



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Dynamic load balancing algorithm for Molecular Dynamics based on Voronoi cells domain decompositions

J. L. Fattebert, D. F. Richards, J. N. Glosli

October 17, 2011

Computer Physics Communications

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.

Dynamic load balancing algorithm for Molecular Dynamics based on Voronoi cells domain decompositions

J.-L. Fattebert^a, D.F. Richards^b, J.N. Glosli^b

^a *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551*

^b *Physical and Life Sciences, Lawrence Livermore National Laboratory, Livermore, CA 94551*

Abstract

We present a new algorithm for automatic parallel load balancing in classical molecular dynamics. It assumes a spatial domain decomposition of particles into Voronoi cells. It is a gradient method which attempts to minimize a cost function by displacing Voronoi sites associated to each processor/sub-domain along steepest descent directions. Excellent load balance has been obtained for quasi-2D and 3D practical applications, with up to $440 \cdot 10^6$ particles on 65536 MPI tasks.

Keywords: Molecular dynamics, parallel load balancing, computational geometry

2000 MSC: 70Fxx, 68W10, 65Dxx

1. Introduction

Achieving a good load balance between cpus in large parallel calculations is crucial for an optimal use of computer resources. With large scale molecular dynamics (MD) now being performed on thousands of processors and involving billions of particles, this is even more important as the time to solution will be determined by the processor taking the longest time to complete its assigned work.

While for some applications such as homogeneous solids or liquids a natural uniform spatial domain decomposition leads to a fairly good load balance, many applications of interest show inhomogeneities due either to the local density of particles or to a different amount of computational work/particle at various locations. Load balancing is also needed for applications to finite systems with vacuum regions which cannot be decomposed naturally into spatially uniform subdomains.

Dynamic load balancing is also important when the performance of each processor differs. This is the case for heterogeneous computer clusters, but also if one processor is running slower than its peak performance due to various

hardware problems. As we are moving towards supercomputers with hundreds of thousand processors, the likelihood of having one processor causing slowdowns becomes non-negligible.

Typically in parallel molecular dynamics simulation algorithms, the spatial computational domain is divided into subdomains associated to processors. Each processor is responsible for advancing particles located inside its assigned sub-domain at a certain point in time. Work assigned to each processor then depends on various algorithmic choices and a good load balancing will be achieved if one can minimize the time processors are idle waiting for others to complete their work. Overhead due to communications should also be limited to minimize time to solution, which in general means keeping the surface to volume ratio to a relatively low value.

Various algorithms have been proposed in recent years to address parallel load balancing in molecular dynamics simulations. Their detailed implementation typically depends on choices made in the domain decomposition algorithm. Deng et al. [4] proposed to divide space into a number of sub-domains larger than the number of processors and assign multiple sub-domains to each processor. Load balancing can then be improved by exchanging sub-domains between tasks. This idea can be generalized and computational work (instead of domains) can be divided into many “units of work”. This is the approach used in NAMD, where load balancing is achieved by an appropriate distribution of these units of work [12].

Other approaches typically modify the shapes and positions of the subdomains associated to each processor. Nakano and Campbell [16] for instance use a curvilinear-coordinate approach to reshape and resize domains while preserving nearest neighbor topology and minimize the number of communications. Deng et al. [5] proposed another scheme in which they move the vertices of the domain decomposition grid towards the most heavily loaded regions of space. They also constrain the original topology of the mesh to remain the same, while allowing the shapes of each domain to vary. Fleissner et al. [6] use an orthogonal recursive bisection as a domain decomposition scheme in a hierarchical approach based on wall clock time measurements. Sub-domain boundaries are aligned with the coordinate axis. In GROMACS, [8] the domain decomposition is done one dimension at a time, load balancing each dimension successively.

Closer to the approach we propose in this paper, Zhakhovskii et al. [19] described an algorithm for 2D domain decompositions made of Voronoi cells. They propose to move the locations of Voronoi sites—and thus domain boundaries—to balance computational work load at regular intervals. Diffusion based algorithms, which move domain boundaries to reach an equal number of particles on each processors were proposed two decades ago[2] for a 1D domain decomposition. In that case, Voronoi tessellations are trivial to compute. Koradi et al. [11] also proposed a load balancing scheme for Voronoi cell based domain decomposition. Their approach consists however in changing the metric used to assign particles to Voronoi centers in order to improve load balancing. Voronoi sites are moved after a load balancing step to coincide with the center of mass of the particles associated with the local subdomain.

In this paper, we present a new dynamic load balancing algorithm for general 3D molecular dynamics simulations for the case of a spatial domain decomposition of particles into Voronoi volumes. This is the domain decomposition algorithm used in our code *ddcMD*. In *ddcMD*, each parallel task is associated with a Voronoi cell and that task is responsible for propagating in time all the particles initially located within the boundaries of that domain, which we will call “local” particles. Each processor also needs to know about the positions of some “remote” particles which interact with the local particles through short range forces but belong to other tasks. Each Voronoi cell is essentially described by the positions of a Voronoi site: all the points in space closer to that site than any other site form the Voronoi cell associated to that site. This means that the positions of the Voronoi sites determine the amount of work each task has to accomplish. Moving those sites will change the assignment of particles to tasks and thus computational load.

Following the idea proposed by Zhakhovskii et al. [19] for 2D domain decompositions, we propose to move the locations of Voronoi sites to balance computational work load at regular intervals. Our approach is a gradient method [3] in which we try to minimize a cost function G . Following the steepest descent direction of the cost function has the net effect of exchanging particles between adjacent domains and leads to a diffusion algorithm. Our approach goes beyond what is proposed in [19] by introducing a better estimate of the work transfer between task using actual values of Voronoi cells volumes and facets, and applying it to large scale 3D problems. To calculate the gradient of the cost function G with respect to Voronoi sites positions, we estimate the amount of work/unit volume and work flows through cells interfaces. We use a widely available computational geometry software (Qhull [1]) to calculate tessellations, as well as volumes and facets areas of general Voronoi cells in 3D. Our approach is scalable and associates one Voronoi cell to each MPI task. But the computational work for each Voronoi cell can be shared among multiple cores.

