Commun. Comput. Phys.
doi: 10.4208/cicp.140810.021210a

Vol. **10**, No. 4, pp. 1071-1088
October 2011

Parallel Molecular Dynamics with Irregular Domain Decomposition

Mauro Bisson[1,*], Massimo Bernaschi[2] and Simone Melchionna[3,4]

[1] *Department of Computer Science, University of Rome "Sapienza", Italy.*
[2] *Istituto Applicazioni Calcolo, Consiglio Nazionale delle Ricerche, Rome, Italy.*
[3] *Institute of Material Sciences and Engineering, École Polytechnique Fédérale de Lausanne, Switzerland.*
[4] *CNR-IPCF, Istituto Processi Chimico-Fisici, Consiglio Nazionale delle Ricerche, Rome, Italy.*

Received 14 August 2010; Accepted (in revised version) 2 December 2010

Available online 7 July 2011

Abstract. The spatial domain of Molecular Dynamics simulations is usually a regular box that can be easily divided in subdomains for parallel processing. Recent efforts aimed at simulating complex biological systems, like the blood flow inside arteries, require the execution of Parallel Molecular Dynamics (PMD) in vessels that have, by nature, an irregular shape. In those cases, the geometry of the domain becomes an additional input parameter that directly influences the outcome of the simulation. In this paper we discuss the problems due to the parallelization of MD in complex geometries and show an efficient and general method to perform MD in irregular domains.

PACS: 02.70.Ns

Key words: Molecular dynamics, irregular domain decomposition, parallel algorithms, hemodynamics.

1 Introduction

Molecular Dynamics (MD) is a very popular simulation method to study many-body systems by looking at the motion of N individual particles. In essence, MD tracks the motion of particles whose trajectories are the result of forces mutually exerted among them. The temporal propagation of the particle positions obeys Newton's equations of motion, by applying a time-discretization procedure of the differential equations followed by an integration in time [1,2].

*Corresponding author. *Email addresses:* bisson@di.uniroma1.it (M. Bisson), massimo@iac.rm.cnr.it (M. Bernaschi), simone.melchionna@epfl.ch (S. Melchionna)

Since its inception, MD has benefited from several algorithmic advances that nowadays permit the simulation up to billions of particles with $\mathcal{O}(N)$ complexity. Multiple techniques for Parallel Molecular Dynamics (PMD) have been put forward over the years. In particular, PMD has now reached a high degree of efficiency when dealing with regular geometries, that is, with bulk systems whose computational domain can be subdivided in terms of cubes, slabs, or other regular subdomains. Several implementations of PMD are freely available and run on both supercomputers, such as the IBM Blue Gene, and high-performance commodity hardware, such as clusters of Graphical Processing Units (GPUs). Among the most popular packages are NAMD [3] and LAMPPS [4], softwares that are known to scale over thousands of processors. Moreover, we recall ACEMD, a production bio-molecular dynamics software specially optimized to run on NVIDIA GPUs [5]. Finally, AMBER [6] is another molecular simulation program featuring NVIDIA GPU acceleration support [7].

Recently, there has been growing interest in employing MD for the multi-scale simulation of particles suspended in a fluid. In the multi-scale framework, solute particles are handled according to the conventional MD scheme whereas the solvent is handled by means of conventional fluid dynamics solvers, such as the popular Lattice Boltzmann (LB) method. The non-conventional aspect of the multi-scale approach enters in the coupling between scales, an aspect that takes into account the physical level of both the solute, the solvent and their mutual interaction. In the case of the LB-MD multi-scale system, such design involves the kinetic level to account for the microdynamics of the solvent. As a result, the LB-MD method enjoys the same $\mathcal{O}(N)$ complexity of stand-alone MD for systems with uniform distribution of solute particles.

In the last few years, our group has been devising and deploying multi-scale methods to study the transport of molecular systems [9] and, more recently, the suspension of red blood cells, an important topic in computational hemodynamics [10]. The latter constitutes a strategic field since it allows to understand the physical behavior of blood from a bottom-up standpoint, that is, by following the motion of red blood cells and plasma. The wealth of information accessible from the multiscale approach and the ensuing biomedical implications are beyond question.

When simulating large cardiovascular systems, the typical geometrical layout consists of several interconnected blood vessels spreading in space with an irregular pattern. Consequently, a parallel algorithm for both the LB and MD components needs to account for the geometric sparsity of the vasculature. The optimal approach to parallelism is to decompose the computational space into subdomains where the fluid and the particles are handled on the same footing. In this way, the solution of the fluid-dynamic equations, the calculation of inter-particles and fluid-particle interactions are mostly local on the processor responsible for the subdomain.

A possible approach to handle complex cardiovascular systems could be, by analogy with large scale stand-alone PMD in a simple regular box, to use a decomposition into box-shaped subdomains. Such approach is highly discouraged since it leads to poorly balanced subdomains, both in terms of number of active (from the fluid dynamics view-

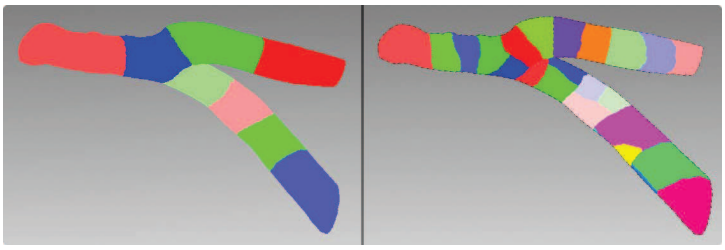


Figure 1: Example of domain partitioning as obtained by the SCOTCH software [12]. The domain represents a bifurcation of a human artery partitioned in 8 (left panel) and 24 (right panel) subdomains.

point) mesh points and average number of red blood cells. The end result would be a rapid degradation of performances on highly scalable architectures for both the LB and the MD components. The demand for load-balanced subdomains requires to decompose the highly irregular geometry by means of more sophisticated partitioning methods.

