has a positive effect on the robustness of the partitioing over time. The drop heights are reduced, but static partitioning quality usually gets worse.



(d) A hybrid load-balancing scheme based on graph partitioning and diffusion. Graph partitioning is applied every 100,000 timesteps (dots) and diffusion (x marks) every 10,000. This leads to the highest speedups.

Figure 4. Load-balancing results in comparison to the unbalanced simulation in blue. Results of static partitioners are shown in the upper left picture; dynamic load-balancing in the upper right. The lower left picture shows the convolution strategy and the lower right picture the hybrid method. The points mark the speedup directly after partitioning. The speedup a partitioning produces in subsequent timesteps is represented by the tails.

subtract the imbalance distributions of subsequent timesteps (up to t_1) from the one of t_0. The same is done for t_1. Figure 5 shows the load dynamics metric for the two mentioned timespans. In the first timespan the load dynamics rises higher and more quickly. Therefore, this timespan is more problematic with respect to the temporal suitability of the subdomains. While graph partitioning assigns a single subdomain roughly 50 % of all cells at t_0, the convolved scheme only assigns approximately 10 % of all cells to the largest subdomain. The same holds true for ORB. Graph partitioning is affected more by higher load dynamics than the other methods as we reasoned in Section IV-C. Another important fact we mentioned before is the subdomain surface. We compare the surface of the two largest subdomains in both cases. For pure graph partitioning this surface is over 10 times higher than for the convolved scheme. For ORB this number is even higher because it produces rectangular subdomains. These differences make the graph partitioning drop steeper than the partitionings produced by any other method.

Diffusion and grid-based load-balancing show a good temporal behavior because we apply them more often. We see that diffusion applied in this frequency is in this scenario capable of producing a partitioning which is almost as good as graph partitioning. Over time, diffusion saves 40 % of the cost and the grid-based load-balancer 34 %. The grid-based method is slightly worse because of the restrictions it imposes on the partitioning.

Finally, we show results for the hybrid methods which we proposed. A natural choice is to combine graph partitioning

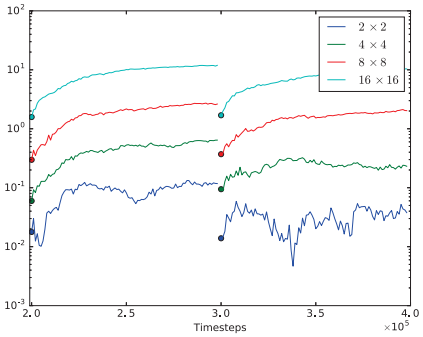


Figure 5. Scenario load dynamics with respect to $t_0 = 200{,}000$ and $t_1 = 300{,}000$ in a semi-log plot. The imbalance increases quickly by more or less one order of magnitude depending on the level in the first time frame. The rise is lower in the second time frame.

and diffusion, see Figure 4(d). As we can see, the diffusive strategy can keep the speedup on the high level which graph partitioning establishes. This is slightly worse in the first few iterations, where the change in variance is very high. More iterations of the diffusive load-balancer or a higher rebalancing frequency solve this problem. The results over time are better than pure diffusion and pure graph partitioning. It saves 43 % of the cost. Another possible hybrid method to test in the future is a combination of the grid-based scheme with a global method establishing load-balance through a rectangular non-equidistant grid.

To really estimate the savings of a simulation we would have to consider the time rebalancing takes. Therefore the savings will—in a real simulation—definitely be lower than the ones we mention here. Choosing the right points in time when to do rebalancing is, given the cost of rebalancing, another factor in the optimization problem. However, to the best of our knowledge there is no general work on when to best do rebalancing steps based on both simulation cost and rebalancing cost, neither results on how long different load-balancers take to execute in example scenarios. We hope to be able to provide more insight to this problem in a follow-up work. Here, we can only conclude with the very general cost estimate and consideration of scalability: It should be obvious that the local methods cause computational work of at most an order of a timestep. The migration volume for one iteration of the diffusive and grid-based scheme is limited to one ghost-layer exchange. This does not change with increasing processor count since these schemes work purely local. Therefore, they are scalable. The SFC-based partitioner needs two collective operations: One *allreduce* to determine the target load and one *exscan* to determine the prefix load. ORB clearly does not scale in the form presented here since it needs a distinguished node to do the partitioning. For graph partitioning the most crucial part is limiting the migration volume. In the version presented here, this is not the case. Also, the graph partitioner itself might need a considerable amount of time when requesting a large number of partitions.

VIII. Conclusions

We have presented metrics to quantify two basic properties of particle simulation scenarios: heterogeneity and load dynamics. We have shown that these two metrics quantify relevant properties of the scenario and enable us to reason over the performance of specific load-balancing methods. Our work demonstrates that it is crucial to define such metrics on scenarios to be able to understand the effects of load-balancing. This is a central step towards an automated mapping from scenarios to suitable load-balancing methods.