After describing our load balancing algorithm in section 2, we demonstrate the efficiency of this approach on various applications in section 3 showing almost perfect load balancing.

Note that the problem of optimizing the locations of Voronoi sites is quite general and in no way restricted to molecular dynamics. For instance the facility location or geographical optimization problem (in 2D) falls into that category and make use of similar algorithms [10].

2. Algorithm

2.1. Load balance Algorithm

Let us assume the calculation is divided among P parallel tasks. Let $t_i^{(k)}$ be a timing measured on task i between the last two load balancing iterations $k - 1$ and k , which may include numerous MD steps. Following [3], suppose we transfer a net amount of work w_{ij} from task j to task i before running the next

iteration. We then expect the new timing on task i to become

$$\tilde{t}_i^{(k+1)} = t_i^{(k)} + \eta \sum_{j \in Z_i} w_{ij} \quad (1)$$

where Z_i denotes the set of all the tasks sharing a Voronoi facet with the Voronoi cell i , and η is the wall clock time necessary to execute a unit of work. We assume the average timing over all the tasks for an iteration is constant over time and we denote it by \bar{T} :

$$\bar{T} = \frac{1}{P} \sum_{i=1}^P t_i^{(k)}. \quad (2)$$

Now let us define a load balancing function

$$G(\{t_i\}_{i=1}^P) = \frac{1}{P} \sum_{i=1}^P \left(\frac{t_i - \bar{T}}{\bar{T}} \right)^2. \quad (3)$$

This function reaches its minimum of zero for a perfectly balanced calculation. The goal of a load balancing algorithm will be to adequately choose the amount of work to transfer between tasks to minimize that function. It is easy to see that G simplifies to

$$G(\{t_i\}_{i=1}^P) = -1 + \frac{1}{P} \sum_{i=1}^P \left(\frac{t_i}{\bar{T}} \right)^2. \quad (4)$$

and thus the problem of minimizing G is equivalent to minimizing F given by

$$F(\{t_i\}_{i=1}^P) = \frac{1}{P} \sum_{i=1}^P \left(\frac{t_i}{\bar{T}} \right)^2. \quad (5)$$

Let $\mathbf{r}_i^{(k)}$ denote the Voronoi site associated to the Voronoi cell i at step k . At step $k+1$, it will be at a new location denoted by

$$\mathbf{r}_i^{(k+1)} = \mathbf{r}_i^{(k)} + \delta \mathbf{r}_i^{(k)} \quad (6)$$

Let V_i denote the volume of the Voronoi cell i . Let us also denote by A_{ij} the area of the common Voronoi facet shared by cells i and j , and by \mathbf{n}_{ij} the unit vector normal to that facet, pointing outside of cell i . We can estimate the amount of work w_{ij} transferred from task j to task i by the following expression

$$\tilde{w}_{ij} = \frac{1}{\eta} \tau_{ij}^{(k)} \cdot A_{ij} \cdot \mathbf{n}_{ij} \cdot \left[\frac{1}{2} \delta \mathbf{r}_i^{(k)} - \frac{1}{2} \delta \mathbf{r}_j^{(k)} \right] \quad (7)$$

where

$$\tau_{ij}^{(k)} = \frac{1}{2} \left(\frac{t_i^{(k)}}{V_i} + \frac{t_j^{(k)}}{V_j} \right).$$

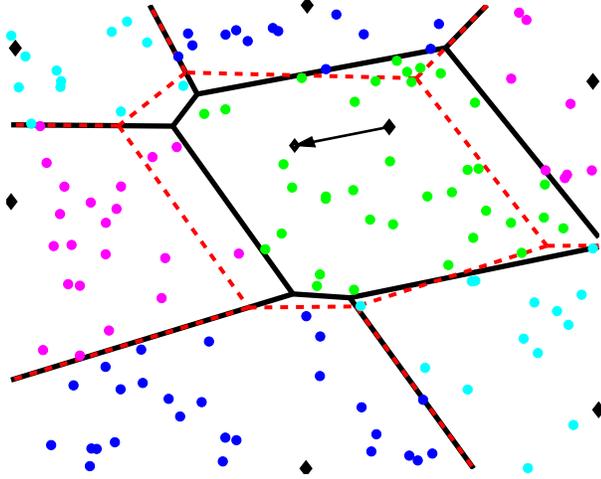


Figure 1: 2D illustration of particles/work transfer resulting from Voronoi sites displacements: One of the initial sites (black diamonds) is moved to a new position resulting in a new Voronoi tessellation (dashed red lines). Some particles (colored according to initial Voronoi tessellation) now belong to a different Voronoi cell which results in more work for the new cell they belong to and less work for the old one.

is a local (symmetric) measure of the timing per unit of volume.

Indeed $\tau_{ij}^{(k)}/\eta$ can be interpreted as the density of work per unit volume, while the rest of the expression—to first order—is the volume transferred from cell j to cell i when the centers i and j move by $\delta\mathbf{r}_i^{(k)}$ and $\delta\mathbf{r}_j^{(k)}$ respectively. Figure 1 illustrates how moving a Voronoi site shifts particles from one subdomain (task) to an adjacent domain. Note that here we use a linear model, that is we assume that the work load is directly proportional to the volume of the Voronoi cell. This is good enough, even though in general the work load scales closer to the square of the number of interacting pairs of particle to be computed.

