Parallel Adaptively Restrained Molecular Dynamics

Krishna Kant Singh, Dmitriy F. Marin, Stephane Redon

Parallel Adaptively Restrained Molecular Dynamics

Krishna Kant Singh
NANO-D, INRIA
University of Grenoble Alpes,
LJK, F-38000 Grenoble, France

Dmitriy F. Marin
NANO-D, INRIA
LJK, F-38000 Grenoble, France
Email: dmitriy.marin@inria.fr

Stephane Redon
NANO-D, INRIA
LJK, F-38000 Grenoble, France
Email: stephane.redon@inria.fr

***Abstract*—Force computations are one of the most time consuming part in performing Molecular Dynamics (MD) simulations. Adaptively Restrained Molecular Dynamics (ARMD) makes it possible to perform fewer force calculations by adaptively restraining particles positions. This paper introduces parallel algorithms for single-pass incremental force computations to take advantage of adaptive restraints using the Message Passage Interface (MPI) standard. The proposed algorithms are implemented and validated in LAMMPS, however, these algorithms can be applied to other MD simulators. We compared our algorithms with LAMMPS for performance and scalability measurements.**

***Index Terms*—Molecular Dynamics, MPI, ARMD, Single-pass Incremental Force Update, LAMMPS**

I. Introduction

Molecular Dynamics (MD) codes are widely used to simulate systems in computational physics, chemistry and biology [1], [2], [3]. Simulated systems may contain billions of particles which interact with each other based on inter-particle potentials. Computation of forces based on inter-particle interactions is typically the most time consuming part of MD simulations [4]. Indeed, the number of such computations drastically increase with the size of the system, and they should be performed at each timestep. The number of timesteps may vary based on the problem. In biology, for example, many interesting phenomena occur at a micro- to milli-seconds time scale, and simulating a biological system at such a time scale is still challenging [5], [6]. All these challenges make MD simulations computationally expensive and, therefore, it is crucial to accelerate them. There are different methods to accelerate MD simulations, including optimized algorithms that perform mathematical operations faster [7], [8], parallelizing the MD codes by means of High Performance Computing (HPC) and building special-purpose super-computers [9].

In all MD simulations, most of the computational time is consumed in non-bonded force calculations, which are typically divided into long- and short-range forces. Long-range interactions involve electrostatic interactions that are generally computed using *e.g.* the reaction field method [10], the cell-multipole method, the fast multipole method [11], the particle mesh Ewald (PME) [12] or the Wolf method [13]. Short-range forces decay fast and are generally truncated at a cut-off distance r_c. These forces are efficiently computed by using either neighbor lists, cell lists, or a combination of them. Note that long-range forces may also be computed via neighbor lists when using the Wolf method.

In most cases, forces acting between particles depend upon inter-particle distances. Therefore, if the distance between two particles does not change, the forces acting on both particles also remain unchanged. Adaptively Restrained Molecular Dynamics (ARMD) is a recent approach that attempts to speed up simulations by reducing the number of force calculations [14]. In ARMD, particles adaptively update their positions based on their instantaneous kinetic energy. The particles that have a kinetic energy smaller than a user-defined threshold at a given time step are considered to be *restrained* (frozen) particles, and their positions are not updated at this time step. On the opposite, the particles that have a kinetic energy larger than another (larger) user-defined threshold are active, and have normal dynamics.

In this paper, we present the first parallel implementation of ARMD in the popular LAMMPS molecular dynamics simulation package.

A. LAMMPS

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a highly parallel and modular MD package. LAMMPS parallel algorithms use MPI and Spatial-Decomposition (SD) techniques to partition the simulation domain into sub-domains, and assign each of them to one MPI process.