We have shown that these metrics can enable us to reason about the behavior of load-balancing methods and partitionings on scenarios. We postulate that the insight gained through these metrics for a specific scenario, e.g. the use of a specific load-balancing method or the optimal rebalancing times, can be generalized to other scenarios with similar properties.

We have presented evaluations of load-balancing methods on an example agglomeration scenario. There, rebalancing too infrequently is clearly harmful to the overall performance. Motivated by the scenario's load dynamics, we have proposed the convolution of load values as an enhancement and we have applied it to graph partitioning. Finally, we demonstrated how hybrid methods that combine global and local methods can be used to "reset" the partitioning to a a better overall optimum.

A. Further Work

Since our main concern is to understand the complex relationship between simulation scenarios and load balancing methods, the amount of knowledge regarding the quality of different load balancing methods for scenarios with different metric values needs to be increased. This will be crucial to formulate clear hypotheses which can then be tested. We will continue to work on more concise metrics for evaluation and extend our assessment to three dimensions.

One necessary step to attain reliable predictive results for the selection of the optimal load-balancing method will be to include the cost of rebalancing. This especially holds true when it comes to huge simulations on large high performance computers (HPC). With the issue of scalability on current HPC systems comes another requirement: Awareness of the network topology has to be considered not only for the produced partitioning but also for the load-balancer itself, e.g. by considering local load exchanges only within a node.

The load model can be improved by including bonded interaction cost for vastly bonded scenarios. Also the possibility of integrating communication hiding or latency cost on a per-neighbor basis should be investigated. For a more realistic model of simulations on machines which are not used exclusively, a stochastic network congestion model could be included in the communication cost.

ACKNOWLEDGMENT

The authors gratefully acknowledge financial support provided by the German Research Foundation (DFG) as part of the Collaborative Research Center (SFB) 716.

REFERENCES

[1] D. C. Rapaport, *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 1995.

[2] J. M. Haile, *Molecular Dynamics Simulation*. John Wiley & Sons, Ltd., 1997.

[3] M. Griebel, S. Knapek, G. Zumbusch, and A. Caglar, *Numerische Simulation in der Moleküldynamik. Numerik, Algorithmen, Parallelisierung, Anwendungen*. Berlin, Heidelberg: Springer, 2003.

[4] L. Isella and Y. Drossinos, "Langevin agglomeration of nanoparticles interacting via a central potential," *Phys. Rev. E*, vol. 82, p. 011404, Jul 2010.

[5] G. Inci, A. Arnold, A. Kronenburg, and R. Weeber, "Modeling nanoparticle agglomeration using local interactions," *Aerosol Science and Technology*, vol. 48, no. 8, pp. 842–852, jul 2014.

[6] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*. Bristol, PA, USA: Taylor & Francis, Inc., 1988.

[7] M. Allen and D. Tildesley, *Computer Simulation of Liquids*, ser. Oxford Science Publ. Clarendon Press, 1989.

[8] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1 – 19, 1995.

[9] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, "NAMD2: Greater scalability for parallel molecular dynamics," *Journal of Computational Physics*, vol. 151, no. 1, pp. 283 – 312, 1999.

[10] B. Hayes, "The easiest hard problem," *American Scientist*, vol. 90, no. 2, pp. 113–117, March–April 2002.

[11] H. Sagan, *Space-Filling Curves*. Springer New York, 1994.

[12] M. Bader, *Space-Filling Curves - An Introduction with Applications in Scientific Computing*, ser. Texts in Computational Science and Engineering. Springer-Verlag, 2013, vol. 9.

[13] C. Xiaolin and M. Zeyao, "A new scalable parallel method for molecular dynamics based on cell-block data structure," in *Parallel and Distributed Processing and Applications*, ser. Lecture Notes in Computer Science, J. Cao, L. Yang, M. Guo, and F. Lau, Eds. Springer Berlin Heidelberg, 2005, vol. 3358, pp. 757–764.

[14] M. J. Berger and S. H. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 570–580, May 1987.

[15] C. Niethammer, S. Becker, M. Bernreuther, M. Buchholz, W. Eckhardt, A. Heinecke, S. Werth, H. Bungartz, C. W. Glass, H. Hasse, J. Vrabec, and M. Horsch, "ls1 mardyn: The massively parallel molecular dynamics code for large systems," *CoRR*, vol. abs/1408.4599, 2014.

[16] R. Brunner, J. Phillips, and L. Kale, "Scalable molecular dynamics for large biomolecular systems," in *Supercomputing, ACM/IEEE 2000 Conference*, Nov 2000, pp. 45–45.

[17] A. Bhatelé, L. V. Kalé, and S. Kumar, "Dynamic topology aware load balancing algorithms for molecular dynamics applications," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 110–116.

[18] H. D. Simon and S.-H. Teng, "How good is recursive bisection?" *SIAM J. Sci. Comput.*, vol. 18, no. 5, pp. 1436–1445, Sep. 1997.

[19] F. Fleissner and P. Eberhard, "Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection," *International Journal for Numerical Methods in Engineering*, vol. 74, no. 4, pp. 531–553, 2008.