We have at first order

$$\begin{aligned} \delta F &= F(\{\tilde{t}_i^{(k+1)}\}_{i=1,\dots,P}) - F(\{t_i^{(k)}\}_{i=1,\dots,P}) \\ &\approx \frac{1}{P} \frac{2}{\bar{T}^2} \sum_{i=1}^P t_i^{(k)} \left(\tilde{t}_i^{(k+1)} - t_i^{(k)} \right) \\ &= \frac{\eta}{P} \frac{2}{\bar{T}^2} \sum_{i=1}^P t_i^{(k)} \sum_{j \in Z_i} \omega_{ij} \end{aligned}$$

Thus the α component of the first derivative of F with respect to a Voronoi site position \mathbf{r}_l is given by:

$$\left. \frac{\partial F}{\partial r_{l,\alpha}^{(k+1)}} \right|_{\mathbf{r}_l^{(k+1)} = \mathbf{r}_l^{(k)}} = 2 \frac{\eta}{P \bar{T}^2} \sum_{i=1}^P t_i^{(k)} \frac{\partial}{\partial r_{l,\alpha}^{(k+1)}} \left(\sum_{j \in Z_i} w_{ij} \right) \quad (8)$$

If we assume that $V_i, A_{ij}, \mathbf{n}_{ij}$ remain constant at first order from step k to step $k + 1$, and $\delta \mathbf{r}_i^{(k)}, i = 1, \dots, n_p$ are small, we can write

$$\frac{\partial}{\partial r_{l,\alpha}^{(k+1)}} \tilde{w}_{ij} \approx \frac{1}{2} \frac{\tau_{ij}^{(k)}}{\eta} A_{ij}(\mathbf{n}_{ij})_\alpha [\delta_{il} - \delta_{jl}]. \quad (9)$$

and then obtain the estimate

$$\begin{aligned} \left. \frac{\partial F}{\partial r_{l,\alpha}^{(k+1)}} \right|_{\mathbf{r}_i^{(k+1)} = \mathbf{r}_i^{(k)}} &\approx \frac{1}{P\bar{T}^2} \sum_{i=1}^P t_i^{(k)} \sum_{j \in Z_i} \tau_{ij}^{(k)} A_{ij}(\mathbf{n}_{ij})_\alpha [\delta_{il} - \delta_{jl}] \\ &= \frac{1}{P\bar{T}^2} \sum_{j \in Z_i} \left(t_l^{(k)} - t_j^{(k)} \right) \tau_{lj}^{(k)} A_{lj}(\mathbf{n}_{lj})_\alpha. \end{aligned}$$

Now we can move the Voronoi sites by one step in the steepest descent direction to reduce F

$$\mathbf{r}_i^{(k+1)} = \mathbf{r}_i^{(k)} - \alpha \nabla_{\mathbf{r}_i} F. \quad (10)$$

Knowing that the minimal value for F is 1, we compute α according to the following equation

$$F(0) - \alpha (\nabla F)^T \nabla F = 1 \quad (11)$$

where

$$\nabla F = \begin{pmatrix} \nabla_{\mathbf{r}_1} F \\ \nabla_{\mathbf{r}_2} F \\ \dots \\ \nabla_{\mathbf{r}_P} F \end{pmatrix}.$$

That results in

$$\alpha = \frac{F(0) - 1}{\|\nabla F\|^2} = \frac{1}{\|\nabla F\|^2} \left(\frac{\sum_{i=1}^P (t_i^{(k)})^2}{P\bar{T}^2} - 1 \right). \quad (12)$$

In practice, we introduce a relaxation parameter γ (typically larger than 1) and move Voronoi centers according to

$$\mathbf{r}_i^{(k+1)} = \mathbf{r}_i^{(k)} - \gamma \alpha \nabla_{\mathbf{r}_i} F. \quad (13)$$

The efficiency of this algorithm can be limited if one needs to wait for the next call to the load balancer to make another move (which may come quite a few MD steps later). However the process can be improved by introducing some inner iterations. If we assume the density of work is somewhat uniform locally and the work is proportional to the volume of the Voronoi cell, one can guess the new load for each task by computing the volume of the new Voronoi cell and multiplying it by the local density of work,

$$\tilde{t}_i^{(k+1)} \approx V_i^{(k+1)} \frac{t_i^{(k)}}{V_i^{(k)}}.$$

This requires only a Voronoi tessellation calculation without moving any particles or computing any forces. Using these new “extrapolated” loads, one can apply another step of the algorithm described above. One can repeat this process several times. We denote the number of these iterations inner iterations as n_{inner} . This procedure is very useful to quickly reach a good load balance when the starting domain decomposition is very unbalanced.

2.2. Voronoi tessellation

For a set of points in the Euclidean space, a Delaunay triangulation is a triangulation such that no point in the set is inside the circumsphere of any simplex in the triangulation (see *e.g.* [17]). The Delaunay triangulation of a discrete set of points located at general positions corresponds to the dual graph of the Voronoi tessellation for this set of points. The original points are called Voronoi sites. The centers of the circumspheres are called Voronoi vertices. Connecting the Voronoi vertices produces the usual Voronoi diagram where each cell is the domain of all the points closer to its Voronoi site than any other site.

We use the Qhull software [1] to compute the 3D Delaunay triangulation of a set of points (Voronoi sites), as well as the Voronoi vertices associated to each site. Qhull implements the Quickhull algorithm for computing the convex hull (see *e.g.* [15]). The problem of finding the Delaunay triangulation of a set of points in three dimensions can be converted to the problem of finding the convex hull of a set of points in a four dimensional space.