The force applied to each particle depends upon its neighboring particles, which may belong to neighboring sub-domains. As a result, inter-processor communications are required for force computations. We will refer to particles whose positions needs to be communicated between MPI processes as *border particles*. In order to optimize the communication of border particles, each sub-domain is further partitioned into bins (based on a cut-off radius), and border particles are then defined as particles belonging to the bins neighboring other sub-domains. Therefore, each MPI process contains two kind of particles: local particles (belonging to this sub-domain) and ghost particles (border particles from neighboring sub-domains). An MPI process is responsible for updating the momenta and positions of its local particles, and computing interactions between local particles (*local interactions*), as well as interactions between local particles and ghost particles (*ghost interactions*). LAMMPS performs a two-way communication: positions are communicated in forward

directions (in the east, north and up directions) and computed forces are communicated back in reverse directions (west, south and down) (Fig. 1). Due to this, LAMMPS can fully utilize the Newton's third law. After setting up the simulation, LAMMPS repeats the steps listed in Alg. 1.

Algorithm 1: LAMMPS integration step

```
1 Update Momenta
2 Update Position
3 if (UpdateNeeded) then
4     Build Neighbor List for local particles
5 Forward Communications of border particles
6 Force Computations
7 Reverse Communications of forces
```

II. Adaptively Restrained Molecular Dynamics

For completeness, this section presents a brief overview of the ARMD methodology [14]. In ARMD, the time evolution of a system is obtained from the Adaptively Restrained Hamiltonian:

$$H_{AR}(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\Phi(\mathbf{q},\mathbf{p})\mathbf{p} + V(\mathbf{q})\,,$$

where $\mathbf{q},\mathbf{p}$ are positions and momenta vectors, respectively; $V(\mathbf{q})$ is a position-dependent potential; $\Phi(\mathbf{q},\mathbf{p})$ represents the inverse inertia matrix that adaptively imposes restraints during the simulation. Specifically, a particle i with instantaneous kinetic energy smaller than its restrained-dynamics threshold ε_i^r is *restrained*, and its position remains unchanged, while its momenta may evolve; a particle i with kinetic energy greater than its full-dynamics threshold ε_i^f is *active*, and has normal dynamics. A particle i with kinetic energy between ε_i^r and ε_i^f is in the *transition region*, and its corresponding $\Phi(\mathbf{q_i},\mathbf{p_i})$ changes according to a chosen smooth function. Particles in the transition region are considered to be active (but have transient dynamics). During simulation, particles might change their state, and the Adaptively Restrained Hamiltonian ensures that stable simulations can be performed. Equilibrium statistics can be recovered by performing the NVT simulation using AR Langevin dynamics. For more details on the ARMD method and its validation, we refer the reader to [14], [15], [16], [17].

III. Parallel ARMD Algorithms

In our work, we are taking advantage of the modular structure of LAMMPS: we implement ARMD as a separate module that allows usage of most implemented force-fields without any modifications. As mentioned above, in LAMMPS, each MPI process manages both local and ghost particles. In ARMD, local particles are further subdivided into active and restrained particles based on their instantaneous kinetic energy. Therefore, local interactions can be categorized as

- *active interactions*: interactions involving at least one active particle;
- *restrained interactions*: interactions involving restrained particles only.

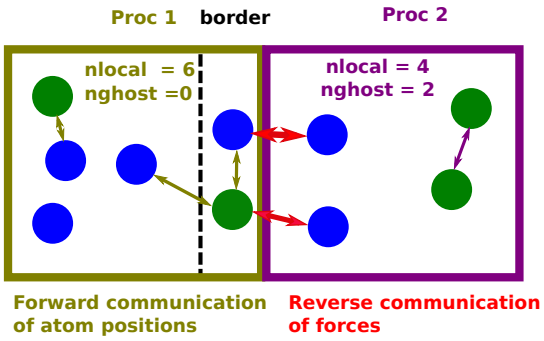


Fig. 1: A system containing 10 particles is divided between two processes (Proc 1 and Proc 2). Proc 1 sends positions of its two border particles (separated by a dashed line) by doing a forward communication. After forward communication is complete, Proc 2 has four local and two ghost particles, and Proc 1 has six local particles. Green (resp. blue) particles represent active (resp. restrained) particles. Each process computes forces acting on its local and ghost particles (forces are shown by arrows), while avoiding computations between restrained-restrained particles. After computing forces, forces acting on ghost particles (red arrows) are communicated back through reverse communication.