Several tools are available to achieve high-quality domain decompositions. In particular, we refer to tools that we customarily employ in our research, such as the *METIS* [11] and *SCOTCH* [12] packages. Both tools do not consider the geometrical shape of the domain, but rather employ the LB connectivity graph of the underlying mesh in order to construct a graph-based partitioning according to some heuristics (e.g., recursive multilevel bisection). The typical appearance of a domain partitioning is shown in Fig. 1 for the case of an arterial system subdivided in a small number of domains (8 and 24).

In presence of highly-irregular domains, several critical issues arise related to the calculation of forces and migration of particles among subdomains. For instance, irregular subdomains imply irregular contact surfaces and, in principle, irregular communication patterns. The geometrical tests for particle ownerships and exchange of particles between domains require strategic decisions that affect the efficiency of stand-alone MD as much as the LB-MD multi-scale method.

The purpose of the present work is to describe a novel PMD method for generic irregular subdomains that features the same $\mathcal{O}(N)$ capability of PMD for regular decompositions. The proposed method relies on two basic notions, proximity and membership tests. These tests are used to discriminate particles according to their positions relative to the geometry of the domains. Proximity tests are used to select the particles that have out-of-domain interactions and are used to perform inter-domain forces computation. The membership tests regard the assignment of particles to domains and exploit a tracking method to associate particles position to the domains' morphology. Armed with these tools, the different stages of the PMD algorithm can be derived. Even if our primary purpose is to define a computational strategy for hemodynamics, we underscore that the present treatment of irregular domains is usable for more microscopic systems. A typical scenario of sparsely distributed computational domain is the simulation of atomic or molecular systems confined in porous environments. Besides that, it is important to realize that our approach is completely independent from the partitioning strategy. As an example, other hemodynamic codes, as for instance HemeLB [8], even if adopting a dif-

ferent partitioning technique (a variant of the so-called Graph Grow Partitioning), could implement our technique to include the motion of red blood cells.

In this paper we illustrate the general aspects of the PMD and the specifications of these aspects for our simulations of coupled LB-MD. We focus on simulations of particles without topological connectivity, as e.g., in the case of molecules. On the other hand, our treatment can be extended to molecules in irregular domains without major modifications. In a forthcoming paper we will describe the practical implementation of these notions for multi-CPU/multi-GPU clusters.

2 Molecular dynamics

Molecular Dynamics is a general term that indicates the numerical technique for the simulation of a wide range of physical phenomena, from the atomic to the cellular scale, in presence of interacting particles. On general grounds, we define a system of N particles by using the set of particle positions $R \equiv \{\vec{r}\}_{i=1,N}$ and forces $F \equiv \{\vec{f}\}_{i=1,N}$ acting on them. Several physical quantities are further associated to the particles, such as their masses, velocities and other quantities. For instance, in the case of red blood cells, particles are bodies having a moment of inertia, angular velocity, orientation that interact via mutual forces and torques.

The physical model considered here is encoded by the interparticle pairwise forces that depend only on particle positions, such that the force acting on particle i is given by

$$\vec{f}_i = \sum_{j(\neq i)} \vec{f}_{ij},$$

where $\vec{f}_{ij} = \vec{f}_{ij}(r_{ij})$ and with r_{ij} being the distance between particles i and j. Hereafter we assume that forces are short-ranged, so that there is a cutoff $r_{\max}$, such that $\vec{f}_{ij} = 0$ for $r_{ij} > r_{\max}$. Besides the interparticle interactions, an additional force avoids particles from crossing the confining walls. This is achieved by treating the wall nodes as fictitious fixed particles, that act on the moving particles via a short-ranged repulsive interaction.

The simulation method consists of a step-by-step numerical solution of the classical equations of motion. At each step, forces acting on the particles are computed and the new state of the system is updated by integrating Newton's law of motion. There are three main components in a MD program:

1. A model for the interaction among system constituents (atoms, molecules, etc). It specifies the physical observables by which particles are represented and the force exercised between pairs.
2. An integrator, which propagates particle positions and velocities from time t to $t+\delta t$. Usually, it is a finite difference scheme which moves trajectories discretely in time.
3. A statistical ensemble, where physical observables like pressure, temperature or the number of particles are controlled. This is used at each simulation step to verify that

the system evolves in a correct way and, at times, to decide when the simulation ends (for example when a quantity reaches a given threshold).

These steps essentially define the MD simulation. From a computational viewpoint they specify the calculations executed at each step of the simulation to compute the forces acting on the particles, to advance their dynamic quantities in time and to decide when the simulation should end. Clearly, different phenomena may require a different implementation of these steps.

In the following, we discuss computational problems common to most MD implementations regardless of the details of the physics involved. We consider an abstract physical system that can represent the most common models used in MD simulations. The implementation of the forces and the integration step will be abstracted with generic function calls. Finally, the simulation is supposed to run as long as the system verifies a generic property P which represents the statistical ensemble of finite size.