Knowing the Voronoi vertices allows us to compute the areas of the common facets shared by pairs of Voronoi cells, as well as the volume of each Voronoi cell. For facets containing more than three Voronoi vertices, we apply again the Qhull algorithm to determine a triangulation (in 2D) of that facet. The area is then computed by summing up the areas of the triangles. Once all the facets areas are known, it is easy to evaluate the volumes of the Voronoi cells by using these areas and the distances between the facets and the Voronoi cells centers, summing up the volumes of the simplices made by each facet and the Voronoi site.

For large scale jobs (10^4 – 10^5 tasks), calculating the whole tessellation on each task would become quite expensive. In practice, this is not required since each parallel task needs to compute only the simplices which include the local Voronoi center. To do that, each task needs to know all the possible neighbor tasks and their Voronoi sites. This may not be possible at the very first step if nothing is known about the distribution of the Voronoi sites. In that case, each parallel task has to compute the geometry of the local Voronoi cell using all the Voronoi sites. Qhull offers the option of computing a limited Delaunay triangulation which include only the facets with vertices including a certain site. Using the Quickhull algorithm however, irrelevant sites may be decimated very quickly and this computation can be done very fast. At any subsequent step however, the local Voronoi tessellation calculation usually involve only of the order of 10 to 30 sites and is extremely fast.

2.3. Implementation

In *ddcMD*, the particle-based domain decomposition scheme does not use spatially implicit communication partners [18]. Instead, each processor maintains a list all the other processors it must communicate with and the particles with which the local particles interact (the communication table). The processors on the list are exactly the ones that own atoms within the interaction cut-off distance of an atom owned by that processor. This list limits communication to as few neighboring processors as possible during a normal molecular dynamics step.

The explicit communication lists allow a simulation to proceed without changing ownership of the atoms each time they cross the boundary of the Voronoi cell of the processor to which they were originally assigned. However, as particles diffuse over time, the range of the communication lists can become unacceptably large. Thus, a reassignment of particles to processors is done periodically to limit communication costs. Each atom is assigned to the processor with the closest domain center (Voronoi site).

To compute the local Voronoi cell and its properties, a task must compute the local Delaunay tessellation (see Section 2.2). For that it needs to know the Voronoi sites of all the adjacent cells it is connected with. In general, the Voronoi sites linked through a Delaunay triangulation to a particular site s on processor P_s may not correspond to the set of processors a particular task need to communicate with to compute atomic interactions. But as long as the list of input sites used to compute the tessellation contains the adjacent sites, it is sufficient.

Measuring load imbalance can be done in various ways. The optimal timer to use for our load balance scheme is likely to depend on the specific problem one tries to simulate and will depend in particular on the relative cost of forces calculation and the average number of particles/task. We have implemented various options to compare load between tasks, such as number of local particles, number of local interacting pairs, measured wall clock time to compute local forces, measured wall clock time minus idle time, etc. For the numerical examples presented in section 3, we estimate the load by measuring wall clock time and subtracting idle time. Idle time is not obvious to measure and is implementation specific. In *ddcMD*, we use two specific barriers and measure the time each task spends waiting at those barriers. Those barriers correspond to a synchronization point after the exchange of interacting particles between adjacent domains and another synchronization point after the force calculation. We also include in the idle time the time each task spends waiting at specific blocking MPI receive calls.

In practice, the load balancing scheme itself is not very computationally intensive. The computation of the local Voronoi structure and the adaptation of the domains centers takes less than 1% of the total computational cost for all the examples we have experimented with. There is however an indirect cost due to the fact that the list of local atoms and interacting atoms belonging to other processors needs to be recomputed after changing the domains. This task

can have a non-negligible cost in some cases, but is mitigated by an appropriate synchronization of load balancing and neighbor table calculation steps, since this task has to be performed at regular intervals anyway.

Note that if load balance is applied at regular intervals like every n steps, while other operations such as calculating neighbor tables are done whenever needed, two consecutive cpu timings measurements may not include the same operations and thus may vary substantially.

Approaches based on wall clock time measurements are capable of taking into account fluctuations from within the simulation due to particle motion as well as from outside the simulation, e.g. uneven processor loads on clustered workstations, or even processor performance on a homogeneous cluster.

3. Numerical Results

We have implemented the algorithm above for the general 3D case usually encountered in MD. We have however adapted it also for quasi-2D problems where one dimension is much smaller than the other two. The domain decomposition is done only in two dimensions for that case. We start by showing results for a quasi-2D problem since it is easier to visualize. Then we apply our algorithm to a finite system (Fe nanowire) and a Lennard-Jones condensation problem.

3.1. Quasi-2D problem: two molten metals in contact

We consider the problem of an interface between two fluids flowing in opposite directions to simulate Kelvin-Helmholtz instability. We consider a physical domain of dimensions $20.1 \times 1254.7 \times 1257.3 \text{\AA}^3$ containing 2×10^6 atoms. The top half ($z > 0$) is composed of aluminium atoms while the bottom half ($z < 0$) is made of copper atoms. Both types of atoms interact with forces derived from an embedded atom model (EAM) potential. [13] Periodic boundary conditions were used in the x - and y -directions, while a static potential was used in the z -direction to confine the atoms.

Load imbalance comes from the fact that liquid copper is more dense than liquid aluminium. This leads not only to a larger number of particles in a given size subdomain, but also to a larger number of interacting pairs for a given potential cutoff radius and consequently more work to compute forces between interacting atoms. The initial and final (after load balancing) Voronoi tessellation are shown in Fig. 2 for an initial $1 \times 16 \times 16$ uniform domain decomposition on 256 processors. Imbalance is not very large for this problem, but load balancing still leads to a reduction in wall clock time of about 10%.