Ghost interactions might also be subdivided into active and restrained interactions. If ghost interactions are divided into active and restrained interactions then the amount of communications will be reduced because positions of restrained ghost particles need not be communicated. However, this would require two extra communication and barrier routines for forward and reverse communications of switched ghost particles: first, positions of switched ghost particles would be communicated to neighboring processes in forward directions; then force components due to these switched particles would be computed and, finally, force components due to switched particles would be communicated using reverse communication. These communication and barrier routines might act as a bottleneck. In order to avoid more communications routines, we propose a parallel algorithm in which only local interactions are subdivided into active and restrained interactions.

ARMD speeds up simulations due to its ability to incrementally update forces, instead of re-computing all of them. This is achieved through the use of single-pass incremental force update algorithm and Active Neighbor Lists (ANLs).

In order to efficiently parallelize ARMD in LAMMPS we introduce three main modifications (Alg. 2) to the LAMMPS integration step (Alg. 1):

1) Lists of active (A), restrained (R), switched to active (S_A) and switched to restrained (S_R) particles are updated at each timestep (line 2 in Alg. 2). ANLs are constructed when necessary (line 4 in Alg. 2) instead of the LAMMPS neighbor lists.
2) Particles that switched their state (from active to restrained or vice-versa) are taken care of (line 5 in Alg. 2).

Algorithm 2: ARMD integration step

```
1 Update Momenta
2 Update A, R, S_A, S_R
3 if (UpdateNeeded) then
4     Build ANLs of local particles
5 SwitchedForces()
6 Update Positions
7 Forward communications of border particles
8 IncrementalForceComputation()
9 Reverse communications of forces
```

3) Finally, force increments are computed (line 8 in Alg. 2) instead of recomputing all forces.

In the next sections, we present these algorithms for parallel ARMD.

A. MPI-enabled Active Neighbor List

This section introduces an algorithm to construct a MPI-enabled ANL that allows us to avoid calculating forces due to restrained interactions, and exploits Newton's third law. The MPI-enabled ANL provides an efficient way to compute forces involving active and ghost interactions. In order to use MPI functionality of LAMMPS (spatial decomposition and communication algorithms), we construct ANLs using cell and Verlet neighbor list algorithms (which are also used to build LAMMPS neighbor lists) [18], [19]. The ANL construction algorithm is shown in Alg. 3, where G is the list of ghost particles for the MPI process managing particle i, and $NeighboringCells[i]$ is the list of boxes neighboring or containing particle i (27 boxes in 3D). The proposed ANL construction algorithm does not require any communications between MPI processes. The ANL of an active particle contains both active and ghost interactions, whereas the ANL of a restrained particle only contains ghost interactions. To allow the usage of Newton's third law, the ANL of particle i does not contain particle j if the ANL of particle j already contains particle i. Therefore, the ANL of i stores pair i–j in two cases (Alg. 3): 1) if particle i is active and particle j is either restrained or a ghost particle; 2) if particle i is restrained and particle j is a ghost particle.

Algorithm 3: $BuildANL(i)$

```
1 ANL(i) ← ∅
2 for j ∈ NeighboringCells[i] do
3     if (i ∈ A and (j ∈ R or i ∉ ANL(j)) or j ∈ G then
4         ANL(i) ← j ∪ ANL(i)
```

B. MPI-enabled Single-Pass Incremental Force Update Algorithm

In parallel ARMD, an MPI process contains four force components based on local and ghost interactions: F_{AA}, F_{AR} and F_{RR} due to active-active, active-restrained and restrained-restrained pairs, respectively, and F_{ghost} due to ghost interactions. The force acting on an active particle i can be expressed as:

$$F_i = F_{AA} + F_{AR} + F_{ghost} \tag{1}$$

and the force acting on a restrained particle i is:

