Journal of Parallel and Distributed Computing 157 (2021) 243–255

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

MDScale: Scalable multi-GPU bonded and short-range molecular dynamics

Gonzalo Nicolas Barreales [a], Marcos Novalbos [c], Miguel A. Otaduy [a,b], Alberto Sanchez [a,b,*]

[a] Universidad Rey Juan Carlos, Madrid, Spain
[b] Research Center for Computational Simulation, Madrid, Spain
[c] U-tad, Madrid, Spain

ARTICLE INFO

Article history:
Received 8 November 2019
Received in revised form 24 March 2021
Accepted 12 July 2021
Available online 21 July 2021

Keywords:
Molecular dynamics
Multi-GPU
Large scale
Bonded forces
Van der walls forces
NAMD

ABSTRACT

GPUs have enabled a drastic change to computing environments, making massively parallel computing possible. Molecular dynamics is a perfect candidate problem for massively parallel computing, but to date it has not taken full advantage of multi-GPU environments due to the difficulty of partitioning molecular dynamics problems and exchanging problem data among compute nodes. These difficulties restrict the use of GPUs to only some of the computations in a full molecular dynamics problem, and hence prevent scalability beyond just a few GPUs. This work presents a scalable parallelization solution for the bonded and short-range forces present in a molecular dynamics problem. Together with existing solutions for long-range forces, it enables highly scalable, parallel molecular dynamics on multi-GPU computing environments. Specifically, the proposed solution divides the molecular volume into independent parts assigned to different GPUs, but it maintains a global bond structure that is efficiently exchanged when atoms move across GPUs. We demonstrate close-to-linear speedup of the proposed solution, simulating the dynamics of gigamolecules with 1 billion atoms on a computing environment with 96 GPUs, and obtaining superior performance to the well known molecular dynamics simulator NAMD.

© 2021 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Molecular systems are composed of a large number of particles that interact with each other at the atomic level. Molecular dynamics simulations [34] attempt to computationally recreate these interactions to estimate the behavior of molecular systems that cannot be solved analytically. Among others, they enable the prediction of protein structure and the design of new drugs. For instance, the pharmaceutical industry tries to anticipate the binding affinity of a ligand in a receptor [32] based on the interactions produced among all the atoms to be able to analyze their properties before synthesizing them. This topic is interesting because the synthesis of a molecular system often involves high costs in terms of materials, laboratory costs and production time. For instance, the HIV-1 virus capsid model [45] is made of tens of millions of atoms, providing a platform for further research in targeted pharmacological intervention. Molecular dynamics considers the characteristics of atoms and their force interactions to faithfully reproduce their motion for a given time. Discretization methods divide the simulation into time steps, and each time step proceeds by computing the interaction forces and solving the new atom positions thanks to numerical integration. The solution to molecular dynamics bears a high computational and memory cost [24]. Supercomputers arise as the natural platform for molecular dynamics, and the largest systems simulated to date have used first-rate supercomputers with hundreds of thousands of cores [12,17,9]. Moreover, molecular dynamics is a massively parallel problem, and is well suited to single-GPU architectures. However, it brings multiple challenges to multi-GPU architectures, which are found today in modern supercomputing centers [39]. In addition, GPUs are limited in memory resources, and the simulation of large molecular systems with many millions of atoms requires partitioning the computational cost.

Molecular dynamics problems combine the structural characteristics of mesh-based computational problems (e.g., FEM simulation) and particle systems (e.g., fluids). Therefore, the partitioning strategies for neither problem apply to molecular dynamics. Bonded interactions require maintaining static connectivity information during the simulation, which prevents using straightfor-

* Corresponding author.
E-mail addresses: gonzalo.nicolas@urjc.es (G.N. Barreales), marcos.novalbos@u-tad.com (M. Novalbos), miguel.otaduy@urjc.es (M.A. Otaduy), alberto.sanchez@urjc.es (A. Sanchez).

https://doi.org/10.1016/j.jpdc.2021.07.006
0743-7315/© 2021 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

ward spatial partitioning. Non-bonded short-range interactions, on the other hand, require dynamic neighbor updates, which prevent using precomputed partitions.

In this work, we present MDScale, a fully scalable multi-GPU molecular dynamics partitioning algorithm. Specifically, in this paper we solve the scalability of the computation of bonded and short-range electrostatic forces, which complement existing scalable solutions for long-range forces [24]. Our algorithm addresses the aforementioned challenges, minimizing the communication between devices in parallel and distributed molecular dynamics, and thus it achieves close-to-perfect scalability.

At a high level, we succeed to distribute molecular dynamics among massively parallel compute elements (i.e., the GPUs). All our algorithms are inherently parallel, and are executed directly on GPU architectures. We do not use GPUs as mere co-processors of CPU nodes which handle the distribution of computations. As a result, we avoid the overhead of CPU-GPU communication and data management, and we maximize scalability. This is in contrast to state-of-the-art distributed molecular dynamics simulators, which share the computation load between the CPU and the GPU [5,25,27,1,15,19].

At a low level, we have designed data structures and algorithms to handle efficiently, both in time and space, dynamic partitioning. For large molecular dynamics problems, it is imperative to distribute both computations and data. Then, the main algorithmic challenge is to perform efficient dynamic partitioning while maintaining static connectivity information. We propose an efficient solution, through massively parallel conversion to and from global connectivity information and local handlers within each partition.

We have implemented our MDScale algorithm on Amazon Web Services, comparing its performance with the well-known molecular dynamics simulator NAMD [27]. We have simulated bonded and short-range electrostatic forces of, among other tests, gigamolecules of 1 billion atoms, and the results indicate scalability.

The paper is organized as follows. Section 2 analyzes different parallel implementations of molecular dynamics simulation techniques. Section 3 describes the background of molecular dynamics and the different types of forces. Section 4 proposes our scalable Multi-GPU molecular dynamics simulator and the different techniques used for optimizing the communication among GPUs. Section 5 discusses implementation details of our algorithm, as well as the data structures used for dividing the molecular volume into independent parts while maintaining the global bond structure when atoms move. Section 6 presents a complete evaluation with various tests, comparing the performance of our solution with NAMD. Lastly, Section 7 discusses the results, sets out the conclusions drawn and proposes future work.

2. Related work

There are different parallel computing algorithms for molecular dynamics simulation [18,42]. Based on them, several molecular dynamics simulators [5,25,27,1,15] have been developed to solve this computational problem. NAMD [27] and GROMACS [1] are the most widely used molecular dynamics simulators. Specifically, NAMD treats the molecule as a three-dimensional patchwork quilt. The number of patches is determined by the size of the simulation independently of the number of computing elements (CEs). Nearby patches are kept on the same processor which minimizes the necessary communications. Nonetheless, neither NAMD nor GROMACS run the entire force calculation on the GPU and performance may therefore be limited by the CPU. For instance, the calculation in NAMD of bonded forces and the exchange of data between patches is done entirely in CPU [28].

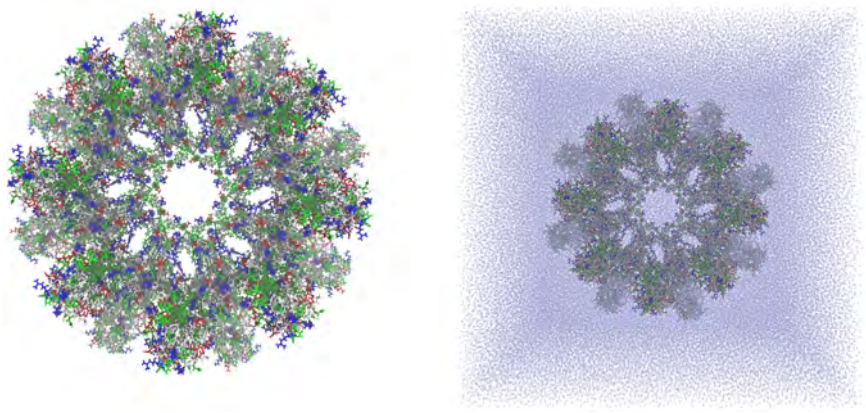
There are also interactive simulators of molecular dynamics, whose objective is to accelerate the simulation by imposing external restrictions that bias the simulation towards phenomena of interest [37]. Current simulation techniques may fail to capture some biological processes due to the large amount of time required for major conformational changes to take place. This can occur when studying the dynamics of complex biological systems [8] consisting of large molecular systems with millions of particles, such as virus capsids, or even the entire virus. Calculating this high number of interactions requires not only high simulation times but also sufficient computing resources. Running in a massively parallel environment with hundreds or thousands of GPUs could significantly improve the performance of the solution.

Most of the current supercomputing centers rely on hybrid architectures, consisting not only of a large number of nodes with multiple CPUs, but also with one or more powerful GPUs with a greater amount of RAM each. More than 90% of the most powerful supercomputers listed in the TOP500 list [39] are equipped with GPU devices to increase their computing capacity. Also different cloud platforms, such as Amazon Web Services, Google Cloud Platform and Microsoft Azure, have begun to provide GPUs in their instances. Multiple GPUs can be used together to massively calculate the large number of interactions between billions of atoms in parallel, but most of the existing molecular dynamics simulators do not run the entire simulation on GPU and if they do, they do not scale efficiently. Specifically, ACEMD [15] performs all the force calculations in a multi-GPU architecture, running each of the three force types on a different GPU. But this limits the maximum number of GPUs to use to three. Instead, MDScale simulates large biological systems by partitioning and running them in large-scale Multi-GPU environments in a fully scalable way. It therefore makes it possible to handle an increasing number of atoms by augmenting the number of GPUs.