A more stringent test of the algorithm on this system is carried out by starting with an initial Voronoi tessellation obtained from randomly placed Voronoi sites. In that case the initial load imbalance is quite high. Applying our load balance algorithm every 100 MD steps quickly leads to an almost perfect load balancing. Convergence is illustrated in Fig. 3 where the minimum, maximum and average values of the load are plotted as a function of MD steps. For the

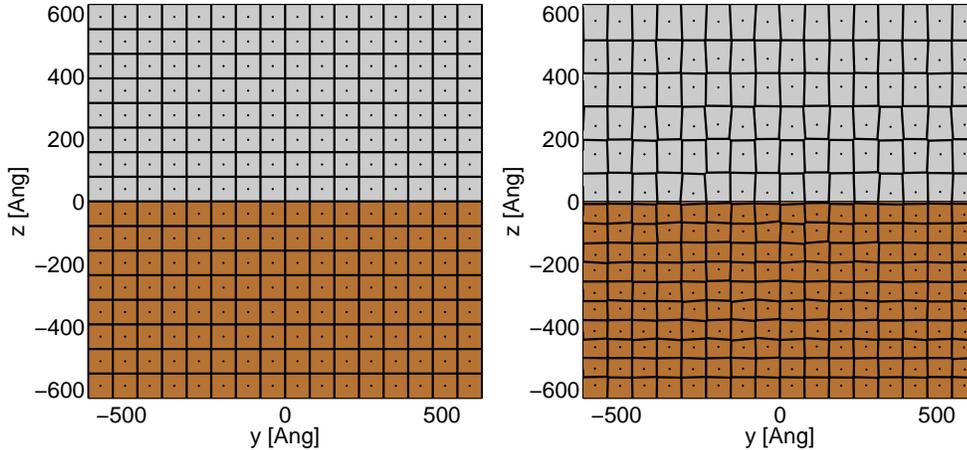


Figure 2: Voronoi cells and sites (in the yz -plane) for Al-Cu two fluids simulation: initial (uniform) domain decomposition (left) and Voronoi cells and sites after load balancing (right).

last data point (MD step 5000), the minimum and maximum values for the load differ from the average value by only 1%. Fig. 4 shows the initial and final tessellations for this run. Although the domain decomposition after balancing is quite different for the two sets of initial Voronoi site (lattice vs. random) the final performance is quite similar with the maximum timing in the random case only about 2% higher than in the lattice case. Thus even if the algorithm may lead to a local minima which depends on the choice of the initial Voronoi tessellation, the timings obtained—at least in this case—are acceptably close to the optimal ones.

3.2. Finite system: Fe nanowire

Finite systems with empty space typically lead to load imbalance if a uniform domain decomposition is used. Some domains contain a large volume of vacuum and not much computational work, while others are densely populated with atoms. We illustrate the performance of our load balancing algorithm for such systems with an application to a Fe nanowires.

We consider a nanowire made of 133280 atoms in a periodic cell of length 200 Angstrom along the wire axis. The length of the computational domain in the other two dimensions is 102 Angstrom. Atoms interaction is modeled with EAM potentials[14].

We performed an MD calculation on 64 processors, starting with a $4 \times 4 \times 4$ uniform domain decomposition and an average number of atoms/task of 2082. With this domain decomposition, the distribution of atoms on tasks has initially a minimum of 680 atoms and a maximum of 2860 per task. After 500 MD steps, applying our load balancing scheme every 100 steps based on measured timings, that spread is substantially reduced, with a minimum number of atoms/task of 1892 and a maximum of 2246. We used $n_{\text{inner}} = 5$ and $\gamma = 20$. The number

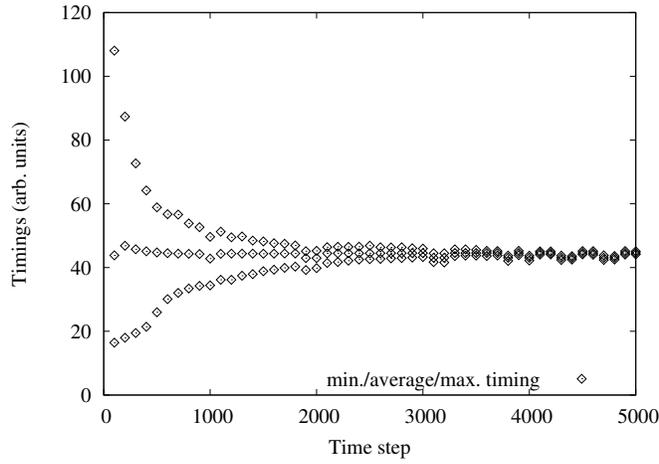


Figure 3: Al-Cu two fluids simulation with 128 MPI tasks: Minimum, maximum and average measured timings over all the MPI tasks starting with randomly placed Voronoi sites. Load balancing algorithm is applied every 100 steps. For this test, $\gamma = 20, n_{\text{inner}} = 5$.