2.1 Parallel MD

The parallelization of Molecular Dynamics requires the partitioning of the simulation system into subdomains, each assigned to a different processor. Traditionally, two different strategies are used, particle decomposition (PD) and domain decomposition (DD). In the first strategy, particles are assigned to processors according to an ordering index (given N particles and K processors, each processor receives N/K particles, regardless of their position) whereas in the second strategy the assignment is based on the position of particles. In large scale systems DD provides a better solution since it ensures a high degree of local operations on each processor. The reason is apparent: particles assigned to a processor are spatially close to each other and thus most of pair interactions are intradomain. In the case of PD, on the other hand, spatial locality of particles is not guaranteed. In the case of DD, particle motion does not affect the average number of local interacting pairs while can have a strong impact in case of PD. Lastly, another advantage of DD is to keep local any operation between different physical methods coupled in the same simulation. This is the case of multi-scale simulations where particles interact with an underlying fluid (or solvent) whose dynamics is described, for instance, by means of the Lattice Boltzmann method. In the following we consider DD as the reference partitioning scheme and describe some critical issues that arise in presence of irregular subdomains along with our proposal to overcome them. The end result is that parallel MD simulations can run with high efficiency also in presence of generic partitioning.

The parallelization of a MD code, while allowing to scale the size of the system by using multiple processors, requires the solution of, at least, two problems related to interdomain pairs and particle migration.

Interdomain pairs are pairs of particles located in different subdomains (processors) at distance smaller than the cutoff $r_{\max}$. In order to compute interdomain forces, a processor requires to fetch information about all external particles that are close enough to interact with its own particles. Similarly, each processor must identify the set of particles

in its subdomain that can interact with those belonging to neighboring processors, and exchange such set. We identify the sequence of operations required to handle interdomain pairs as "frontier management".

In a similar way, once particle positions are updated, particles that depart from a subdomain need to be sent to the processors in charge of the destination subdomains. This requires that processors identify their own departing particles and exchange them with neighbors. The sequence of operations that handle particle migration is identified as "migration management".

We define the simulation system as $S=(D,R^0,P,r_{\max})$ where D is the spatial domain in which particles move, R^0 is the set of particles positions at time $t=0$, $P=\{p_1,\cdots,p_n\}$ is a set of processors and $r_{\max}$ is the cutoff distance for the forces. The parallel MD code can be represented as a couple of algorithms, (A_{DEC},A_{PMD}). A_{DEC} is a serial algorithm executed only once, before the simulation starts, to perform the domain decomposition and to identify the subdomains containing putative interacting particles. A_{PMD} is a parallel algorithm executed by every processor to perform the MD simulation. A_{DEC} takes as input the system S and produces a decomposition:

$$\{(D_1,R_1^0,N_1),(D_2,R_2^0,N_2),\cdots,(D_n,R_n^0,N_n)\},$$

where $D_1,\cdots,D_n$ and $R_1^0,\cdots,R_n^0$ are partitions of the spatial domain D and the initial positions R^0, respectively, and $N_1,\cdots,N_n$ is a subset of P. D_i is the spatial subdomain assigned to processor p_i, R_i^0 is the set of particles with initial position inside D_i and N_i is the set of processors whose subdomains are at distance smaller than $r_{\max}$ from any point of D_i.

In our LB-MD simulations the domain D is identified by the underlying LB mesh used for the simulation of the solute surrounding the particles. The mesh has an irregular shape (embedded in a much larger *bounding box*) and typically covers the extension of multiple intersecting vessels in extended hemodynamics simulations. Let us define a mesh by the set of cartesian points $\mathcal{M}\equiv\{\mathbf{n}\}\subset\mathbb{N}^3$. The spatial domain D in which particles move is the set of points at distance at most 0.5 (the mesh spacing step is conventionally chosen as 1) from a point in $\mathcal{M}$. What we formalized as A_{DEC}, is implemented as follows. In the first step, a graph partitioning tool (e.g., SCOTCH [12]) is run on the mesh $\mathcal{M}$ in order to produce a partition $\mathcal{M}_1,\cdots,\mathcal{M}_n$. The set of particles is then divided by assigning to partition i all particles in the subdomain identified by $\mathcal{M}_i$ (more about that later in this section). As a final step, we need to identify the set N_i, a non-trivial task that in principle requires an awkward number of geometrical tests. The problem is that, with subdomains of arbitrary shape, the particles in two subdomains may be in interaction even if their surfaces are not in direct contact (for instance one can imagine two subdomains in 2D separated by a long thin stretch of a third subdomain). A possible solution, specific to our case, could be to leverage the granularity of each subdomain and compute the relative distance between cartesian mesh points belonging to all subdomains. These tests require to broadcast all mesh points among processors and perform operations with a computational cost of $\mathcal{O}(n^2)$, with n the number of mesh points. However, we have devised a method to determine domains in interaction based on a reliable, yet simple

Algorithm 1 Identification of subdomains within the cutoff distance.

Require: $r_{\max}$
Require: $x_m, x_M, y_m, y_M, z_m, z_M$
1: $N_i \leftarrow \emptyset$
2: $Gather(x[], x_m - r_{\max}, X[], x_M + r_{\max})$
3: $Gather(y[], y_m - r_{\max}, Y[], y_M + r_{\max})$
4: $Gather(z[], z_m - r_{\max}, Z[], z_M + r_{\max})$
5: **for** $j = 1$ **to** n **do**
6: **if** $j == i$ **continue**
7: **if** $(x_M < x[j])$ || $(x_m > X[j])$ **continue**
8: **if** $(y_M < y[j])$ || $(y_m > Y[j])$ **continue**
9: **if** $(z_M < z[j])$ || $(z_m > Z[j])$ **continue**
10: $N_i \leftarrow N_i \cup \{j\}$
11: **end for**

criterion, that does not require such massive circulation of data among processors and can be performed in full generality. In Algorithm 1, we describe the operational procedure executed by processor p_i to construct the set N_i in the practical implementation of the method. We assume that every processor can compute the bounding box of its subdomain. This substep requires just a single scan of the mesh $\mathcal{M}_i$.