Other algorithms designed for particle systems also use partitioning techniques to simulate massive particle systems [31]. For instance, particle-based fluid simulation shares many features with molecular dynamics, as both disciplines deal with particles that interact with each other. Particle-based fluid simulations can be solved through the Position Based Dynamics algorithm [21] that is easily parallelizable [10,11]. However, in this case the particles involved in the simulation are not directly related (bonded) and this allows them to move independently between the different CEs. In molecular dynamics, particles (atoms) are physically bonded and their movement is not totally free, as they are directly coupled to the motion of the bonded atoms. GPU partitioning requires that each CE not only has to store the atoms that correspond to it to execute but also needs to identify those atoms of other CEs with which its atoms are bonded. MDScale is able to solve this problem efficiently by running the entire simulation directly on the GPU.

3. Molecular dynamics background

Molecular dynamics is based on the calculation of the new position of each atom that composes a molecular system according to the forces that occur among them. Atoms are defined in a three-dimensional space that represents a molecule within a specific volume. For instance, Fig. 1(a) shows the 3D representation of a multi-molecular holoenzyme complex assembled around the adaptor protein dApaf-1/DARK/HAC-1, named 4V4L. The set of atoms of a biomolecule is usually surrounded by water molecules, as shown in Fig. 1(b), and must meet certain periodic conditions and limits. Periodic boundary conditions imply that an atom that leaves one side of the simulation volume is supposed to enter the opposite side, and the simulation therefore is not affected by any restriction. The partitioning of the system can be based on differ-



(a) (b)

Fig. 1. 4V4L molecule. In (b) the molecule is represented with water molecules around (~650K atoms).

ent topologies [23], which affects the treatment of these periodic boundary conditions.

Simulation usually involves dividing all the time to be simulated (of the order of seconds or milliseconds) into smaller time steps of a magnitude of femtoseconds (fs, 10^{-15} seconds). In each time step the simulator integrates the forces that affect each atom to obtain its new velocity and position. The forces acting on each of the atoms can be divided into:

- *Bonded* forces given by the bonds of the atoms that together make up the molecule. These forces are modeled like a spring, and only act on the atoms that form the bond. The bonded forces can be different depending on the number of atoms affected by the bond (single, angle, proper and improper). These forces depend directly of the positions of the affected atoms, as can be seen in the energy equations of the bonded forces:

$$\begin{aligned} E_{single} &= k(|r_{ij}| - r_0)^2 \\ E_{angle} &= k_\theta(\theta - \theta_0)^2 + k_{ub}(|r_{ik}| - r_{ub})^2 \\ E_{dihedral/improper} &= \begin{cases} k(1 + \cos(n\Phi + \theta)) & \text{if} \quad n > 0 \\ k(\Phi - \theta)^2 & \text{if} \quad n = 0 \end{cases} \end{aligned} \tag{1}$$

- *Non-bonded short range* or *Van der Waals* forces are those resulting from interactions between atoms. These interactions happen between atoms that may be separated. These forces decay very fast as the distance increase, therefore it is possible to establish a cutoff radius R_c from where the force is negligible. The energy equations of the Van der Vaals forces are:

$$E_{vdw} = \frac{A}{r_{ij}^{12}} - \frac{B}{r_{ij}^{6}} \tag{2}$$

A and B are constants precomputed using parameters σ_{ij} and ϵ_{ij}, which also are precomputed using σ and ϵ values of the single atoms:

$$\begin{aligned} \sigma_{ij} &= \frac{\sigma_i + \sigma_j}{2} \\ \varepsilon_{ij} &= \sqrt{\varepsilon_i \varepsilon_j} \\ A &= 4\sigma_{ij}^{12}\varepsilon_{ij} \\ B &= 4\sigma_{ij}\varepsilon_{ij} \end{aligned}$$

where σ_{ij} is the distance (finite) at which the potential between particles i and j is zero, and ϵ_{ij} is the depth of the potential between particles i and j [6].

- *Non-bonded electrostatic or long range* forces produced by the electrostatic charge of atoms. They are calculated using all interactions among the atoms of the system despite their distance. There are several methods for calculating these forces, such as PME [29] or MSM [14], and some implementation has demonstrated their scalability in Multi-GPU environments [24]. Therefore this paper will not discuss these calculations.

To solve these equations there are many integration methods to solve particles simulation forces. We can differentiate between first order methods (Euler methods) or second order methods. In these second order methods we can include Multiple-Time-Step (MTS) integrators [16,38]. There are several MTS integrators like Verlet-I/r-RESPA [13,40], MOLLY [33] or LN [3].

4. Scalable parallel multi-GPU molecular dynamics

Parallelizing a molecular dynamics algorithm to simulate large molecular volumes in a multi-GPU architecture implies certain challenges. First, it is necessary to partition the molecule among the different CEs (GPUs in our case, for best performance). The partitions are not static, and atoms that move must be exchanged among CEs during the simulation. In order to obtain a fully scalable system, each CE must keep only its own data in its local memory. This strategy minimizes memory usage, and hence it allows larger partitions and higher performance.

Second, although molecular dynamics can be classified as a particle system (atoms in this case), it entails other challenges for parallelization that are not found in regular particle systems. Specifically, atoms are physically bonded, as seen in (1). A bond involves a connection between two or more atoms. The movement of one of these atoms pulls the other atoms of the bond. A CE cannot simulate independently bonded forces of its own atoms; it also needs to know the atoms shared with other CEs. When atoms move, the connections with their bonded atoms must be maintained. As bonded atoms may reside on different CEs, this involves communication between CEs not only to exchange atoms, but also bonds. Our work addresses the exchange of atom and bond information, to allow each CE to efficiently compute all forces of its corresponding atoms on each time step of the simulation.

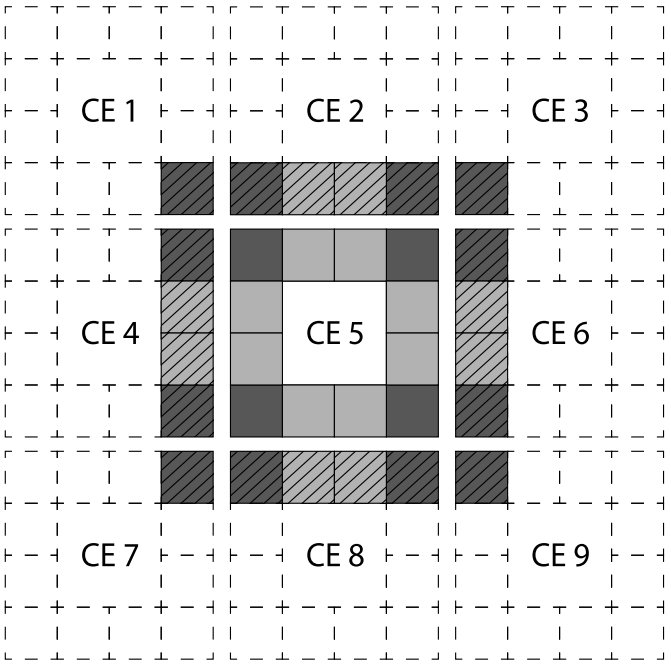


Fig. 2. Shared data between neighboring CEs, in 2D. Each CE shares data with its neighbors (8 neighbors in 2D, 26 in 3D). The interface area of CE 5 (striped) contains the data shared by other CEs, necessary for the calculation of its non-bonded short range forces. In addition, CE 5 has several data shared areas (colored without stripes). Some of the data is shared with multiple CEs (dark gray color), e.g. the corner adjacent with CE 1 is shared with CE 1, CE 2 and CE 4. It also shares data with individual CEs (light grey), such as with CE 2 on top of CE 5. This happens similarly in other shared areas.

4.1. Sharing atoms

MDScale distributes atoms among the different CEs as shown in Fig. 2. As atoms can move between CEs, they must communicate these changes in order to ensure that they own the right atoms at all times (dynamic partitioning). It is necessary to duplicate the data of some atoms at the partition boundaries to allow each CE to correctly calculate the non-bonded short range forces by itself. Note that these forces depend on the cutoff distance R_c between atoms as shown in (2). It is also necessary to know the data of all the atoms surrounding every atom within R_c, even at the edge of the partition. As a result, we identify three distinct zones in each CE:

- Private area: atoms that a CE does not share.
- Shared area: atoms shared with other CEs.
- Interface area: atoms not belonging to the CE, but shared by neighboring CEs to allow the computation of interaction forces among its private atoms.

Interface and shared zones contain the same data, but in different CEs, i.e. atoms stored in the interface of a CE correspond to atoms in the shared zone of another CE. The forces of the atoms within the interface area are calculated on the CE that contains them in its shared area. The interface area is just a storage to have the necessary data to calculate the forces of the atoms located at the edge of each CE. This partitioning system allows each CE to calculate the forces of its private atoms without any need to access private atoms belonging to other CEs. Atoms are communicated or updated among CEs only when they move.

We use the cellList algorithm [34,43,7] to partition the system. This is an algorithm initially designed for the computation of non-bonded short range forces. It divides the entire molecular system into cells of equal size. Cells are classified into the three types of areas seen above (see Fig. 3). The cell size is an integer fraction of the cutoff radius R_c. This means that, for each atom, we can find all the atoms within R_c into a constant number of cells. The cellList is fixed in the system, and each atom is initially assigned to one of these cells according to its x, y, z global position within the whole molecule. The system assigns the cells with their corresponding atoms to each CE according to their position. The atom data changes only when the position of the atoms in the cellList varies. Each predetermined number of steps the positions of the atoms inside the cellList are recalculated and, if necessary due to their motion, their data are updated among the CEs. R_c defines the number of cells that are exchanged with other CEs (interface-shared area), with the aim of sharing only the data needed to calculate non-bonded short range forces [35]. Note that each CE can perform these calculations independently, as it has the data shared by neighboring CEs in its own interface zone. The forces of atoms located at the edge of the shared area can be calculated thanks to the interface cells. We have adapted the data structures to optimize the simulation, reducing communication times among the different CEs (see Section 5).