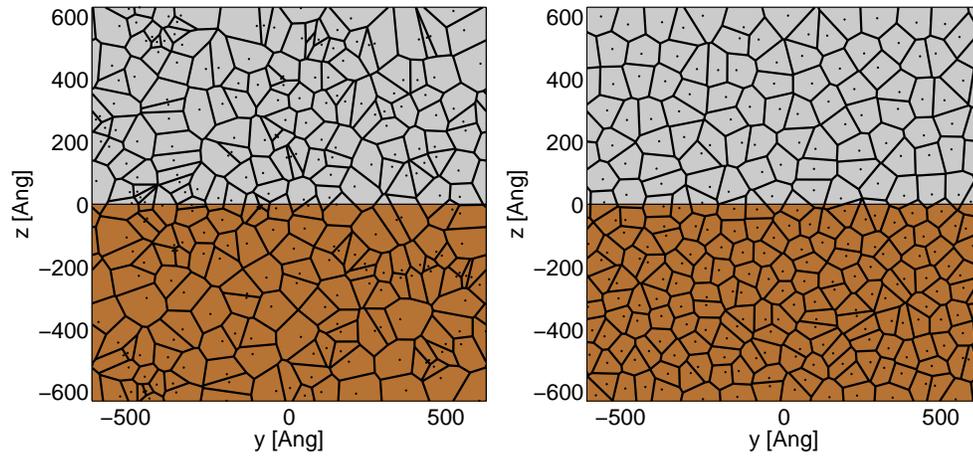


Figure 4: Voronoi cells and sites for Al-Cu two fluids simulation: initial (random) conditions (left) and tessellation obtained after load balancing (right).

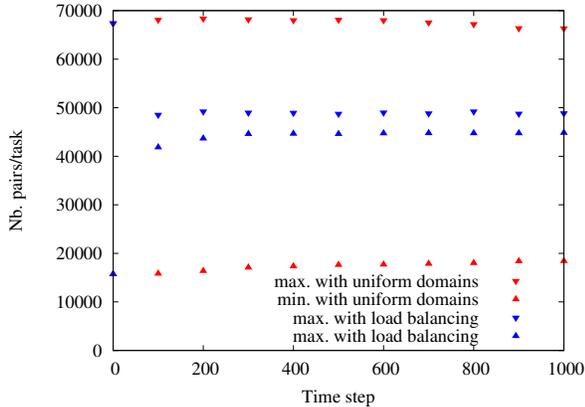


Figure 5: Nanowire simulation: Minimum and maximum number of pairs of interacting atoms computed by each task as a function of MD steps with and without load balancing. The number of local interacting pairs is highly correlated to the computational load of each task.

of interacting pairs of atoms on each task is more closely related to computer time. Its spread also decreases quickly as illustrated in Fig. 5. Fig. 6 shows a cross section of the nanowire with atoms colored according to the domain decomposition. Load balancing leads to a reduction in wall clock time of about 30 %.

3.3. Lennard-Jones condensation

As an illustration of load balancing for a 3D non-uniform MD problem, we consider condensation in a Lennard-Jones gas. The Lennard-Jones potential is defined by

$$\phi(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \quad (14)$$

where r is the distance between two particles (see *e.g.* [7]). The simulations are performed using a shifted truncated potential, $\phi(r) = 0$ for $r > R_c$ and continuous at $r = R_c$, with a cut-off radius of $R_c = 2.5\sigma$. We use the parameters $\epsilon = 0.0104eV$ and $\sigma = 3.405\text{\AA}$ corresponding to Argon, with masses of 40 atomic units.

We setup our simulation with $2.5 \cdot 10^6$ particles placed on a simple cubic lattice in a cubic domain of side length 816 Angstrom, which corresponds to about 0.22 the density of liquid. We then apply a small random displacement to each particles before running a molecular dynamics simulation with a Langevin thermostat set at $T = 81$ K, that is below the boiling temperature. Periodic boundary conditions are used in all three directions. The calculation was carried out using 512 tasks on 256 nodes of a Blue Gene/L computer.

Initially, load imbalance is quite small since the particle distribution is quite uniform. As times goes on, condensation begins and clusters of particles start to form. As more and more particles group together, load imbalance increases.

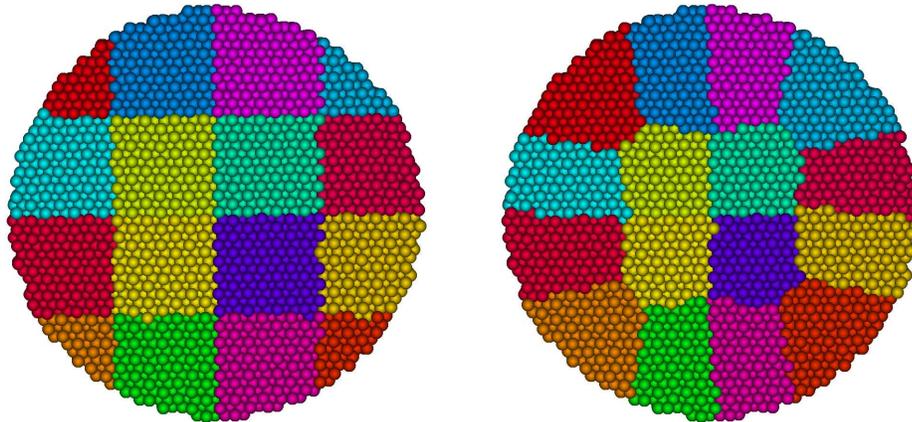


Figure 6: Cross section of nanowire with atoms colored according to subdomain they belong to. Comparison of run with uniform domain decomposition (left) and after applying load balancing algorithm (right).

Fig. 7 shows the minimum and maximum timings measured over the 512 processors involved. A run without load balance is also shown for comparison. After 2.5×10^6 steps, the wall clock time/iteration for the unbalanced calculation is 32% higher than the balanced one. load balancing keeps the minimum and maximum load within 2.5% of each other.