At first, each processor initializes the set N_i and broadcasts to the other processors the bounding box of its domain augmented by the cutoff distance $r_{\max}$ (lines 1-4). As a result of the gathering function, the minimum and maximum coordinates of the boxes are stored in vectors indexed by the processor id, named with lowercase and uppercase letters, respectively. Now, processor p_i loops through the vectors to find the boxes that have a non-empty intersection with its own, non-augmented, box. The processor ids corresponding to the indices of the intersecting boxes are then added to the set N_i. Although this procedure may generate a superset of the actual neighbors, we consider it an acceptable tradeoff given the limited computational effort it requires.

Once the initial stage of the domain decomposition and identification of interacting subdomains is completed, the parallel algorithm A_{PMD} is executed by every processor p_i on the set (D_i, R_i^0, N_i).

Algorithm 2 shows the pseudocode for A_{PMD} that implements the MD method on the subdomain D_i. The first half of the loop implements frontier management (lines 3-15). In the first part (lines 3-9) particles at a distance less than $r_{\max}$ from the surface of D_i, $S(D_i)$, are selected and transmitted to the neighboring processors (line 9). The second part handles external particles received from neighbors (lines 10 to 15). For each received particle, its distance from D_i is first checked (line 12). If it is greater than $r_{\max}$ the particle does not interact with the subdomain and is discarded (line 13). After this check, set E contains the received particles that putatively interact with the internal ones.

Computation of forces and integration of Newton's equation of motion are performed

Algorithm 2 Parallel MD algorithm.

Require: $r_{\max}$, (D_i, R_i^0, N_i)

1: $t \leftarrow 0$
2: **while** not P **do**
3: $S \leftarrow \emptyset$
4: **for each** $\overrightarrow{r}$ **in** R_i^t **do**
5: **if** $d(\overrightarrow{r}, S(D_i)) \leq r_{\max}$ **then**
6: $S \leftarrow S \cup \{\overrightarrow{r}\}$
7: **end if**
8: **end for**
9: $S \rightarrow mcast(N_i)$
10: $E \leftarrow recv(N_i)$
11: **for each** $\overrightarrow{r}$ **in** E **do**
12: **if** $d(\overrightarrow{r}, S(D_i)) > r_{\max}$ **then**
13: $E \leftarrow E \setminus \{\overrightarrow{r}\}$
14: **end if**
15: **end for**
16: $R_i^{UP} \leftarrow ForcesAndIntegrate(R_i^t, E)$
17: $S \leftarrow \emptyset$
18: **for each** $\overrightarrow{r}$ **in** R_i^{UP} **do**
19: **if** $\overrightarrow{r} \notin D_i$ **then**
20: $R_i^{UP} \leftarrow R_i^{UP} \setminus \{\overrightarrow{r}\}$
21: $S \leftarrow S \cup \{\overrightarrow{r}\}$
22: **end if**
23: **end for**
24: $S \rightarrow mcast(N_i)$
25: $E \leftarrow recv(N_i)$
26: **for each** $\overrightarrow{r}$ **in** E **do**
27: **if** $\overrightarrow{r} \in D_i$ **then**
28: $R_i^{UP} \leftarrow R_i^{UP} \cup \{\overrightarrow{r}\}$
29: **end if**
30: **end for**
31: $R_i^{t+1} \leftarrow R_i^{UP}$
32: $t \leftarrow t+1$
33: **end while**

by the function call $ForcesAndIntegrate(R_i^t, E)$ that returns the set of updated particle positions R_i^{UP} (line 16). Although the actual implementation depends on the particular physical model under consideration, particles within set E should only be considered for their influence over internal particles, i.e., interaction pairs should be searched in $R_i^t \times (R_i^t \cup E)$. After the position of internal particles has been updated, particle migration

is handled in the second half of the main loop (lines 17 to 32). All particles in the updated set R_i^{UP} that moved outside D_i are removed from the set (line 20), collected together (line 21) and then transmitted to the neighboring processors for possible admission (line 24). Lastly, particles received from neighboring processors (line 25) are scanned to identify new entries and each particle that moved inside D_i is added to the set R_i^{UP}(line 28). Finally R_i^{UP} is assigned to R_i^{t+1} and a new iteration starts.

The efficiency of PMD is strongly influenced by the frontier and migration management (lines 4-15 and 18-30) that can be measured by looking at the number of executions of two basic tests on particles:

- Proximity test: $d(\overrightarrow{r}, S(D_i)) \leq^? r_{\max}$
- Membership test: $\overrightarrow{r} \in^? D_i$

These tests, in turn, depend heavily on the representation of the domain and usually the efficiency of their implementation is tied to the "degree" of regularity of the spatial domain. For regular subdomains, like cubes or parallelepipeds, these tests can be easily implemented as computations of distances between points and planes, or, in case of axes-alignment, as simple differences. However, when there is no regularity, these tests can become time consuming and an effort to limit the number of executions is highly desirable. For example, Algorithm 2 is quite inefficient since it performs the tests on all particles at every iteration.