4.2. Sharing bonds

Due to bonds, atoms pull from each other in their motion. When atoms move between CEs we update and send not only the atoms, but also the bonds to which they belong (see Fig. 4). We consider bonds as elements with their own x, y, z positions. They must be sent when their positions fall within the interface area of other CEs. Thereafter, each CE stores the corresponding bonds and atoms and can calculate their bonded and non-bonded short range forces. Received atoms are positioned in the interface cells. This implies that the receiver does not calculate the bonded forces of these atoms; only the sending CE that privately owns them calculates their forces. Receivers therefore do not need the bonds of interface atoms until they move to their shared cells.

Note that a bond combines two or more atoms, and this union may involve sharing information between atoms belonging to different CEs. It is therefore necessary to identify each atom as a whole in order to maintain the static relationship between the bonded atoms that may be in different CEs. Keeping a list of all atoms of the molecular volume in each CE involves a high memory cost, which would prevent scalability of the system. As each CE only calculates the forces of its shared and private atoms, it is more memory efficient to store only the necessary atoms and manage them locally. For this dual global and local vision, we identify the atoms within the system using two types of identifiers: i) global: identifies the atom within the whole molecule; and ii) local: identifies the atom within each CE. When sending an atom, the identifier must be converted from local to global and vice versa.

We perform this conversion using a hash table. We have chosen to use the hash table implementation of Nvidia CUDPP [2], which allows us to drastically reduce the execution time and memory usage. In all tests performed, the execution time required by the transformation using the hash table is less than 0.01% of the simulation time, which is considered negligible.

Specifically, when a CE receives a new atom, it creates a local atom and it assigns it the global identifier. The global identifier is useful for identifying which other atoms of other CEs are bonded. Local atoms are stored according to their position in the cells to improve access performance. This arrangement allows fast memory access and optimization of force calculation for each cell.

MDScale is able to calculate in parallel the bonded forces from the sorted local data structure. Each local atom identifier contains references to its bonds, and each bond also contains references to the atoms it connects, to allow the computation of the bonded forces. Fig. 5 shows this process.

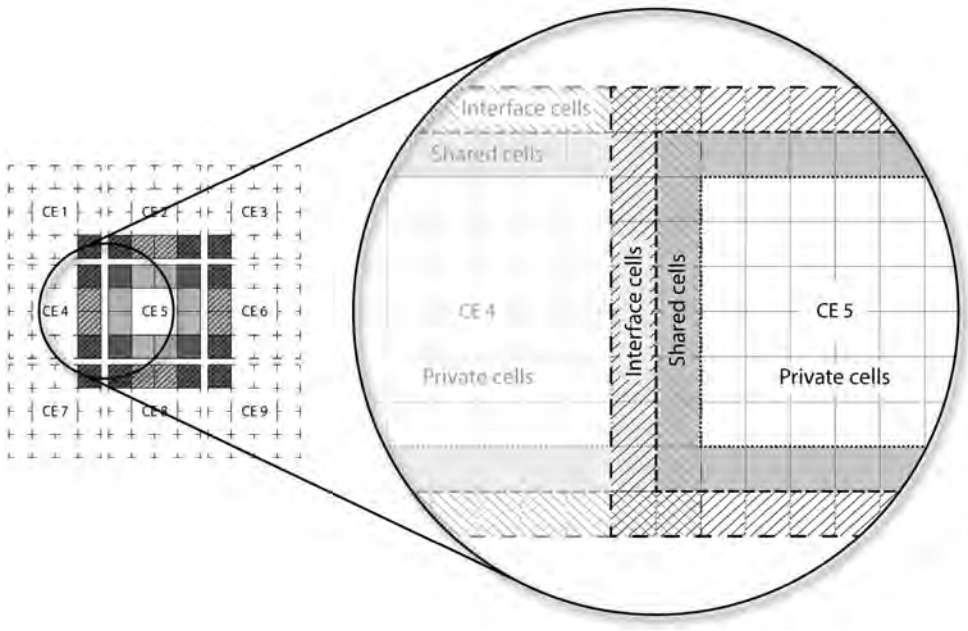


Fig. 3. Cell types. Each CE contains the three types of cells shown in Fig. 2. Shared cells contain atoms shared with other CEs. Interface cells (striped) contain the atoms shared by other CEs and therefore overlap with their corresponding shared cells. In this example, the width of the shared area is only one cell, but the method extends to larger sizes.

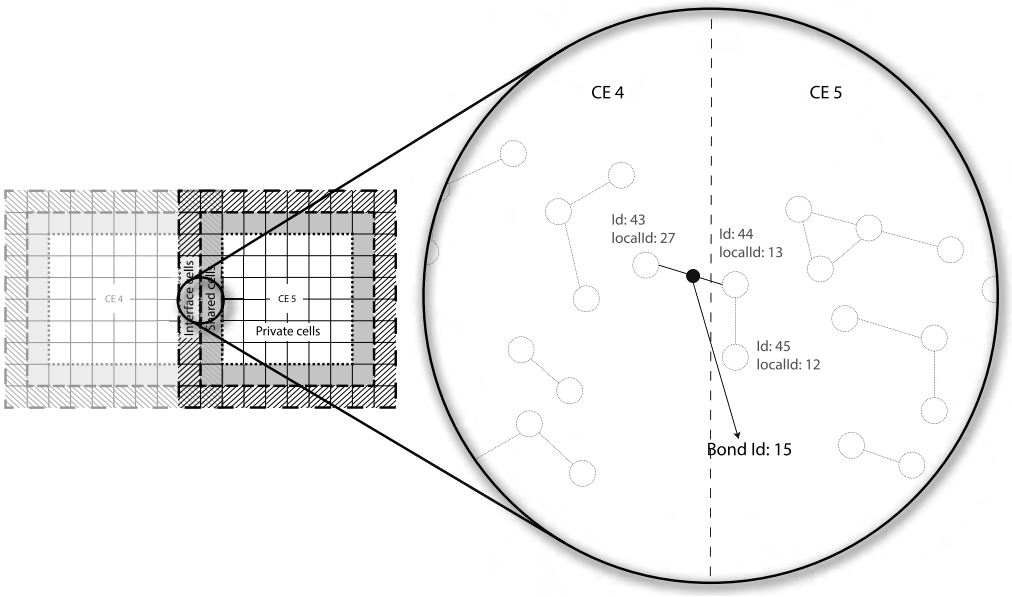


Fig. 4. Bonds distributed in two CEs. An atom moving from a private cell to a shared area has to be sent to the corresponding neighbors so that they can calculate their own non-bonded short range forces. Furthermore, when an atom (e.g. id 43) moves from a shared cell to interface area, it is necessary to send the bonds that affect it, to be able to simulate the bonded forces in all CEs that share a bond. The new CE needs to know the global identifiers of all atoms belonging to the bond (in this case ids. 43, 44 and 45).

4.3. Multi-GPU hardware architecture

Partitioning large molecular volumes in a multi-GPU environment requires a hardware communication system. There are two major types of multi-GPU architectures that can be used for this purpose. One possibility is to use an on-board hardware architecture with multiple GPUs via GPUDirect, NVLink, NVSwitch or PCIe connections [20]. This distribution is efficient in terms of communication between GPUs, since it is done through the motherboard data bus. But it has a major limitation: the small number of simultaneous GPUs that can be used (e.g. only 16 GPUs can be fully connected with the NVSwitch architecture).

Another possibility is to use a multi-GPU architecture with GPUs distributed across different compute nodes. Each node, which may contain one or more GPUs, communicates with others using advanced high-speed networks. Traditionally, information was sent from the RAM memory of the sending node via its network interface, and data was received in the RAM memory of the receiving node. This method caused a delay in communication between GPUs, because before the data was sent, it was dumped from GPU to RAM, and when data was received, it had to be dumped back into the destination GPU memory.

Currently, multi-GPU environments can implement a direct communication between GPUs through the network interface. This

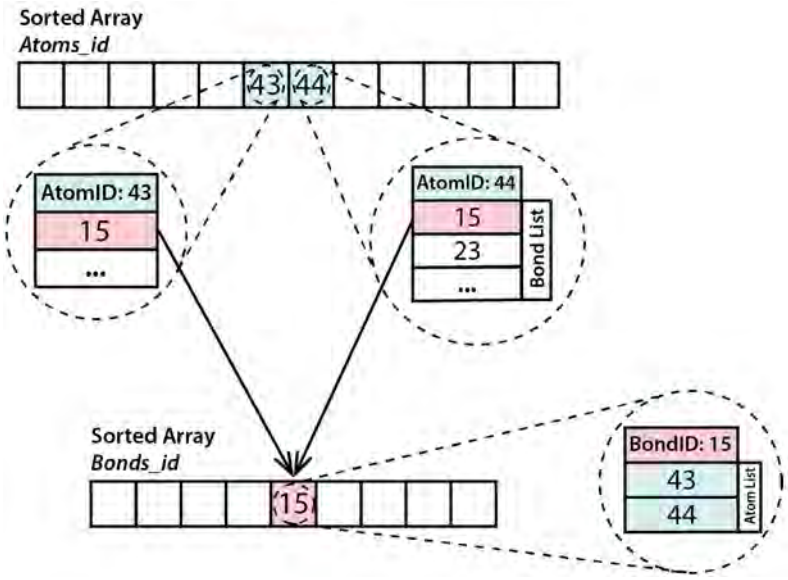


Fig. 5. Relationship between atoms and bonds. The data structures facilitate the calculation of the bonded forces for each atom without having replicated or inconsistent data.