[20] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation," *J. Chem. Theory Comput.*, vol. 4, no. 3, pp. 435–447, Feb. 2008.

[21] A. Nakano and T. Campbell, "An adaptive curvilinear-coordinate approach to dynamic load balancing of parallel multiresolution molecular dynamics," *Parallel Comput.*, vol. 23, no. 10, pp. 1461–1478, Oct. 1997.

[22] C.-E. Bichot and P. Siarry, *Graph partitioning*. John Wiley & Sons, 2013.

[23] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, no. 12, pp. 1519 – 1534, 2000, graph Partitioning and Parallel Computing.

[24] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdağ, R. T. Heaphy, and L. Riesen, "A repartitioning hypergraph model for dynamic loadbalancing," *J. Parallel Distrib. Comput.*, vol. 69, no. 8, pp. 711–724, Aug. 2009.

[25] J. Boillat, F. Brugé, and P. Kropf, "A dynamic load-balancing algorithm for molecular dynamics simulation on multi-processor systems," *Journal of Computational Physics*, vol. 96, no. 1, pp. 1 – 14, 1991.

[26] M. H. Willebeek-LeMair and A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 9, pp. 979–993, Sep. 1993.

[27] Y. Deng, R. W. Lau, D. of Computer Science, C. U. of Hong Kong, and H. Kong, "Heat diffusion based dynamic load balancing for distributed virtual environments," in *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '10. New York, NY, USA: ACM, 2010, pp. 203–210.

[28] Y. Hu and R. Blake, "An optimal dynamic load balancing algorithm," Daresbury Laboratory, Warrington, WA4 4AD, UK, techreport, 1995, preprint DL-P-95-011.

[29] Y.-F. Deng, R. McCoy, R. Marr, R. Peierls, and O. Yasar, "Molecular dynamics on distributed-memory MIMD computers with load balancing," *Applied Mathematics Letters*, vol. 8, no. 3, pp. 37 – 41, 1995.

[30] Y. Deng, R. F. Peierls, and C. Rivera, "An adaptive load balancing method for parallel molecular dynamics simulations," *Journal of Computational Physics*, vol. 161, no. 1, pp. 250 – 263, 2000.

[31] C. Begau and G. Sutmann, "Adaptive dynamic load-balancing with irregular domain decomposition for particle simulations," *Computer Physics Communications*, vol. 190, no. 0, pp. 51 – 61, 2015.

[32] J. Roth, *IMD – A Typical Massively Parallel Molecular Dynamics Code for Classical Simulations – Structure, Applications, Latest Developments*. Springer, 2013, ch. Part II - 2, pp. 63–76.

[33] S. Schambeger and J. Wierum, "Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves," in *Proceedings of the 7th International Conference on Parallel Computing Technologies*, ser. LNCS, vol. 2763, 2003, pp. 165–179.

[34] W. F. Mitchell, "A refinement-tree based partitioning method for dynamic load balancing with adaptively refined grids," *J. Parallel Distrib. Comput.*, vol. 67, no. 4, pp. 417–429, apr 2007.

[35] M. Buchholz, *Framework zur Parallelisierung von Molekulardynamiksimulationen in verfahrenstechnischen Anwendungen*. Verlag Dr. Hut, 2010.

[36] L. Nyland, J. Prins, H.-C. Kum, and L. Wang, "Modeling dynamic load balancing in molecular dynamics to achieve scalable parallel execution," in *IRREGULAR '98: Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel*. Springer-Verlag, 1998, pp. 356–365.

[37] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990.

[38] R. H. Bisseling and W. F. McColl, "Scientific computing on bulk synchronous parallel architectures," Department of Mathematics, Utrecht University, techreport, Apr. 1994.

[39] A. Arnold, O. Lenz, S. Kesselheim, R. Weeber, F. Fahrenberger, D. Roehm, P. Košovan, and C. Holm, "ESPResSo 3.1: Molecular dynamics software for coarse-grained models," in *Meshfree Methods for Partial Differential Equations VI*, ser. Lecture Notes in Computational Science and Engineering, M. Griebel and M. A. Schweitzer, Eds., vol. 89. Springer, September 2012, pp. 1–23.

[40] C. Xiong and S. Friedlander, "Morphological properties of atmospheric aerosol aggregates," *Proceedings of the National Academy of Sciences*, vol. 98, no. 21, pp. 11 851–11 856, 2001.

[41] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[42] K. Schloegel, G. Karypis, and V. Kumar, "Parallel multilevel algorithms for multi-constraint graph partitioning," in *European Conference on Parallel Processing*. Springer, 2000, pp. 296–310.

[43] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[44] D. Hilbert, "Über die stetige Abbildung einer Line auf ein Flächenstück," *Mathematische Annalen*, vol. 38, no. 3, pp. 459–460, 1891.

[45] G. M. Morton, "A computer oriented geodetic data base; and a new technique in file sequencing," IBM Ltd., Tech. Rep., 1966.