When the domain is represented by a Cartesian mesh $\mathcal{M}$, the membership test can be written as:

$$\overrightarrow{r} \in D_i \Longleftrightarrow round(\overrightarrow{r}) \in \mathcal{M}_i\,,$$

where $round(\overrightarrow{r})$ represents the vector whose components are obtained by rounding to the nearest integer the real components of vector $\overrightarrow{r}$. At first sight, due to the irregularity of the mesh, this operation requires scanning all mesh points in $\mathcal{M}_i$ with a linear search method. Actually, an efficient search can be performed by using a binary search algorithm or by a hash table. In our case, we store the mesh as a compact, one-dimensional array and sort the mesh points in ascending order according to a 1D-index equal to $k*nx*ny+j*nx+i$, where nx,ny,nz define the *bounding box*. The binary search has logarithmic cost so that the membership test is much more affordable from the computational viewpoint. For what concerns the proximity test, it can be viewed as another membership test run on the subset of $\mathcal{M}_i$ that covers the frontier region of D_i.

Another crucial issue that influences the efficiency is the communication scheme used to exchange particles among processors in both the frontier and migration management steps. For these steps, there are two possibilities:

1. multicast transmissions;
2. point-to-point transmissions.

In the multicast scheme (used in Algorithm 2) each processor sends groups of particles to all neighbor processors whereas, in the point-to-point scheme, a processor sends to

each neighbor only its particles that interact with those in the domain of the receiver. From the communication viewpoint, the latter has the clear advantage of imposing much less overhead on the network since processors receive only data strictly required. Nevertheless, to implement such a communication scheme it is necessary for the processors to associate frontier and migrated particles to external domains. This requires multiple executions of proximity and membership tests for each neighboring subdomain which, in turn, requires knowledge of the geometry of the neighbors, typically consisting of a region of size $r_{\max}$. This operation results in an increase of memory requirements per processor and, more importantly, in a greater number of tests performed per particle.

In the multicast approach, as shown in Algorithm 2, processors do not need to know the geometry of neighboring subdomains but received particles must be processed to identify those that are of interest for the receiver.

To summarize, the differences between the two approaches are:

1. multicast: higher network overhead, only-local geometry required, simple test implementation;
2. point-to-point: lower network overhead, non-local geometry handling, complex test implementation.

These are general considerations and there may be exceptions. For instance, in the most favorable case of a regular Cartesian decomposition, each processor easily identifies the subdomains to which a given particle needs to be sent by using only its position and the global decomposition rules. For the regular case it is possible to combine the advantages of both approaches, point-to-point transmissions by using only local geometry and efficient tests implementations.

Algorithm 3 Frontier management algorithm according to the point-to-point model.

$S_j \leftarrow \emptyset$, $\forall j \in N_i$
for each $\overrightarrow{r}$ **in** R_i^t **do**
  for each j **in** N_i **do**
    if $d(\overrightarrow{r}, S(D_j)) \leq r_{\max}$ **then**
      $S_j \leftarrow S_j \cup \{\overrightarrow{r}\}$
    end if
  end for
end for
for each j **in** N_i **do**
  $S_j \rightarrow send(j)$
end for
for each j **in** N_i **do**
  $E_j \leftarrow recv(j)$
end for
$E \leftarrow \bigcup_{j \in N_i} E_j$

Algorithm 4 Migration management algorithm according to the point-to-point model.

$S_j \leftarrow \emptyset$, $\forall j \in N_i$
for each $\overrightarrow{r}$ **in** R_i^{UP} **do**
  if $\overrightarrow{r} \notin D_i$ **then**
    for each j **in** N_i **do**
      if $\overrightarrow{r} \in D_j$ **then**
        $S_j \leftarrow S_j \cup \{\overrightarrow{r}\}$
        $R_i^{UP} \leftarrow R_i^{UP} \setminus \{\overrightarrow{r}\}$
      end if
    end for
  end if
end for
for each j **in** N_i **do**
  $S_j \rightarrow send(j)$
end for
for each j **in** N_i **do**
  $E_j \leftarrow recv(j)$
end for
$R^{t+1} \leftarrow R_i^{UP} \cup \bigcup_{j \in N_i} E_j$

Algorithms 3 and 4 implement the frontier and migration management sections of Algorithm 2 for the point-to-point solution.

In the following section we introduce a domain decomposition scheme that, applied to the subdomains, allows to substantially reduce the number of proximity and membership tests performed at each iteration.

2.2 Cell tiling

We present now a general technique to reduce the number of proximity and membership tests performed at each iteration, that does not depend on the representation and shape of the subdomains. Moreover, this approach allows to implement proximity tests as table lookups at the cost of a limited overestimation of the number of frontier particles.

As anticipated, our simulations make use of subdomains of arbitrarily complex geometry and a decomposition algorithm that is executed only once, before the simulation starts. It produces subdomains that require a new formulation of the proximity tests. Due to the lack of regularity in the subdomains, we employ the multicast scheme for the communications and design a more general scheme for the proximity and membership tests.

The basic idea is to approximate the critical regions around the contact surfaces of the subdomains in such a way that is computationally simple to find a superset of the particles located inside those regions and to apply the tests only to those particles. This

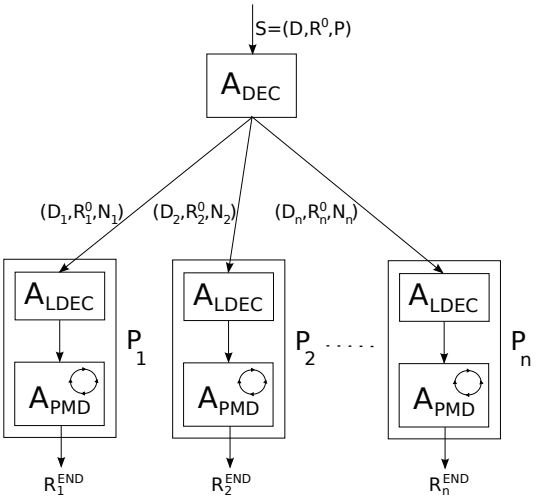


Figure 2: Initially the domain decomposition algorithm A_{DEC} is run on the simulation system to partition the domain in subdomains. Each processor then executes the tiling algorithm A_{LDEC} on its subdomain and, finally, the iterative phase of the simulation starts by running the parallel MD algorithm A_{PMD}.