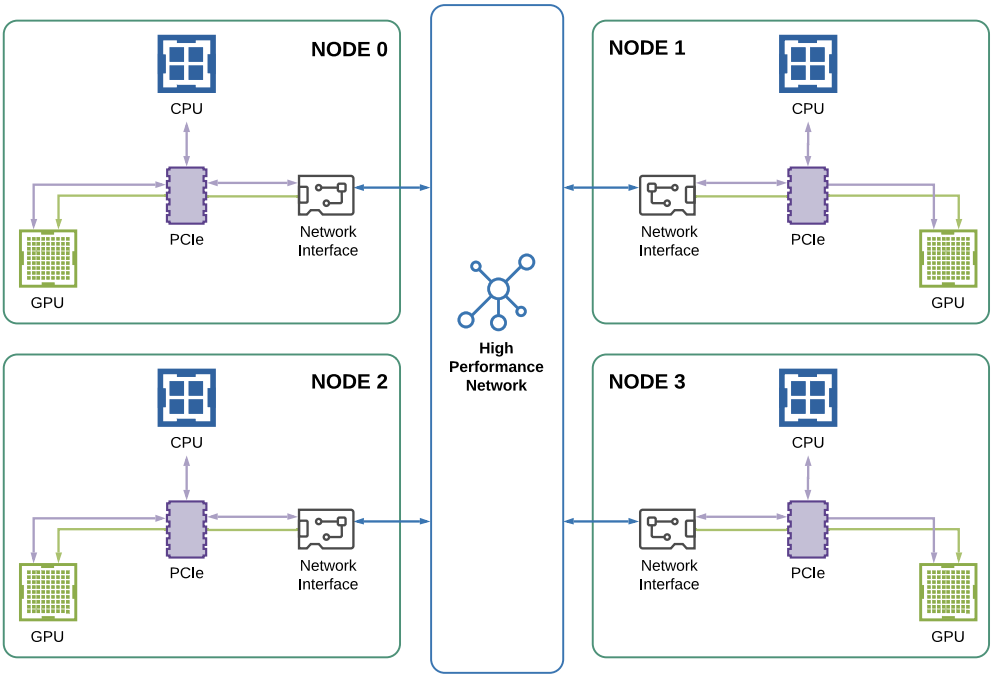


Fig. 6. Remote GPU-direct architecture.

communication, known as GPUDirect RDMA (remote direct memory access), allows network devices to directly access GPU memory, drastically reducing communication times between GPUs by avoiding RAM memory (see Fig. 6). In addition, the standard Message Passing Interface (MPI) for the development of parallel applications in distributed environments, via CUDA-aware MPI, allows these data exchanges to take place transparently. CUDA-aware MPI abstracts the communication system and enables sending and receiving GPU buffers directly, without having to first dump them in RAM memory. This allows the development of our simulator without having to take into account the type of architecture available: on board, traditional network communication, or GPUDirect RDMA. Moreover, this abstraction allows to use any of these architectures in a hybrid way.

5. Algorithm overview

We have used a generic Verlet-I/r-RESPA integrator to solve the equations of motion and calculate the trajectories of atoms due to the stability and simplicity it provides. It is a MTS integration method, where each time step is solved in 2 half steps. The first one gets the velocity of the atoms at half time. The second one calculates the rest of the velocity and their positions. The formulas are as follows:

$$v(t+\frac{\Delta t}{2}) = v(t) + \frac{1}{2m}F(t) \tag{3}$$

$$v(t+\Delta t) = v(t+\frac{\Delta t}{2}) + \frac{1}{2m}F(x(t+\Delta t)) \tag{4}$$

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2m}F(t)\Delta t^2 \tag{5}$$

Algorithm 1 Proposed Multi-GPU Verlet-I/r-RESPA MTS integrator.

```
1: procedure STEP(currentStep)
2:     integrateHalfVelocity() ← Equation (4)
3:     integratePosition() ← Equation (5)
4:     transferPositions(sharedIDs)
5:     if currentStep mod stepUpdate = 0 then
6:         atomIDs[]=identifyEvictedAtomIDs(AtomInfo)
7:         bondIDs[]=identifyEvictedBondsIDs(AtomInfo, BondElements)
8:         transferEvicted(atomIDs, bondIDs)
9:         updateCellList()
10:        sharedIDs[] = identifySharedAtomIds(CellList)
11:        transferPositions(sharedIDs)
12:    computeBondedForces()
13:    computeNonBondedShortRangeForces()
14:    integrateHalfVelocity() ← Equation (3)
15:    currentStep = currentStep + 1
```

Algorithm 2 Identification of atom identifiers to be shared.

```
1: procedure IDENTIFYEVICTEDATOMIDS(AtomInfo)
2:     localEvicted[] = computeAtomsEvicted(AtomInfo)
3:     globalEvicted[] = hashTable.retrieve(localEvicted)
4:     mergeSort(globalEvicted)
5:     return globalEvicted
```

where $v(t)$ represents the last velocity of each atom and $x(t)$ its last position.

We calculate the new position of each atom by computing the different bonded and non-bonded short range forces following (1) and (2). Algorithm 1 shows our implementation of the Verlet-I/r-RESPA integrator highlighting in brown the modifications with respect to the original algorithm. There are multiple data (atoms and bonds) exchange functions required for running in a distributed environment. In our partition, there are some data shared between CEs, e.g. the positions of the atoms in the shared cells. Each CE calculates the forces, velocities and positions of the atoms in its private and shared cells, but not those corresponding to atoms located in its interface cells. Each CE sends the positions of the atoms in its own shared cells and receives the same data for the atoms in its interface cells. Note that the movement of atoms in each step is slow and is less than the width of a cell, thus there is no need to interchange data between the CEs at each step [43]. After a certain number of steps (*stepUpdate*), the location of the atoms and bonds in the cellList has to be recalculated, sending the atoms and bonds that have left each CE (evicted) to their corresponding neighbors. First of all, it is necessary to discover the atoms and bonds that have changed their position in the system and must be transferred to another CE by means of *identifyEvictedAtomsIDs* and *identifyEvictedBondsIDs*. Algorithm 2 shows how to identify the atoms that move from shared to interface cells. Similarly, we can extrapolate this algorithm to Algorithm 3 since we consider bonds as one more element of the system. Those atoms and bonds outside the CE are subsequently transferred.

Once each CE receives the new atoms and bonds for its shared cells, it reallocates all local atoms and bonds in their corresponding cells using *updateCellList*. This enables a better organization of atoms and bonds according to their local ids in each CE to improve performance. As the location of the atoms in the cells has changed, it is necessary to identify which atoms are currently in the shared

Algorithm 3 Identification of bond identifiers to be shared as interface. Bonds are considered an element of the system, so they have a x, y, z position and can be placed within the cellList.

```
1: procedure IDENTIFYEVICTEDBONDIDS(AtomInfo , BondElements)
2:     computeBondsPositions(BondElements, AtomInfo)
3:     localEvicted[]=computeBondsEvicted(BondElements, BondPositions)
4:     globalEvicted[]=hashTable.retrieve(localEvicted)
5:     mergeSort(globalEvicted)
6:     return globalEvicted
```

Algorithm 4 Identification of atom identifiers to be shared to interface cells of other CEs.

```
1: procedure IDENTIFYSHAREDATOMIDS(CellList)
2:     sharedLocalIds[]
3:     sharedCells[] = CellList.getSharedCells()
4:     for each item cell in sharedCells do
5:         sharedLocalIds.add(cell.getAtomIds())
6:     sharedGlobalIds[] = hashTable.retrieve(sharedLocalIds)
7:     mergeSort(sharedGlobalIds)
8:     return sharedGlobalIds
```

cells using *identifySharedAtomsIds* to update the positions of atoms between CEs. Finally, each CE sends the positions of the atoms located in its shared cells with *transferPositions*. Algorithm 4 shows how to identify atoms located in the shared cells to subsequently send only their positions to the interface cells of neighboring CEs.

We use multiple data structures to store the information needed, optimize calculation forces and exchange data between CEs. Specifically, we use the following data structures to store the atoms in each CE:

- *AtomInfo* contains different information about the atoms (positions, velocities, physical parameters, and local and global identifiers).
- *AtomData* contains the bonds to which an atom is bonded. It is directly related to *AtomInfo*.
- *CellList* contains the cells of each CE, sorted by cell type. Each cell stores the local identifiers of the atoms it contains.

We use the following data structures to store the bonds and its relationship to atoms:

- *BondElement, AngleElement, DihedralElement, ImproperElement* arrays contain bond data for the calculation of forces, like the local identifiers of the affected atoms.
- *BondPositions, AnglePositions, DihedralPositions, ImproperPositions* arrays contain the position of the bonds in 3D space. These positions are calculated every *stepUpdate* and used to transfer atoms between CEs.

When data is updated between CEs, each CE calculates the new position of each atom and bond, and stores them in *AtomInfo* and *BondPositions* respectively. The positions of the atoms are used in the simulation with other data contained in *AtomInfo*, while *BondPositions* is only used for data transmission. After a predefined number of time steps *stepUpdate*, each CE computes the new location of atoms and bonds into the cellList and obtains the new shared cells to send the atoms belonging to the neighboring CEs. Each CE updates the positions of the atoms in the interface zone with data received from the neighboring CEs. At the same time it sends the position of the atoms in its shared zone. This allows each CE to compute the new forces with the most recent data of the simulation.