$$F_i = F_{AR} + F_{RR} + F_{ghost}. \tag{2}$$

The F_{AA} and F_{AR} force components are associated with active interactions, thus they need to be computed at each timestep. Forces due to ghost particles, F_{ghost}, are computed irrespective of their state also at each timestep (Fig. 1). Although it is possible to further reduce force computations by using the state of ghost particles, it requires at least two more communications among MPI processes. Since distances between restrained particles remain unchanged, the force component based on these distances (F_{RR}) on each MPI process need to be computed only once (at the beginning of the simulation and when a particle switches from active to restrained) and can be retained as long as the involved particles are restrained. Therefore, at each timestep, an MPI process is responsible for locally computing F_{AA}, F_{AR} and F_{ghost} force components, and then for communicating the F_{ghost} force components to neighboring MPI processes (and use Newton's third law). The algorithm that incrementally updates forces is shown in Alg. 4, where C is a list of all particles on an MPI process, and $\mathbf{f}_i^+$ and $\mathbf{f}_i^-$ store F_{RR} and F_{AR} force components, respectively. The total force acting on particle i is stored in $\mathbf{f}$. The proposed algorithm is designed in such a way that it does not require any extra communications and reduces the number of force computations.

Algorithm 4: $IncrementalForceComputation()$

```
1 for i ∈ C do
2     if i ∈ A or i ∈ G then
3         f_i^+ ← 0
4     else
5         f_i^+ ← f_i^+ − f_i^−
6     f_i^− ← 0
7 f ← f^+ + ComputeForces(A, ANL)
```

C. Switching states

In ARMD, at each time step, particles can switch states. Precisely, a particle can either gain enough momenta to become active (to have full dynamics or transition dynamics), or lose enough momenta to become restrained. We refer to these particles as switched particles.

When a particle switches from a restrained state to an active state, we update its ANL using algorithm 3. If a particle switches to a restrained state, we remove active interactions from its ANL. To achieve this, for each switched particle we extract from the ANL a reduced ANL (ANL′)

which contains local interactions only, i.e. local restrained particles (see Alg. 5). Due to force decompositions (eq. 1 and 2), whenever a local particle switches its state the force components F_{AA}, F_{AR} and F_{RR} should be updated based on local interactions alone, without any communications between MPI processes. Since the F_{ghost} component in eq. 1 and 2 is computed for both active and restrained particles, there is no need to compute this force component for a switched particle. Forces acting on switched particles are computed according to the direction of switching (see Alg. 6). If $i \in S_R$, then the force component F_{RR} for this particle is computed (force component F_{AA} and F_{AR} switches to F_{AR} and F_{RR}, respectively) and the F_{RR} force components for local restrained neighbors of this particle are updated. If $i \in S_A$, then the force component F_{RR} is set to zero for this particle and the F_{RR} force components of its local restrained neighbors are updated accordingly. For more information on incremental force computation and the switching process we refer the reader to [17]. These operations are done on each MPI process separately and do not require any communications.

Algorithm 5: $ExtractANL'(i)$

```
ANL'(i) ← ∅
for j ∈ ANL(i) do
    if j ∈ R and (i ∈ R or (i ∈ A and j ∉ S_A ∪ S_R)) then
        ANL'(i) ← ANL'(i) ∪ j
```

Algorithm 6: $SwitchedForces()$

```
for i ∈ S_R do
    ExtractANL'(i)
    f_i^+ ← f_i^+ + ComputeForces(i, ANL'(i))
for i ∈ S_A do
    BuildANL(i)
    ExtractANL'(i)
    f_i^- ← f_i^- + ComputeForces(i, ANL'(i))
```

IV. Load Balancing

LAMMPS uses a Recursive Coordinate Bisection (RCB) algorithm to distribute sub-domains with particles over MPI processes. In RCB, weights can be assigned to particles so that partitioning can be balanced according to these weights. If no weights are assigned to particles, then RCB will try to partition the domain into sub-domains with equal numbers of particles. Because of the adaptive nature of particles, ARMD can lead to a large load imbalance depending on the simulated system. One solution can be to consider a cost function for load balancing as a number of active particles, but this might lead to an imbalance as particles start to switch their state. In ARMD, the workload is not only proportional to the number of active particles, but also to the number of active interactions, which includes active-active and active-restrained interactions. To optimize the load balancing approach used in LAMMPS for ARMD, we introduce an AR load balancing scheme that uses RCB weights of particles and takes both these interactions into account. Instead of assigning equal numbers of particles to each process, the AR load balancer distributes equal weights to each process as in RCB. To address this, we introduce two load factors α and β: α is a weight associated with active particles; β is a weight associated with restrained particles. The weight for a process i can be expressed as $\alpha N_A^i + \beta N_R^i$, where N_A^i and N_R^i are number of active and restrained particles, respectively, assigned to i-th process. If values α and β are the same, then equal numbers of particles are distributed over processes. The choice of parameters α and β is problem dependent and can be done in two ways: 1) user defined and constant; 2) adaptively updated starting from a user defined or default values. Since it is the ratio between α and β which guides the load balancing, parameter α can be set constant and equal to 1, while parameter β can be adaptively updated as