is possible by covering each subdomain with identical box-shaped cells. The cells tile a larger region as compared to the original subdomain extension. The tiling is computed by the algorithm A_{LDEC} (Fig. 2) that processor p runs on its subdomain D_i to produce the tiling (C,I_c,F_c,E_c), where $C=\{C_1,\cdots,C_k\}$ is the associated set of cells, $I_c\subseteq[k]$, $F_c\subseteq[k]$ and $E_c\subset[k]$ represent, respectively, the set of internal, frontier and external cells. Sets I_c, F_c and E_c form a partitioning of C.

The tiling verifies the following properties:

1. Every point of D_i is within either an internal or a frontier cell.
2. Internal cells contain only points of D_i at distance greater than $r_{\max}$ from the domain boundary.
3. Frontier cells contain all points of D_i at distance less than or equal to $r_{\max}$ from the domain boundary.
4. External cells contain only points outside D_i.
5. All external points at distance less than or equal to $r_{\max}$ from the domain boundary lie within either an external or a frontier cell.

The tiling is built in the following way. The bounding box of D_i, having size $n_x\times n_y\times n_z$, is initially divided in $m_x\times m_y\times m_z$ identical, regular cells, such that

$$m_x=\left\lfloor\frac{n_x}{r_{\max}}\right\rfloor,\quad m_y=\left\lfloor\frac{n_y}{r_{\max}}\right\rfloor,\quad m_z=\left\lfloor\frac{n_z}{r_{\max}}\right\rfloor.$$

As a consequence, all cells have the same size $c_x\times c_y\times c_z$, that verifies the following conditions

$$c_x=\frac{n_x}{m_x}\geq r_{\max},\quad c_y=\frac{n_y}{m_y}\geq r_{\max},\quad c_z=\frac{n_z}{m_z}\geq r_{\max}.$$

As a result, the bounding box is divided in the maximum number of cells whose size is greater or equal to $r_{\max}$ in each direction. The pool of cells is then augmented by including a layer of cells external to the bounding box (Fig. 3(a)) and to each cell is assigned a unique identifier. This pool of cells forms the initial set C from which sets I_c, F_c and E_c are subsequently built. Initially, all cells containing at least one point of D_i are selected (Fig. 3(b)). From those cells, sets F_c and I_c are built by assigning to F_c the cells that contain at least one point at distance less than or equal to $r_{\max}$ from the frontier of D_i, and to I_c the remaining set (Fig. 3(c)). Finally, the unselected cells of C are scanned, assigning to E_c the cells with a neighbor in F_c. The remaining cells are then discarded from C (Fig. 3(d)). There are a number of issues in the process of building E_c that are worth mentioning. Choosing external cells based on the proximity of frontier cells may lead to picking cells that contain only external points at distance greater than $r_{\max}$ from D_i. Nevertheless, this construction guarantees that every frontier cell has all 27 neighbors in C. Moreover, when building E_c, the case of periodic boundary conditions needs to be properly managed while searching for neighboring frontier cells. In case of a periodic domain, for each periodic dimension, if the subdomain fully extends along that direction, then the searched neighborhood must account for the periodicity. Fig. 4 shows an example of tiling of a subdomain with periodicity along the horizontal axis.

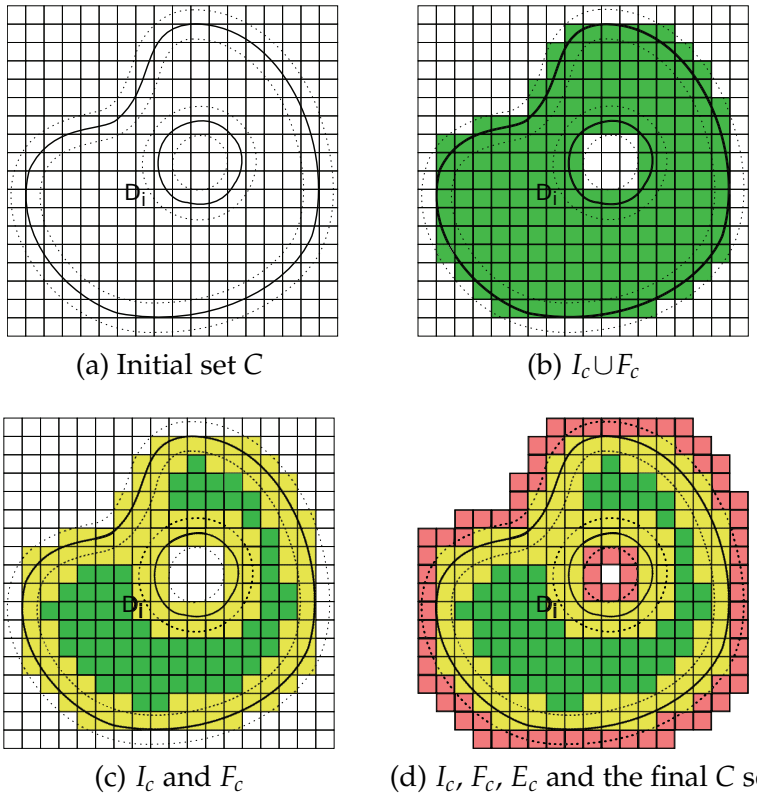
Algorithms 5 and 6 show two pseudocodes for frontier and migration management that rely on the subdomain tiling.