6. Evaluation

This section analyzes in depth the scalability and different benefits of the proposed approach. This analysis aims to demonstrate the performance of MDScale compared to the well known molecular dynamics simulator NAMD. We have chosen NAMD because it provides a wide variety of settings. In particular, in NAMD it is possible to disable non-bonded long-range forces, which are not covered in this work. By disabling these forces, we were able to perform fair comparisons. NAMD allows water molecules to be simulated as rigid bodies, but we have disabled this option in order to run forces for all the atoms. Both in our MDScale system and in NAMD we compute bonded forces and short-range non-bonded

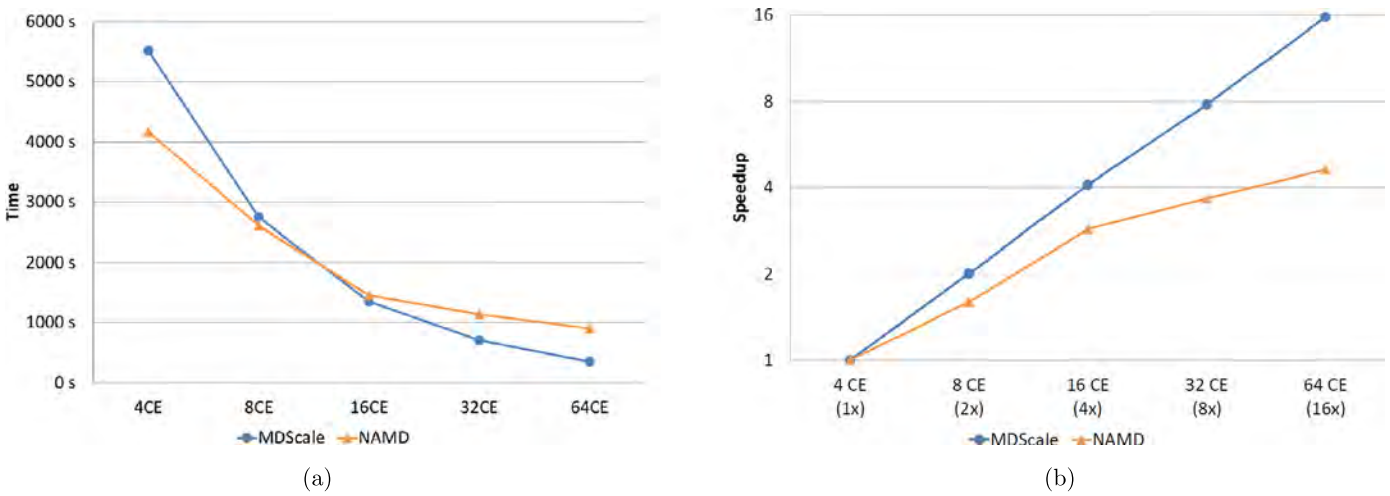


Fig. 7. Simulation of 100 steps of 8 million atom water molecules with NAMD and MDScale on an AWS K80 testbed. (a) shows the simulation time. (b) shows the speedup where the scale is logarithmic in base 2 to be able to appreciate the linear speedup of MDScale. The speedup is calculated using as reference the simulation time with the least number of CEs (4 in this case) that can hold the complete 8-million-atom system.

forces in all steps. Then, with both systems we reach the same precision of results for all scenarios. Note that, due to the lack of non-bonded long-range forces, the validity of the structure of the molecule over a long time is not measurable. Therefore, we limit all executions to 100 steps in our comparisons.

NAMD performs force calculations using either CPUs or GPUs depending on the computational load of each process, assigning heavier tasks to GPUs. The molecular system is divided into patches, whose number and size does not depend on the number of CEs, but on the total size of the system. To carry out the calculations, NAMD uses a primary-secondary architecture, where the primary CE monitors the whole process and sends the patches to each of the secondary CEs, who calculate the forces of the corresponding atoms. In contrast to MDScale, atoms are not distributed into patches according to their position, but according to the number of bonds and the density of atoms. In this way, all patches have a similar computational load, and a priori no CE should wait for the others. To enable updates between patches, each patch contains a neighbor list, with the necessary atoms from other patches to compute the electrostatic forces. These neighbor lists are updated every 10 steps, same as in MDScale. But in contrast to MDScale, the content of the patches does not need to be updated, as it does not depend on the positions of the atoms. Note that while NAMD can be affected by load imbalance because neighborhoods between patches may grow, MDScale may suffer it since the partitioning system is fixed in space.

We have simulated two types of benchmarks. On one hand, water molecules which allow us to vary both the number of atoms and CEs to measure performance, speedup and GPU memory usage. On the other hand, well-known molecules to compare the performance and GPU memory usage with different atom size and GPU types on realistic scenarios. In all benchmarks, we distribute the molecules following a binary partition according to [23] in 3 dimensions, taking into account that there are periodic boundary conditions. As mentioned above, we limit all simulations to 100 steps, as this allows us to obtain execution measures without wasting too many platform resources. We update atoms and bonds among CEs every 10 steps. Increasing *stepUpdate* caused the simulations to be unstable, and decreasing it did not imply a significant difference in the precision of the results obtained. In all cases we use a cutoff value of 1.0 nm, which is a value commonly used in practice for molecular dynamics simulation (see e.g. [41,30]).

We have used as testbed two clusters of several GPU instances running in Amazon Web Services (AWS) [22]. We use p2.xlarge and p3.2xlarge instances respectively, which use Nvidia K80 (p2.xlarge) and V100 GPUs (p3.2xlarge). Although these types of graphics cards, K80 or V100, are composed of two GPUs each, AWS virtualization provides only one per instance. Therefore, each instance runs on a single GPU with 12 GiB RAM on K80 and 16 GiB RAM on V100.

Firstly, we have compared MDScale with NAMD on a balanced scenario made of water molecules (8 million atoms in total) on an AWS K80 testbed. Fig. 7(a) shows the performance with both simulators. NAMD presents better performance on few CEs. With water molecules, the cost of bonded forces is low, and NAMD's CPU-based computation of these forces turns out more efficient. In contrast, as the number of CEs increases, MDScale overcomes NAMD thanks to better scalability. As shown in Fig. 7(b), MDScale obtains better speedup than NAMD, in particular as the number of CEs grows. We compute the speedup with respect to the simulation in 4 CEs since the molecular system does not fit in less CEs due to memory limitations. By increasing the number of CEs to 64 (x16) the speedup obtained by MDScale is 15.76, therefore its scalability is practically optimal.

As shown in Fig. 8, the data update time between CEs only uses a small percentage of the simulation time, between 1-6%. The percentage grows as the number of nodes increases due to the greater number of neighbors with whom the information is shared. Even so, we measure that the memory size used for interface area is low, not exceeding 3.25% of occupied memory with different number of CEs. Therefore, 94-99% of the time in the simulation is being fully parallelized, guaranteeing a high speedup.

Next, we have simulated water molecules with different number of atoms per CE, but ensuring a balanced load across CEs. Specifically we have simulated 1, 2 and 4 million atoms per CE running in 2, 4, 8, 16 and 32 GPUs on an AWS K80 testbed. Fig. 9 shows the MDScale memory usage per CE for the simulated molecules. The GPU memory usage increases slightly from 2 to 4 CEs and then it is stabilized, ensuring in-memory scalability. Fig. 10 shows the simulation performance (ps/day) obtained with MDScale and NAMD with the same molecules. We use this common measurement unit for comparison since it is useful to know how many days of simulation time are needed to complete, e.g., a microbiology synthesis process. In all cases the same number of atoms is simulated for both simulators. Note that increasing the number of elements implies a linear increase in the number of simulated atoms (e.g. 128 million distributed into 4 million per CE on 32 GPUs). As the number of atoms per node doubles, the com-

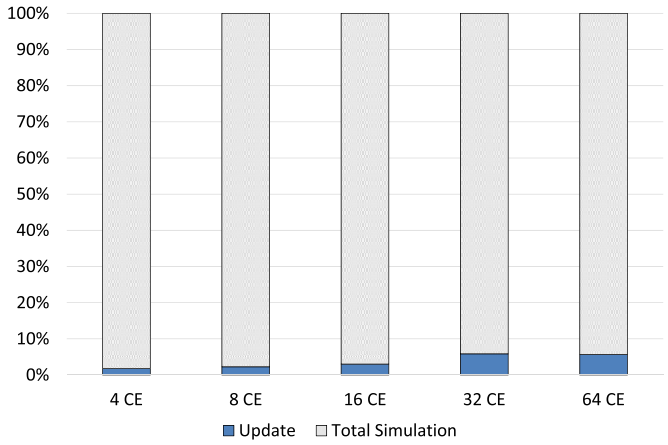


Fig. 8. Percentage of the time required in MDScale for updating atoms and bonds compared to the simulation of 8 million atom water molecules on an AWS K80 testbed.

Table 1
Measures obtained from simulating a water system with 1 billion atoms on 128 Nvidia Tesla K80 GPUs.

Performance	Update transfers	Memory usage
0.43 ps/day	6.15%	10.42 GB/CE

Table 2
Measures obtained from simulating water molecules on 96 Nvidia Tesla V100 GPUs.

Num. Atoms	Performance	Updates	Mem. usage
1 Billion	4.8 ps/day	5.78%	12.32 GB/CE
1.2 Billion	2.92 ps/day	5.23%	15.63 GB/CE

putation time is less than double, taking advantage of the massive parallelism of the GPU. As discussed earlier, NAMD presents better results than MDScale under small amounts of water molecules per CE due to the low number of bonded forces. As the number of atoms and CEs increases, the capacity of NAMD decreases while MDScale remains stable. This allows MDScale to handle large-scale molecular dynamics with a growing amount of atoms by simply adding GPUs to the infrastructure.

As a demonstration of the scalability of MDScale, we have simulated a water system with 1 billion atoms, distributed in a balanced way over 128 K80s. Table 1 shows the performance obtained on the AWS K80 testbed. These measures are in compliance with Figs. 9 and 10. The simulation uses more than 10 GB per GPU out of the 12 GB RAM available in each K80. The percentage of time spent on the exchange and identification of atoms is not significant and remains almost stable with this amount of atoms and CEs.

The limiting factor for scalability in this case is the GPU memory size. We have run a similar simulation of water molecules using an AWS V100 testbed to be able to increase the GPU memory size. We have been able to run the same number of atoms (1 billion) using only 96 V100, due to the increase in GPU memory from 12GB in K80 to 16GB in V100. Furthermore, in the same 96 V100 we have been able to simulate up to 1.2 billion atoms. As it is shown in Table 2, the performance is significantly higher in comparison to Table 1 due to architectural improvements of the GPU.

Once the scalability of the proposal has been verified with water molecules, we have tested its operation in simulations of some biological macromolecular structures obtained from the Protein Data Bank [4]. First, we have compared MDScale and NAMD on a medium-sized molecular system. Specifically, we simulate 100 steps of bonded and non-bonded short ranges forces of the well-known biomolecule 4V4L [44] replicated 12 times (∼7.5 million atoms) shown in Fig. 11(a). Second, we have tested a more challenging scenario, simulating the large molecular system 3J3Q [26] representing the HIV-1 capsid (∼66 million atoms) shown in Fig. 11(b). Note that the HIV-1 virus capsid has been previously simulated using NAMD and CHARMM on the TITAN Supercomputer with 3880 GPU accelerated Cray-XK nodes [45]. On both molecular systems, the binary partitioning produces partitions that are far from perfectly balanced. There are areas with greater density of atoms and greater number of bonds.