$$\beta = \frac{\sum_{i=0}^{p} t_R^i}{\sum_{i=0}^{p} t_A^i},$$

where the sum is done over all processes (p is the number of processes), t_A^i and t_R^i are time spent on routines associated with active and restrained particles, respectively, on i-th process. In our benchmarks, we use constant parameters $\alpha = 1$ and $\beta = 0.5$.

V. Results and Discussion

To show the performance and scalability of our MPI-enabled implementation of ARMD we present results for a standard Lennard–Jones (LJ) liquid benchmark [4], [20].

The Lennard-Jones potential is often used to model and benchmark van der Waals forces in MD simulations. The potential is also referred to as 12-6 potential and is expressed as:

$$U = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

where ε is a well depth, σ and r are optimal and instantaneous inter-particle distances, respectively. The Lennard-Jones potential is truncated at a cut-off distance $r_c = 2.5\sigma$; beyond this distance, the truncated potential is set to zero. Atoms are placed in a 3D cubic domain according to the fcc lattice, with a lattice constant equal to 0.8442; periodic boundary conditions are used; the initial temperature of the system is set to $T^* = 1.44$; timestep $\Delta t = 0.001$; the neighbor list is updated every 20 timesteps. All parameters are given in standard dimensionless LJ units. Simulations are done in the NVE ensemble for 5000 timesteps. The data on performance and the percentage of restrained particles are averaged over timesteps. All benchmarks are performed using a cluster with 8 nodes equipped with 8/16 CPUs Intel Xeon E5540 and a Gigabit Ethernet network. We run the benchmark both for ARMD and for LAMMPS on one node with different number of processes and on four nodes with four processes per each node (16 processes on total). The results obtained for ARMD