We assume that Algorithm 5 receives the particles binned into the cells. In the first two lines of Algorithm 5 frontier particles are sent (in multicast) to neighboring processors. This is done by simply transmitting all particles inside frontier cells. External particles received from neighbors must be checked for putative interdomain interacting pairs. For each received particle the index of the containing cell is computed (line 5). If it identifies an external or a frontier cell (line 6) then the particle is inserted into the cell (line 7), otherwise it is ignored.

The proximity tests required to spot frontier particles that need to be transmitted are

Algorithm 5 Frontier management using the subdomain tiling.

Require: N_i
Require: C, I_c, F_c, E_c
1: $S \leftarrow \bigcup_{cid \in F_c} C_{cid}$
2: $S \rightarrow mcast(N_i)$
3: $E \leftarrow recv(N_i)$
4: **for each** $\overrightarrow{r}$ **in** E **do**
5: $\quad cid \leftarrow coo2cell(\overrightarrow{r})$
6: $\quad$ **if** $cid \in F_c \cup E_c$ **then**
7: $\quad\quad C_{cid} \leftarrow C_{cid} \cup \{\overrightarrow{r}\}$
8: $\quad$ **end if**
9: **end for**



(a) Initial set C (b) $I_c \cup F_c$

(c) I_c and F_c (d) I_c, F_c, E_c and the final C set

Figure 3: Example of the tiling of a 2D subdomain, as irregular as presenting an internal hole. The continuous line represents the perimeter of the subdomain and the dashed lines mark the internal and external regions at distance less than or equal $r_{\max}$ to the perimeter.

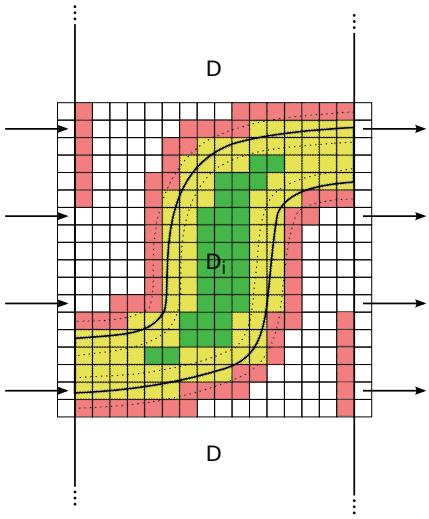


Figure 4: Example of the tiling of a subdomain taking into account periodic boundary conditions. In this case the selection of external cells must follow the periodicity.

Algorithm 6 Migration management using the subdomain tiling.

Require: D_i, N_i
Require: C,I_c,F_c,E_c
1: $S \leftarrow \bigcup_{cid \in E_c} C_{cid}$
2: $C_{cid} \leftarrow \emptyset$, $\forall cid \in E_c$
3: **for each** cid **in** F_c **do**
4: **for each** $\overrightarrow{r}$ **in** C_{cid} **do**
5: **if** $\overrightarrow{r} \notin D_i$ **then**
6: $S \leftarrow S \cup \{\overrightarrow{r}\}$
7: $C_{cid} \leftarrow C_{cid} \setminus \{\overrightarrow{r}\}$
8: **end if**
9: **end for**
10: **end for**
11: $S \rightarrow mcast(N_i)$
12: $E \leftarrow recv(N_i)$
13: **for each** $\overrightarrow{r}$ **in** E **do**
14: $cid \leftarrow coo2cell(\overrightarrow{r})$
15: **if** $cid \in F_c$ **then**
16: **if** $\overrightarrow{r} \in D_i$ **then**
17: $C_{cid} \leftarrow C_{cid} \cup \{\overrightarrow{r}\}$
18: **end if**
19: **end if**
20: **end for**

replaced by the selection of particles in frontier cells. Property 3 of the tiling procedure ensures that all internal particles potentially involved in interdomain interacting pairs are transmitted. Since frontier cells cover a superset of the internal frontier, some internal particles that could never interact with neighboring domains are also transmitted, resulting in an communication overhead that is expected to be limited. On the receiving side, proximity tests are performed by checking that received particles are located inside external and frontier cells. For property 4, all external particles potentially involved in interdomain interacting pairs are retained. As in the previous case, since external and frontier cells together represent a superset of the external frontier, some external particles that do not interact with the subdomain are kept. This may result also in a limited computation overhead when searching external cells for interacting pairs.

There may be cases in which either the communication or the computation overhead becomes too high. However, the communication overhead can be eliminated by performing the proximity test on frontier particles before the transmission. As per the computation overhead, it can be avoided by performing the proximity test on the subset of the received particles that lie within external or frontier cells. In either cases the test is performed only on a limited superset of the internal and external particles that can be

involved in interdomain interacting pairs.

After frontier particles have been exchanged, forces can be computed and particles positions updated. The tiling allows to perform these operations by using a link-list algorithm [2]. The update can be done by scanning all internal cells and, for each particle in each cell, by searching for all interacting particles inside the 27 neighboring cells. The procedure is similar for frontier cells, with the difference that interacting pairs must be searched only for elements belonging to the domain. This distinction usually does not require membership tests. For example, if cells are implemented as lists, then it is sufficient to maintain two distinct lists for internal and external particles in each frontier cell.