Table 3
Measures obtained from simulating 1 billion atoms of 96 4UDF viruses replicated on 96 Nvidia Tesla V100 GPUs.

Performance	Update transfers	Memory usage
4.02 ps/day	7.10%	13.40 GB/CE

Both molecules were distributed among different numbers of nodes with their respective GPUs: 4, 8, 16 and 32 for 4V4L and 16 and 32 for 3J3Q, due to memory requirements. As shown in Fig. 12(a), MDScale outperforms NAMD regarding simulation times. The performance of NAMD gets worse compared to Fig. 7(a). This is because, when running a complex molecule system, NAMD spends time trying to perform load balancing if there is a high number of patches. For 12x4V4L and 3J3Q the speedup obtained in MDScale is almost linear: 1.98 and 1.88 respectively with 32 GPUs compared to 16 GPUs (2x). The difference in performance between the two tests is due to the greater imbalance of the 3J3Q molecular system. Additionally, as the size and complexity of the molecule grows, the performance improvement of MDScale becomes more significant.

Fig. 12(b) shows for each molecule the size of the occupied memory and interface area by CE in MDScale. For instance, the 3J3Q molecule divided into 16 K80s occupies almost the entire GPU memory. Note that the molecules are divided based on the position of the atoms, not the complexity of the molecule. This causes that there are CEs that contain a greater density of atoms with complex bonds and therefore they require more memory to store them. This is represented in the figure by standard deviation lines to measure the spread of the memory size distribution. Despite this load imbalance, as seen, speedup has not been significantly affected. The total memory size used decreases almost linearly as we increase the number of CEs. The memory used by the interface cells decreases less in comparison. The size of the interface does not depend on the number of atoms being simulated, but on the size of the cells, the cutoff radius and the number of neighbors. In any case, note that the interface size is very small (the scale of the figure is logarithmic) compared to the memory space required to store the atoms that corresponds to each CE.

Finally, we simulate a very large complex molecular system. In particular, we have replicated 96 times the 4UDF virus [36] (human parechovirus neutralization by human monoclonal antibodies, composed of ∼10.4 million atoms surrounded by water), to obtain a gigamolecule of 1 billion atoms (see Fig. 13). Table 3 shows the performance obtained in a balanced distribution on a 96 V100 testbed (one CE per 4UDF). The simulation uses more than 13 GB per GPU of the 16 GB RAM available in each V100. As expected, performance decreases slightly compared to the simulation of water molecules with the same number of atoms and GPUs (see Table 2), due to the high complexity of the molecule. Nevertheless, it shows that MDScale can tackle massive molecules adding as many GPUs as necessary to cover the molecular size and obtaining a good performance.

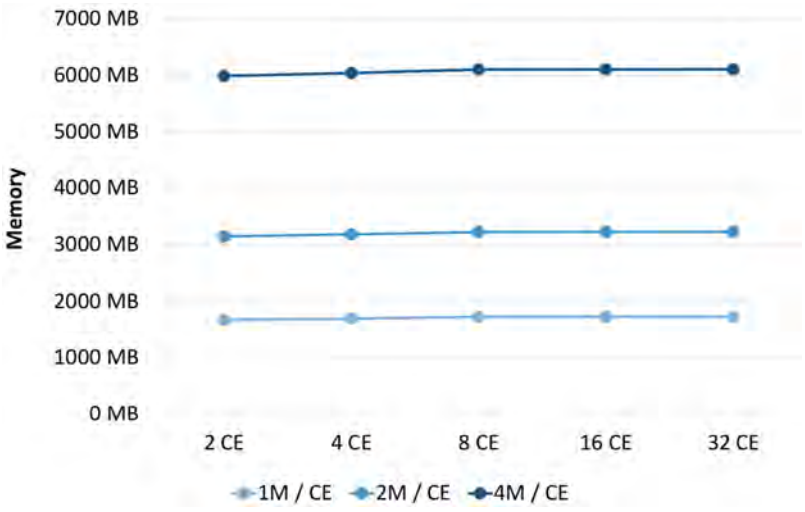


Fig. 9. MDScale memory usage per CE for the water molecules simulated in Fig. 10.

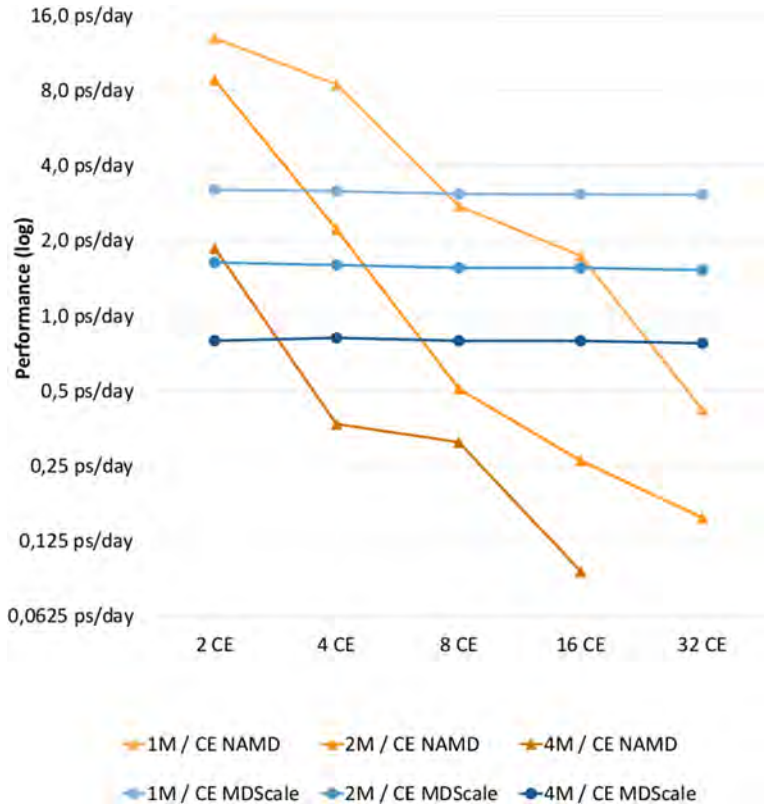


Fig. 10. Performance of NAMD and MDScale in logarithmic scale increasing the number of GPUs and atoms per CEs to simulate water molecules with tens of millions of atoms.

7. Conclusions

This article presents a scalable multi-GPU algorithm for bonded and short-range molecular dynamics, named MDScale, that runs the entire force calculation on GPUs. Our experiments suggest that it achieves efficient communication between different number of CEs and reduced memory usage. We have tested its operation by comparing it to the well-known molecular dynamics simulator, NAMD, by running the same simulations of different balanced and unbalanced molecular systems, like 4V4L and 4UDF replicated several times, 3J3Q and tens of millions atoms of water molecules. Our partitioning system allows us to add as many GPUs as necessary to cover the molecular size without losing performance. As proof of its scalability, we have carried out an evaluation checking its performance and memory use with different numbers of GPU instances on Amazon Web Services. Results show a close-to-linear speedup only minimally affected by different atom densities.

As a limitation, our partitioning system does not guarantee a load balance between GPUs, as it is done based on the current position of the atoms, not the complexity of the molecule. In addition, we have evaluated by simulating only 100 steps of bonded and non-bonded short range forces. This does not allow us to thoroughly test for load imbalance over time. We may want to argue that molecular dynamics, due to pressure exchange, does not suffer much load imbalance. As future work, we plan to include non-bonded electrostatic forces, which were outside the scope of this

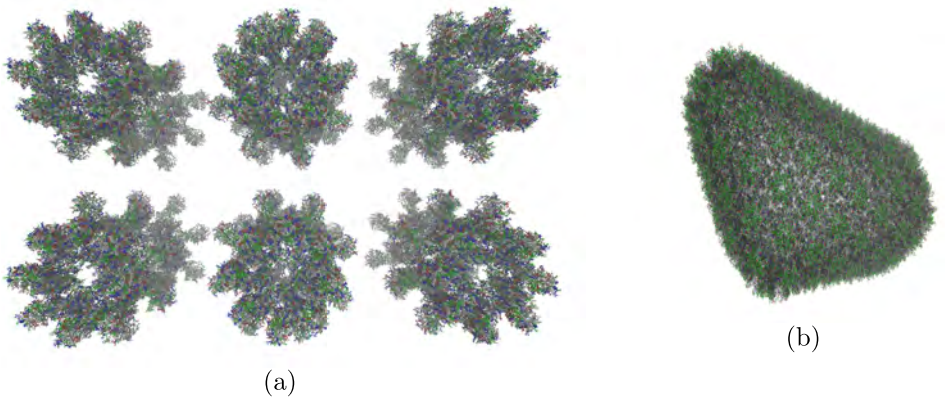


Fig. 11. Testbed molecules. (a) 12 molecules of 4V4L made of about 7.5 million atoms including the surrounding water molecules. (b) 3J3Q, HIV-1 capsid made of about 66 million atoms including the surrounding water molecules.