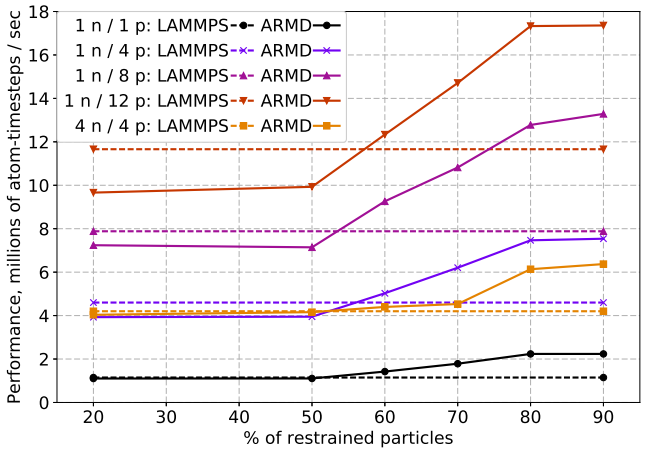
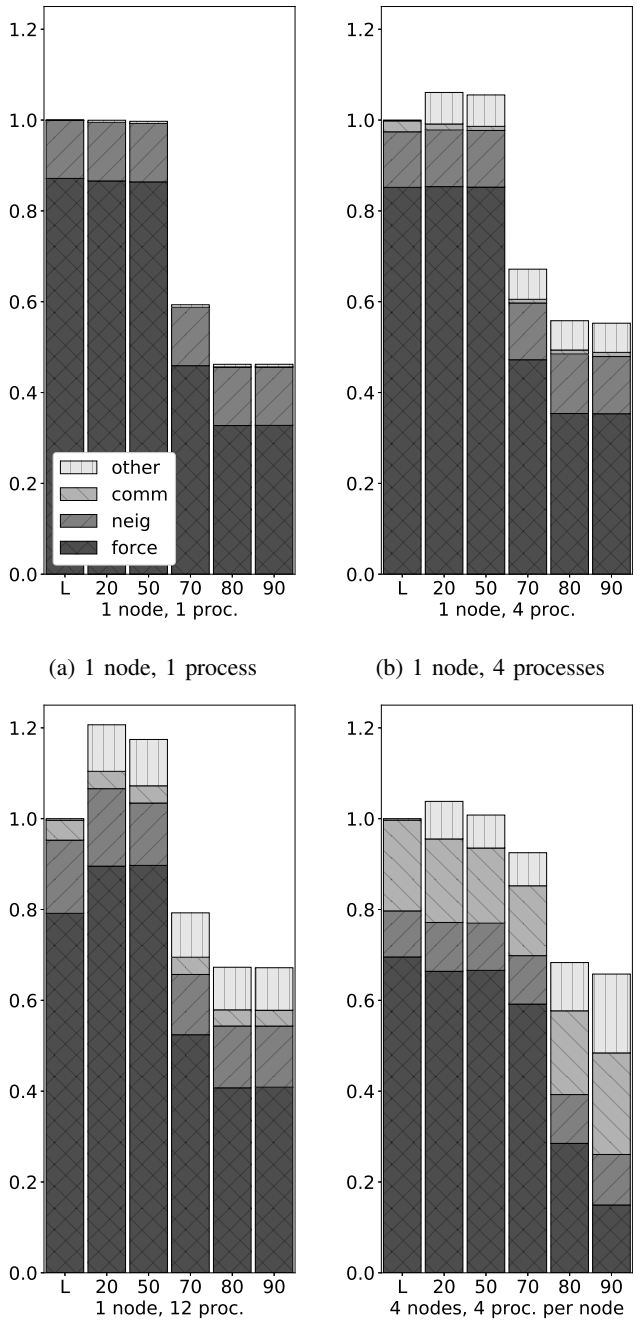


Fig. 2: Performance depending on the percentage of restrained particles for different number of nodes (n) and processes (p) per each node. Performance of non-modified LAMMPS is shown as a reference (dotted lines) — it does not depend on the percentage of restrained particles.

are compared with LAMMPS performance results for the same systems computed on the same equipment. To compare performance, we use a standard performance metric of millions of atom-timesteps per second provided by LAMMPS.

The amount of interaction computations in ARMD depends on the percentage of restrained particles, and, therefore, the performance of ARMD will change based on this percentage. To test the performance of our MPI-enabled implementation of ARMD depending on the percentage of restrained particles, we simulate a system of 864 000 particles using different ARMD parameters $\varepsilon_r, \varepsilon_f$ to get different percentages of restrained particles. The comparison of results obtained with LAMMPS is shown in Fig. 2, with the averaged percentage of restrained particles on x axis. For low percentages of restrained particles, ARMD shows less performance in comparison with LAMMPS due to additional operations introduced in ARMD; ARMD starts to overperform LAMMPS when the percentage of restrained particles in the system reaches some threshold (break-even point). With an increasing number of MPI processes this break-even point shifts to a larger percentage of restrained particles. In this benchmark, ARMD outperforms LAMMPS if more than 60% of restrained particles is present in the system. This threshold depends on a number of factors: the cluster architecture; the simulated problem; the ε_r and ε_f parameters; the distribution of active particles over simulated domain, *e.g.* if all active particles are placed in one region of the domain then there will be fewer number of active–restrained interactions as compared to the case of even distribution of active particles over domain. In order to assess the worst case scenario — active particles are homogeneously distributed over the domain — we apply the same parameters $\varepsilon_r, \varepsilon_f$ for all particles in the simulated system. For non-homogeneous systems, ARMD will give better performance.



(a) 1 node, 1 process

(b) 1 node, 4 processes

(c) 1 node, 12 processes

(d) 4 nodes, 4 processes per node

Fig. 3: Breakdown of wall-clock time for 1 and 4 nodes with 4 processes per each node normalized by LAMMPS timing (L) for different percentage of restrained particles (20%, 50%, 70%, 80%, 90%). Other – Load balancing, ARMD routines for switched particles (ANL and force computations of switched particles), position & momenta update; comm – communications; neig – neighbor list construction; force – force computation.