To test our algorithm on a large scale simulation, we extended our Lennard-Jones test problem to 42×10^6 particles and ran a molecular dynamics simulation using the same conditions as described above for the smaller problem. We carried out 10^6 steps using 16384 MPI tasks on 8192 nodes of a Blue Gene/L computer. The results are plotted in Fig. 8. One notices that the spread between the minimum and maximum timers, while still small, is larger than for the smaller scale test. One also notices a “bump” in the maximum time with load balancing at roughly step 900000. A closer look at the number of particles (local and remote) on one of the tasks involved in this bump (task 2835), shows discontinuities in the particle count at the same time (Fig. 8, right). A similar discontinuity can be observed for the volume of the Voronoi cell associated to that task. These discontinuities are an indication that at some point during the simulation, the tessellation was trapped into a local minima of the load balance cost function. As this local minima leads to a situation where load is increasingly unbalanced, the situation becomes unstable and leads suddenly to more drastic moves and reorganizations of the Voronoi tessellation. While such situations may temporarily increase load imbalance the overall effect of the load balance algorithm is still a substantial reduction in wall clock time compared to an unbalanced simulation, keeping imbalance to a very reasonable overhead.

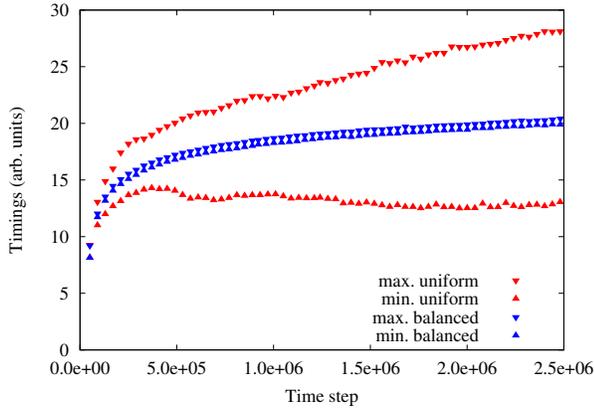


Figure 7: Lennard-Jones condensation problem with $2.5 \cdot 10^6$ particles and 512 MPI tasks: Minimum and maximum timings over all the MPI tasks for unbalanced and balanced cases. For a better display, only one out of every 400 points is shown here. For this test, $\gamma = 10$, $n_{\text{inner}} = 1$.

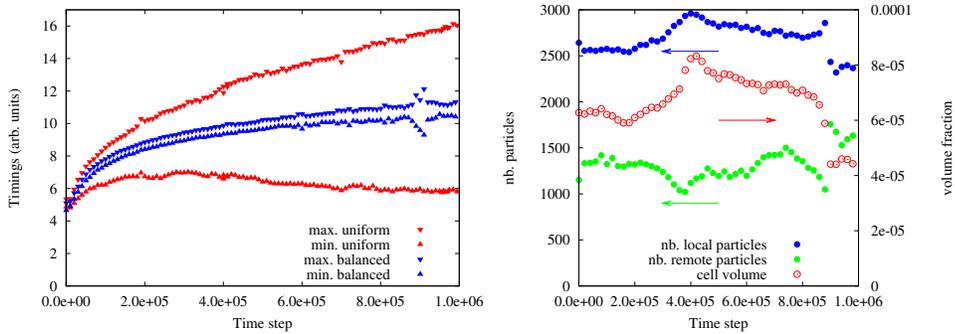


Figure 8: Lennard-Jones condensation problem with $42 \cdot 10^6$ particles and 16384 MPI tasks. Left: Minimum and maximum timings over all the MPI tasks for unbalanced and balanced cases. For a better display, only one out of every 100 points is shown here. For this test, $\gamma = 10$, $n_{\text{inner}} = 1$. Right: number of local and remote particles on task 2835, as well as volume fraction of Voronoi cell associated to that task, as a function of MD step.

3.4. Imperfect Hardware

Imperfect or underperforming hardware can cause a load imbalance even when the system being simulated has a homogeneous density. One example of such problems we have observed involves high rates of correctable DRAM errors on Blue Gene/L and Blue Gene/P nodes. Because the errors are correctable the simulation can continue, however when the rate of such corrections becomes sufficiently high, the time spent performing these corrections can substantially reduce the computing performance of the affected CPU.

To demonstrate the performance of our algorithm in such a case, we ran a simulation on 16,384 Blue Gene/P nodes (65,536 MPI tasks) at a time when there were nodes in the system that were about to be replaced due to a high rate of correctable errors. The simulation consisted of 440×10^6 copper particles with uniform density. Forces were again calculated with an EAM potential [13].

The impact of the failing nodes on the performance of the calculation was profound. Although the number of atoms per task was initially uniform all but two mpi ranks spent over 80% of their time in synchronization barriers waiting for the slow hardware to catch up. The wall clock time per MD time step was initially nearly 3 seconds. After 1,400 time steps with load balancing, the time spent in barriers was reduced from 2.4 seconds per step to less than 0.3 seconds per step, a reduction of 87%. The overall wall time per step was reduced to less than 0.75 seconds per step. While our window of opportunity to run this large scale test was relatively short, it was sufficient to demonstrate how our algorithm was able to quickly reduce imbalance and improve performance by more than a factor of 4 in this case of imperfect hardware.

4. Concluding remarks

We have presented a dynamic load balancing algorithm for classical MD simulations. By moving the Voronoi sites that define each parallel subdomain, excellent load balancing has been obtained for various applications including problems with imbalances caused both by inhomogeneous atom arrangements and by inhomogeneous hardware performance. Overhead is very small and usually well below 1% of the computation time.

One drawback of a diffusion algorithm such as the one we are proposing is the potentially long time it may take to propagate information across the machine. However, it can be argued that this is an initial condition problem only, and that once any initial load imbalance has been accounted for, dynamic particle motion that tends to increase load imbalance should not propagate faster than the particles and thus should be resolved on the fly. Local diffusion processes also have the significant advantage that they are much easier to parallelize on a large number of processors since they involve no centralized load balancing strategy or global communication whatsoever. Also the inner iterations we have introduced allow a much faster redistribution of the load across the processors and achieve a good load balance quickly, even for non-optimal initial domain decompositions. The diffusion process could be further accelerated by integrating our load balancing technique into a multi-level diffusion strategy [9].