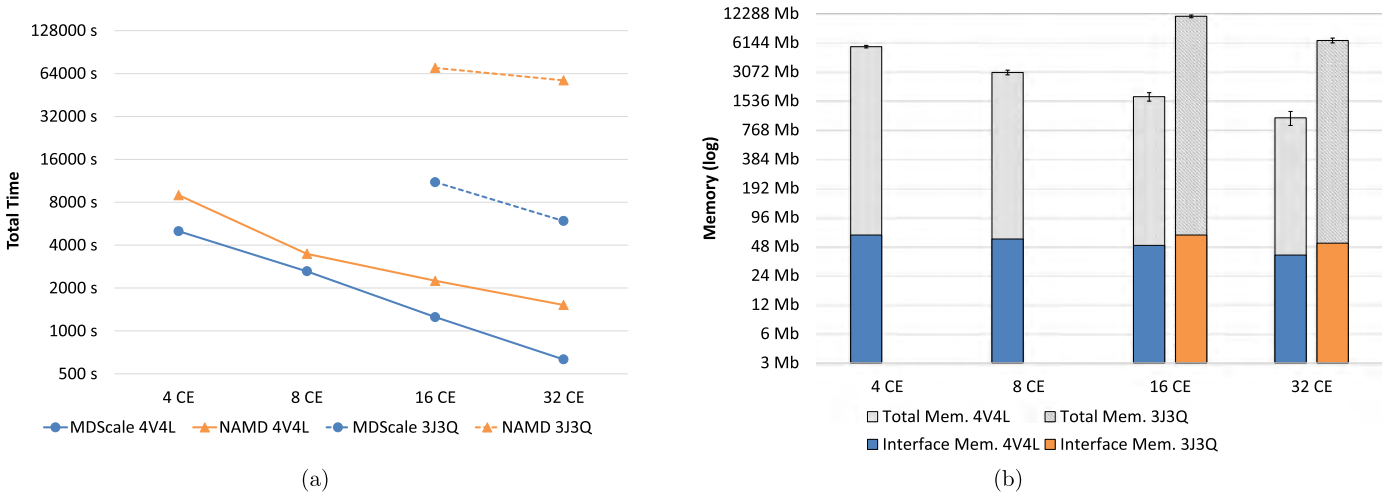


Fig. 12. Simulation of 12x4V4L and 3J3Q molecular systems on an AWS K80 testbed. (a) Comparison of MDScale and NAMD simulation time. The performance of MDScale is superior in all cases (4, 8, 16 and 32 GPUs) in both scenarios. (b) Memory footprint in MDScale per CE in the 3D binary partition distributed equitably among the different CEs. The graph is in logarithmic scale to appreciate the evolution of the used memory. Note that the interface area uses a small amount of memory compared to the total simulation size.

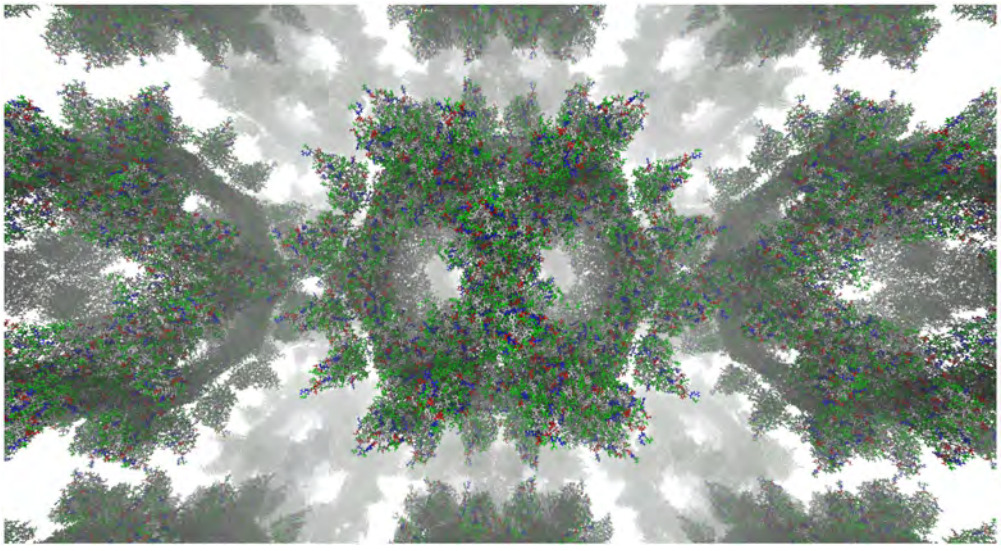


Fig. 13. Gigamolecule composed of 96 4UDF viruses replicated to obtain 1000 million atoms including the surrounding water molecules.

work. We also intend to offer MDScale as a software as a service (SaaS) for molecular dynamics simulation where domain experts can run their own large scale simulations with larger number of steps.

CRediT authorship contribution statement

Gonzalo Nicolas Barreales: Investigation, Software, Writing – original draft. **Marcos Novalbos:** Formal analysis, Software. **Miguel A. Otaduy:** Funding acquisition, Methodology, Writing – review & editing. **Alberto Sanchez:** Conceptualization, Funding acquisition, Supervision, Validation, Visualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partly supported by the Spanish Ministry of Science, Innovation and Universities (grant RTI2018-098694-B-I00). The evaluation was made possible by a grant from AWS Cloud Credits for Research.

References

[1] M.J. Abraham, T. Murtola, R. Schulz, S. Páll, J.C. Smith, B. Hess, E. Lindahl, GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers, SoftwareX 1–2 (2015) 19–25, https://doi.org/10.1016/j.softx.2015.06.001.

[2] D.A. Alcantara, A. Sharf, F. Abbasinejad, S. Sengupta, M. Mitzenmacher, J.D. Owens, N. Amenta, Real-time parallel hashing on the gpu, ACM Trans. Graph. 28 (2009) 154:1–154:9, https://doi.org/10.1145/1618452.1618500.

[3] E. Barth, T. Schlick, Extrapolation versus impulse in multiple-timestepping schemes. II. linear analysis and applications to Newtonian and Langevin dynamics, J. Chem. Phys. 109 (1998) 1633–1642, https://doi.org/10.1063/1.476737.

[4] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne, The protein data bank, Nucleic Acids Res. 28 (2000) 235–242, https://doi.org/10.1093/nar/28.1.235.

[5] B.R. Brooks, C.L. Brooks, A.D. Mackerell, L. Nilsson, et al., CHARMM: the biomolecular simulation program, J. Comput. Chem. 30 (2009) 1545–1614, https://doi.org/10.1002/jcc.21287.

[6] W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M. Merz, D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell, P.A. Kollman, A second generation force field for the simulation of proteins, nucleic acids, and organic molecules, J. Am. Chem. Soc. 117 (1995) 5179–5197, https://doi.org/10.1021/ja00124a002.

[7] M. Dobson, I. Fox, A. Saracino, Cell list algorithms for nonequilibrium molecular dynamics, J. Comput. Phys. 315 (2014), https://doi.org/10.1016/j.jcp.2016.03.056.

[8] M. Dreher, M. Piuzzi, A. Turki, M. Chavent, M. Baaden, N. Férey, S. Limet, B. Raffin, S. Robert, Interactive molecular dynamics: scaling up to large systems, Proc. Comput. Sci. 18 (2013) 20–29, https://doi.org/10.1016/j.procs.2013.05.165.

[9] W. Eckhardt, A. Heinecke, R. Bader, M. Brehm, N. Hammer, H. Huber, H.-G. Kleinhenz, J. Vrabec, H. Hasse, M. Horsch, M. Bernreuther, C.W. Glass, C. Niethammer, A. Bode, H.-J. Bungartz, 591 TFLOPS multi-trillion particles simulation on SuperMUC, in: International Supercomputing Conference, Springer Berlin Heidelberg, Leipzig, Germany, 2013, pp. 1–12.

[10] M. Fratarcangeli, F. Pellacini, A GPU-based implementation of position based dynamics for interactive deformable bodies, J. Graph. Tools 17 (2015) 59–66, https://doi.org/10.1080/2165347X.2015.1030525.

[11] M. Fratarcangeli, F. Pellacini, Scalable partitioning for parallel position based dynamics, Comput. Graph. Forum 34 (2015) 405–413, https://doi.org/10.1111/cgf.12570.

[12] T. Germann, K. Kadau, Trillion-atom molecular dynamics becomes a reality, Int. J. Mod. Phys. C 8 (2008) 1315–1319, https://doi.org/10.1142/S0129183108012911.

[13] H. Grubmüller, H. Heller, A. Windemuth, K. Schulten, Generalized Verlet algorithm for efficient molecular dynamics simulations with long-range interactions, Mol. Simul. 6 (1991) 121–142, https://doi.org/10.1080/08927029108022142.

[14] D.J. Hardy, Z. Wu, J.C. Phillips, J.E. Stone, R.D. Skeel, K. Schulten, Multilevel summation method for electrostatic force evaluation, J. Chem. Theory Comput. 11 (2015) 766–779, https://doi.org/10.1021/ct5009075.

[15] M.J. Harvey, G. Giupponi, G.D. Fabritiis, ACEMD: accelerating biomolecular dynamics in the microsecond time scale, J. Chem. Theory Comput. 5 (2009) 1632–1639, https://doi.org/10.1021/ct9000685.

[16] J. Izaguirre, Q. Ma, T. Matthey, J. Willcock, T. Slabach, B. Moore, G. Viamontes, Overcoming instabilities in Verlet-I/r-RESPA with the mollified impulse method, in: Computational Methods for Macromolecules: Challenges and Applications, in: Proceedings of the 3rd International Workshop on Algorithms for Macromolecular Modeling, vol. 24, Springer, New York, USA, 2002, pp. 146–174.

[17] K. Kadau, T. Germann, P.S. Lomdahl, Molecular dynamics comes of age: 320 billion atom simulation on BlueGene/L, Int. J. Mod. Phys. C 17 (2006) 1755–1761, https://doi.org/10.1142/S0129183106010182.

[18] S. Kupka, Molecular dynamics on graphics accelerators, in: Proceedings of CESCG, Institute of Computer Graphics and Algorithms, Castá-Papiernicka Centre, Slovakia, 2006, pp. 1–4.

[19] L. Lagardère, L.-H. Jolly, F. Lipparini, F. Aviat, B. Stamm, Z.F. Jing, M. Harger, H. Torabifard, G.A. Cisneros, M.J. Schnieders, N. Gresh, Y. Maday, P.Y. Ren, J.W. Ponder, J.-P. Piquemal, Tinker-HP: a massively parallel molecular dynamics package for multiscale simulations of large complex systems with advanced point dipole polarizable force fields, Chem. Sci. 9 (2018) 956–972, https://doi.org/10.1039/C7SC04531J.