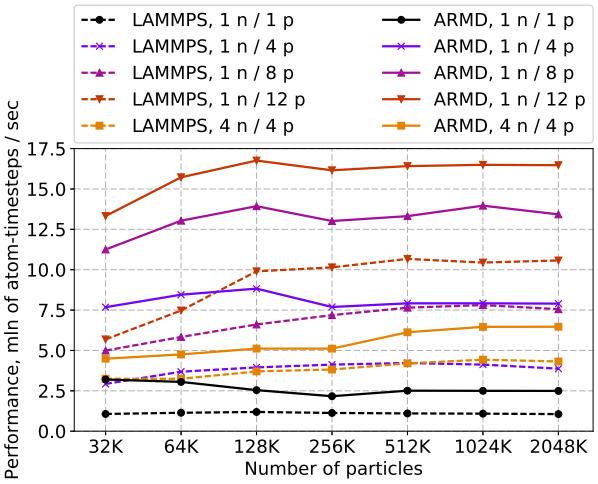


Fig. 4: Performance of ARMD and LAMMPS depending on the number of particles in the system for different number of nodes (n) and processes (p) per each node.

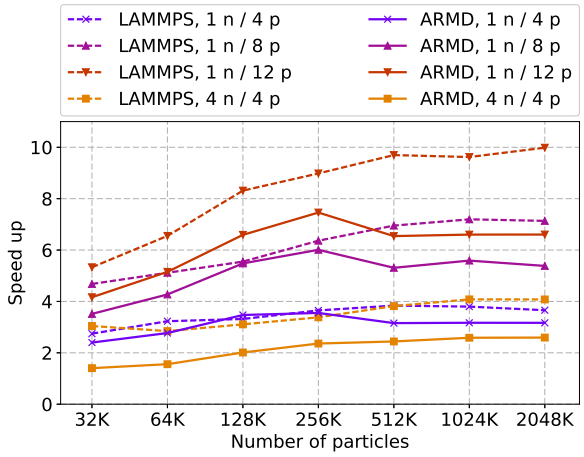


Fig. 5: Speed up of ARMD and LAMMPS for different number of nodes (n) and processes (p) per each node compared to their serial versions

Figure 3 shows a breakdown of wall-clock time normalized by LAMMPS time for different number of processes. With the increasing percentage of restrained particles, ARMD total time decreases due to a decrease in time spent on force computations. This decrease in time occurs after the aforementioned break-even point. If the percentage of restrained particles is smaller than this threshold, ARMD may even provide worse performance than LAMMPS (see Fig. 3b, 3c, 3d) because of additional computations due to switched particles (force computations, updating the ANL). An increase in the number of processes leads to an increase in time spent on ARMD routines due to load-balancing, updating the ANL of particles switched from restrained to active. Since it is necessary to take into account ghost particles while updating the ANL, the more processes are used the more ghost particles are present and should be included in the ANL.

To show the scalability of our parallel implementation of ARMD depending on the number of particles in the simulated system, we fix parameters $\varepsilon_r, \varepsilon_f$ so there will be on average 80% of restrained particles at each timestep (since ARMD is most useful to apply when the percentage of restrained particles in the system is relatively high). ARMD and LAMMPS show good scaling of the performance with the number of particles (Fig. 4). As can be seen from Fig. 3, ARMD total time decreases due to a decrease in time spent on force computations, while time spent on the neighbor list construction and communications stays the same as compared to LAMMPS for the same configuration of nodes and processes. Since in the case of ARMD the force computation part — usually the most parallelizable part — takes less time, the speed up of ARMD compared to ARMD on one core is less than speed up of LAMMPS compared to serial LAMMPS (Fig. 5) due to the Amdahl's law. Speed up of ARMD in comparison with LAMMPS for the same configuration of nodes and processes also decreases with the number of used processes (Fig. 6)