We have shown that our strategy is highly scalable, with successful tests on up to 65,536 MPI tasks. Although we occasionally observed some temporary deviation from good load balancing related to local minimas of the cost functional we are trying to minimize, these deviations remain small and the algorithm brings the system back to good load balancing after a short time. In any case, even with these temporary small imbalances, our algorithm leads to a much faster time to solution for non-uniform particle distributions.

5. Acknowledgments

The authors are thankful to Luis A. Sandoval for sharing snapshots from his Fe-nanowire simulations and to Mike Surh for his assistance in performing BG/P simulations.

We also acknowledge the support of the Lawrence Livermore National Laboratory under DOE contract DE-AC52-07NA27344 and LDRD program.

References

- [1] CB Barber, DP Dobkin, and H Huhdanpaa, *The Quickhull algorithm for convex hulls*, ACM Transactions on mathematical software **22** (1996), no. 4, 469–483, <http://www.qhull.org>.
- [2] J. E. Boillat, F. Brug, and P. G. Kropf, *A dynamic load-balancing algorithm for molecular dynamics simulation on multi-processor systems*, Journal of Computational Physics **96** (1991), no. 1, 1 – 14.
- [3] Andrey G. Bronevich and Wolfgang Meyer, *Load balancing algorithms based on gradient methods and their analysis through algebraic graph theory*, J. Parallel Distrib. Comput. **68** (2008), no. 2, 209–220.
- [4] Y.-F. Deng, R.A. McCoy, R.B. Marr, Peierls, R.F., and O. Yasar, *Molecular dynamics on distributed-memory MIMD computers with load balancing*, Applied Mathematics Letters **8** (1995), no. 3, 37–41.
- [5] YF Deng, RF Peierls, and C Rivera, *An adaptive load balancing method for parallel molecular dynamics simulations*, J. Comput. Phys. **161** (2000), no. 1, 250–263.
- [6] Florian Fleissner and Peter Eberhard, *Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection*, Int. J. Numer. Methods Eng. **74** (2008), no. 4, 531–553 (English).
- [7] Daan Frenkel and B. Smit, *Understanding molecular simulation, second edition: From algorithms to applications (computational science)*, 2 ed., Academic Press, 2001.

- [8] Berk Hess, Carsten Kutzner, David van der Spoel, and Erik Lindahl, *GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation*, Journal of Chemical Theory and Computation **4** (2008), no. 3, 435–447 (English).
- [9] G Horton, *A multi-level diffusion method for dynamic load balancing*, Parallel Computing **19** (1993), no. 2, 209 – 218.
- [10] M. Iri, K. Murota, and T. Ohya, *A fast voronoi-diagram algorithm with applications to geographical optimization problems*, Lecture Notes in Control and Information Science, vol. 59, 1984, Proc. 11th IFIP Conf. on System Modelling and Optimization, pp. 273–288.
- [11] R Koradi, M Billeter, and P Guntert, *Point-centered domain decomposition for parallel molecular dynamics simulation*, Comput. Phys. Commun. **124** (2000), no. 2-3, 139–147.
- [12] Sameer Kumar, Chao Huang, Gengbin Zheng, Eric Bohm, Abhinav Bhatele, James C. Phillips, Hao Yu, and Laxmikant V. Kalé, *Scalable Molecular Dynamics with NAMD on Blue Gene/L*, IBM Journal of Research and Development: Applications of Massively Parallel Systems **52** (2008), no. 1/2, 177–187.
- [13] DR Mason, RE Rudd, and AP Sutton, *Stochastic kinetic Monte Carlo algorithms for long-range Hamiltonians*, Comput. Phys. Commun. **160** (2004), no. 2, 140–157.
- [14] R. Meyer and P. Entel, *Martensite-austenite transition and phonon dispersion curves of $\text{Fe}_{1-x}\text{Ni}_x$ studied by molecular-dynamics simulations*, Phys. Rev. B **57** (1998), 5140–5147.
- [15] Ernst Mücke, *Quickhull: Computing convex hulls quickly*, Computing in Science and Engineering **11** (2009), 54–57.
- [16] A Nakano and T Campbell, *An adaptive curvilinear-coordinate approach to dynamic load balancing of parallel multiresolution molecular dynamics*, Parallel Computing **23** (1997), no. 10, 1461–1478 (English).
- [17] Franco P. Preparata and Michael I. Shamos, *Computational geometry: an introduction*, Springer-Verlag New York, Inc., New York, NY, USA, 1985.
- [18] Frederick H. Streitz, James N. Glosli, Mehul V. Patel, Bor Chan, Robert K. Yates, Bronis R. de Supinski, James Sexton, and John A. Gunnels, *Simulating solidification in metals at high pressure: The drive to petascale computing*, SciDAC 2006: Scientific Discovery Through Advanced Computing (Tang, WM, ed.), Journal of physics conference series, vol. 46, 2006, 2nd Annual Scientific Discovery Through Advanced Computing Conference (SciDAC 2006), Denver, CO, JUN 25-29, 2006, pp. 254–267.

- [19] V Zhakhovskii, K Nishihara, Y Fukuda, S Shimojo, T Akiyama, S Miyanaga, H Sone, H Kobayashi, E Ito, Y Seo, M Tamura, and Y Ueshima, *A new dynamical domain decomposition method for parallel molecular dynamics simulation*, 2005 IEEE International Symposium on Cluster Computing and the Grid, Vols 1 and 2 (New York, NY USA), IEEE, 2005, 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, Wales, May 2005, pp. 848–854.