[20] A. Li, S.L. Song, J. Chen, J. Li, X. Liu, N.R. Tallent, K.J. Barker, Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect, IEEE Trans. Parallel Distrib. Syst. 31 (2020) 94–110.

[21] M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff, Position based dynamics, J. Vis. Comun. Image Represent. 18 (2007) 109–118, https://doi.org/10.1016/j.jvcir.2007.01.005.

[22] G. Nicolas-Barreales, A. Sujar, A. Sanchez, A web-based tool for simulating molecular dynamics in cloud environments, Electronics 10 (2021) 185, https://doi.org/10.3390/electronics10020185, https://www.mdpi.com/2079-9292/10/2/185.

[23] M. Novalbos, J. Gonzalez, M.A. Otaduy, A. Lopez-Medrano, A. Sanchez, On-board multi-GPU molecular dynamics, in: Proceedings of the Euro-Par 2013 Parallel Processing, Springer, Aachen, Germany, 2013, pp. 862–873.

[24] M. Novalbos, J. Gonzalez, M.A. Otaduy, R. Martinez-Benito, A. Sanchez, Scalable on-board multi-GPU simulation of long-range molecular dynamics, in: Proceedings of the Euro-Par 2014 Parallel Processing, Springer, Porto, Portugal, 2014, pp. 752–763.

[25] D.A. Pearlman, D.A. Case, J.W. Caldwell, W.S. Ross, T.E. Cheatham, S. DeBolt, D. Ferguson, G. Seibel, P. Kollman, AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules, Comput. Phys. Commun. 91 (1995) 1–41, https://doi.org/10.1016/0010-4655(95)00041-D.

[26] J.R. Perilla, G. Zhao, P. Zhang, K.J. Schulten, PDB ID: 3J3Q. Atomic-level structure of the entire HIV-1 capsid, https://doi.org/10.2210/pdb3J3Q/pdb, 2013.

[27] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kalé, K. Schulten, Scalable molecular dynamics with namd, J. Comput. Chem. 26 (2005) 1781–1802, https://doi.org/10.1002/jcc.20289.

[28] J.C. Phillips, J.E. Stone, K. Schulten, Adapting a message-driven parallel application to gpu-accelerated clusters, in: SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008, pp. 1–9.

[29] S. Plimpton, R. Pollock, M. Stevens, Particle-mesh Ewald and rRESPA for parallel molecular dynamics simulations, in: Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, Society for Industrial & Applied Mathematics, Minneapolis, Minnesota, USA, 1997, pp. 1–13.

[30] J.-Y. Raty, F. Gygi, G. Galli, Growth of carbon nanotubes on metal nanoparticles: a microscopic mechanism from ab initio molecular dynamics simulations, Phys. Rev. Lett. 95 (2005) 096103, https://doi.org/10.1103/PhysRevLett.95.096103, https://link.aps.org/doi/10.1103/PhysRevLett.95.096103.

[31] E. Rustico, G. Bilotta, G. Gallo, A. Herault, C. Del Negro, Smoothed particle hydrodynamics simulations on multi-gpu systems, in: 2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, IEEE Computer Society, Garching, Germany, 2012, pp. 384–391.

[32] V. Salmaso, S. Moro, Bridging molecular docking to molecular dynamics in exploring ligand-protein recognition process: an overview, Front. Pharmacol. 9 (2018) 923, https://doi.org/10.3389/fphar.2018.00923.

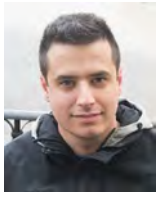
[33] J.M. Sanz-Serna, Mollified impulse methods for highly oscillatory differential equations, SIAM J. Numer. Anal. 46 (2008) 1040–1059, https://doi.org/10.1137/070681636.

[34] T. Schlick, Molecular Modeling and Simulation: An Interdisciplinary Guide, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[35] L. Schwiebert, E. Hailat, K. Rushaidat, J. Mick, J. Potoff, An efficient cell list implementation for Monte Carlo simulation on GPUs, CoRR, arXiv:1408.3764, 2014.

[36] S. Shakeel, B.M. Westerhuis, A. Ora, G. Koen, A.Q. Bakker, Y. Claassen, K. Wagner, T. Beaumont, K.C. Wolthers, S. Butcher, PDB ID: 4UDF. Structural basis of

human parechovirus neutralization by human monoclonal antibodies, J. Virol. 89 (2015) 9571–9580, https://doi.org/10.1128/JVI.01429-15.
[37] J. Stone, J. Gullingsrud, K. Schulten, A system for interactive molecular dynamics simulation, in: 2001 ACM Symposium on Interactive 3D Graphics, ACM, New York, NY, USA, 2001, pp. 191–194.
[38] W. Streett, D. Tildesley, G. Saville, Multiple time-step methods in molecular dynamics, Mol. Phys. 35 (1978) 639–648, https://doi.org/10.1080/00268977800100471.
[39] Top500.org, TOP500 supercomputer sites list statistics, https://www.top500.org/statistics/list/, 2018. (Accessed 1 December 2018).
[40] M. Tuckerman, B.J. Berne, G.J. Martyna, Reversible multiple time scale molecular dynamics, J. Chem. Phys. 97 (1992) 1990–2001, https://doi.org/10.1063/1.463137.
[41] F. Vitalini, F. Noé, B. Keller, Molecular dynamics simulations data of the twenty encoded amino acids in different force fields, Data in Brief 7 (2016) 582–590, https://doi.org/10.1016/j.dib.2016.02.086, http://www.sciencedirect.com/science/article/pii/S235234091630110X.
[42] J. Yang, Y. Wang, Y. Chen, GPU accelerated molecular dynamics simulation of thermal conductivities, J. Comput. Phys. 221 (2007) 799–804, https://doi.org/10.1016/j.jcp.2006.06.039.
[43] Z. Yao, J.-S. Wang, G. Liu, M. Cheng, Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method, Comput. Phys. Commun. 161 (2004) 27–35, https://doi.org/10.1016/j.cpc.2004.04.004.
[44] S. Yuan, M. Topf, L. Dorstyn, S. Kumar, S.J. Ludtke, C.W. Akey, PDB ID: 4V4L. Structure of the Drosophila apoptosome at 6.9 angstrom resolution, Structure 19 (2011) 128–140, https://doi.org/10.1016/j.str.2010.10.009.
[45] G. Zhao, J. Perilla, E. Yufenyuy, X. Meng, B. Chen, J. Ning, J. Ahn, A.M. Gronenborn, K. Schulten, C. Aiken, P. Zhang, Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics, Nature 497 (2013) 643–646, https://doi.org/10.1038/nature12162.



Gonzalo Nicolas is a pre-doctoral researcher at Universidad Rey Juan Carlos in Madrid. He obtained his degree in Computer Science at University of Leon in 2015 and the Master of Computer Graphics, Videogames, and Virtual Reality at the Universidad Rey Juan Carlos in 2016. His main field of research is HPC and GPU computing. He is currently working on parallel molecular dynamics simulation using high performance Multi-GPU architecture to simulate large molecular systems. He has wide experience in the use of supercomputers, clusters and cloud computing environments, using hundreds of nodes in different supercomputing facilities, like Barcelona Supercomputing Center and Amazon Web Services.

Alberto Sanchez is an associate professor at Universidad Rey Juan Carlos and researcher at Research Center for Computational Simulation. He received MS and PhD degrees, obtaining the Extraordinary PhD Award, in Computer Science from Universidad Politécnica de Madrid (Spain) in 2004 and 2008, respectively. Since 2007 he has a faculty position at the Department of Computer Science & Statistics, at Universidad Rey Juan Carlos (Madrid, Spain), where he is currently the vice-rector for digital learning and IT. His primary research areas are data analysis and visualization, high-performance and large-scale computing, where he has published several journal papers, book chapters and articles in international conferences. He has also done long placement abroad in some prestigious international researching centers, such as CERN, NeSC, NRC-Canada and the University of Melbourne. Finally, he had been also involved in the organization of some international conferences (IDA 2005, HPCS 2012, Cluster 2014) and in more than 25 program committees of different workshops and conferences.

Marcos Novalbos is a professor at Centro Universitario de Tecnología Arte Digital (U-tad Madrid) of GPU Processing and High Performance Computing. He received his BS (2006) in Computer Science and his PhD (2015) in Computer Science from the Universidad Rey Juan Carlos (URJC Spain). Between 2006 and 2018, he worked as research associate at Grupo de Modelado y Realidad Virtual (GMRV URJC), working in several fields related to GPU and MultiGPU optimizations, e.g. optimizing encryption and hashing algorithms in order to perform faster in multiGPU clusters. His research interests are centered around the optimization of applications by using GPU and multiCPU systems, with applications to biomedicine and cryptographic models. He worked at Plebiotic SL (Madrid, Spain), developing some optimizations in their molecular dynamics simulator.

Miguel Otaduy is associate professor at Universidad Rey Juan Carlos (URJC Madrid), where he leads the Multimodal Simulation Lab, in the Department of Computer Science. He received his BS (2000) in Electrical Engineering from Mondragon Unibertsitatea (Spain), and his MS (2003) and PhD (2004) in Computer Science from the University of North Carolina at Chapel Hill. Between 2005 and 2008, he worked as research associate at the Computer Graphics Laboratory of ETH Zurich, before joining URJC.

His research interests are centered around the simulation of mechanical systems in computer graphics, with applications to biomedicine, textiles, animation, or virtual touch. He has published over 100 papers and is coinventor of 8 patents. He has led multiple research projects, most notably a 2011 ERC Starting Grant, and a 2017 ERC Consolidator Grant. He has served as associate editor for journals in computer graphics, virtual reality and robotics, and has been the program chair for several conferences in computer animation and haptics.