After positions have been updated, Algorithm 6 is executed to handle particles migration. We assume that the update algorithm returns the cells containing only the updated internal particles without the external ones (that can be safely discarded after forces have been computed). In the first part of the algorithm, particles that departed from D_i are searched (lines 1 to 10). From property 5, it follows that all particles that moved to external cells have left D_i, so that they can be added to set S and removed from the cells (lines 1 and 2). Within the frontier cells, particles that actually departed from D_i must be found. This step requires the execution of the membership test on all particles inside frontier cells (lines 3 to 10). Clearly, for property 2, internal cells can be ignored. At this point, particles that left D_i are sent in multicast to neighboring processors (line 11). In the second part of the algorithm, particles that left neighboring subdomains are received (line 12) and searched for new entries (lines 13 to 20). For each received particle, the index of the containing cell is first computed (line 14). If it is an external cell, then the particle is ignored because it can not belong to D_i. If it is a frontier cell then the membership test is performed (line 16). If it enters D_i, it is added to the cell (line 17), otherwise it is discarded.

With this procedure, we limit the communication overhead in frontier and migration management by having processors exchange a limited superset of the particles involved in those operations. To assess the advantage of our solution, we run a parallel simulation on an irregular domain and applied the tiling to the resulting subdomains. We measured the amount of data transferred to exchange particles during both frontier and migration management. In Table 1 we report the average number of particles sent by processors compared to the total number of particles in their subdomains. With 8 subdomains, processors exchange only 5.4% of the their data, resulting in a bandwidth saving of $\sim$95%. In our tests, the percentage of transferred data increases with the number of processors up to 10.9%, with 24 processors.

From the computational viewpoint, frontier management requires only table lookups that are performed on particles received by neighboring processors. For what concerns migration management, it requires both table lookups and membership tests. Before data exchange, particles to be sent are found by applying a membership test for each particle in frontier cells. After the exchange, a table lookup is performed on each particle received and, only to those located in a frontier cell, is applied the membership test. Table 2 shows the average number of cells resulting from the tiling procedure. We used the

Table 1: Number of particles transferred vs total number of particles per processor, for frontier and migration management using the cell tiling. The system is an irregular domain $342\times228\times600$ with 500000 particles evenly distributed partitioned in 8, 16 and 24 subdomains. Data is averaged over 1000 iterations.

tasks	sent particles	total particles	sent %
8	3434	62500	5.4%
16	2708	31250	8.6%
24	2300	20832	10.9%

Table 2: Average number of internal, frontier and external cells, per processor, with an irregular domain $342\times228\times600$ partitioned in 8, 16 and 24 subdomains. The upper part contains the number of cells obtained with $r_{\max}=0.81$, corresponding to a much finer cell tiling, while the lower part is obtained with $r_{\max}=1.62$.

tasks	internal	frontier	external
8	891270 (77%)	170947 (15%)	91140 (8%)
16	429835 (72%)	107198 (18%)	58060 (10%)
24	277584 (69%)	82246 (20%)	45054 (11%)
8	95613 (60%)	41297 (26%)	23498 (14%)
16	44103 (52%)	25659 (30%)	15072 (18%)
24	27316 (46%)	19485 (34%)	11719 (20%)

same domain of Table 1, partitioned in 8, 16 and 24 subdomains, tiled with two different cutoffs. In both cases the percentage of frontier cells that contains particles to which tests are applied, is quite stable with any number of processors.

3 Conclusions

To the best of our knowledge, the present work is the first effort to define a general method to perform parallel Molecular Dynamics in presence of irregular domains. The end result is a scheme that features $\mathcal{O}(N)$ complexity together with a general way to handle particle migration and computation of forces among irregular subdomains. The presented method is implemented in a multi-GPU code, by using CUDA and MPI programming paradigms, and integrated into the package MUPHY, one of the first examples of high performance parallel code for the simulation of multi-physics/scale bio-fluidic phenomena. Details of the implementation and the corresponding results will be the subject of a forthcoming paper.

References

[1] D. Frenkel and B. Smit, Understanding Molecular Simulation, Academic Press, London, 1996.
[2] M. P. Allen and D. J. Tildesley, Computer Simulation of Liquids, Clarendon Press, Oxford, 1987.

[3] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale and K. Schulten, Scalable molecular dynamics with NAMD, J. Comput. Chem., 26 (2005), 1781–1802.
[4] S. J. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys., 117 (1995), 1–19.
[5] M. Harvey, G. Giupponi and G. De Fabritiis, ACEMD: Accelerated molecular dynamics simulations in the microseconds timescale, J. Chem. Theory Comput., 5 (2009), 1632–1639.
[6] D. A. Case et al., The Amber biomolecular simulation programs, J. Comput. Chem., 26 (2005), 1668–1688.
[7] `http://ambermd.org/gpus/`.
[8] M. D. Mazzeo and P. V. Coveney, HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries, Comput. Phys. Commun., 178(12) (2008), 894–914.
[9] M. Bernaschi, M. Fyta, E. Kaxiras, S. Melchionna, J. Sircar and S. Succi, MUPHY: A parallel MUlti PHYsics/scale code for high performance bio-fluidic simulations, Comput. Phys. Commun., 180 (2009), 1495–1502.
[10] S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, F. J. Rybicki, D. Mitsouras, A. U. Coskun and C. L. Feldman, Hydrokinetic approach to large-scale cardiovascular blood flow, Comput. Phys. Commun., 181 (2010), 462–472.
[11] `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`.
[12] `http://www.labri.fr/perso/pelegrin/scotch/`.
[13] `http://www.ibm.com/systems/deepcomputing/bluegene/`.