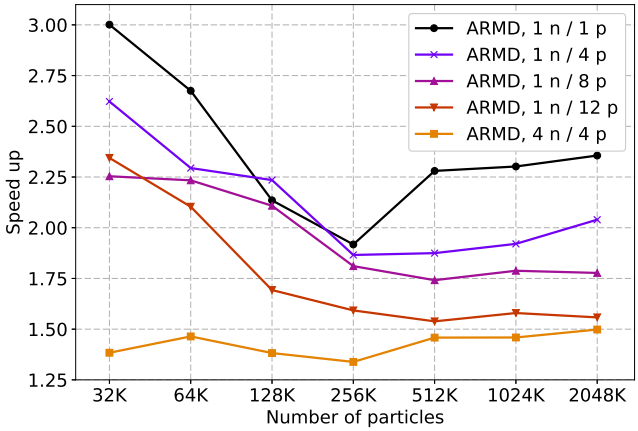


Fig. 6: Speed up of ARMD in comparison with LAMMPS for the same number of number of nodes (n) and processes (p) depending on the number of particles in the system

due to the same reason: the amount of interactions computations per process decreases, while time consumed by other operations common with LAMMPS, i.e. neighbor list construction and communications, stays the same. The fact that the performance of ARMD on 4 nodes is smaller than on 1 node (with 4 processes per node) (Fig. 4) is due to the significant amount of communications between nodes and a high-latency network.

VI. Conclusion

ARMD allows us to accelerate MD computations by decreasing the number of interaction computations and provides better performance than classical MD for systems with a sufficiently large percentage of restrained particles (e.g. more than 60% for the benchmark used in the paper). This threshold depends on the simulated problem and the architecture of

a computational system. When the percentage of restrained particles in the simulated system becomes smaller than this threshold, computations may be switched from ARMD to classical MD, and returned to the usage of ARMD once this percentage reaches the threshold. The suggested paralellization of ARMD using MPI allows us to gain an additional speed up by the usage of multi-core CPUs and distributed systems. The ARMD shows good scalability with the number of particles in the simulated system. However, since ARMD accelerates only the interaction computation part of MD, the speed up of ARMD compared to LAMMPS for the same combination of nodes and processes decreases with the number of processes due to Amdahl's law. To overcome this limitation on distributed systems, we will investigate the possibility of developing a new approach that decreases the amount of data necessary to communicate by taking into account the state (active or restrained) of ghost particles. In future work, we would like to perform benchmarks on clusters with low-latency interconnection networks. We would also like to explore with a more extensive set of parameters, *e.g.* varying the number of nodes and processors, to study the scalability and influence of the number of ghost particles, different load factors for load balancing. Also, we now want to develop parallel ARMD algorithms for central or graphics processing units in combination with parallelization over a distributed system, in order to fully utilize modern clusters.

ACKNOWLEDGMENT

We gratefully acknowledge funding from the Rhone-Alpes Region through the ARC program, and the European Research Council through the ERC Starting Grant n. 307629.

REFERENCES

[1] M. Matsumoto, S. Saito, and I. Ohmine, "Molecular dynamics simulation of the ice nucleation and growth process leading to water freezing," *Nature*, vol. 416, pp. 409–413, Mar. 2002.

[2] F. Benkabou, H. Aourag, and M. Certier, "Atomistic study of zinc-blende CdS, CdSe, ZnS, and ZnSe from molecular dynamics," *Materials Chemistry and Physics*, vol. 66, pp. 10–16, Sept. 2000.

[3] W. A. Curtin and R. E. Miller, "Atomistic/continuum coupling in computational materials science," *Modelling Simul. Mater. Sci. Eng.*, vol. 11, no. 3, p. R33, 2003.

[4] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, Mar. 1995.

[5] D. Hamelberg, J. Mongan, and J. A. McCammon, "Accelerated molecular dynamics: A promising and efficient simulation method for biomolecules," *The Journal of Chemical Physics*, vol. 120, pp. 11919–11929, June 2004.

[6] A. Laio and M. Parrinello, "Escaping free-energy minima," *PNAS*, vol. 99, pp. 12562–12566, Oct. 2002.

[7] E. Krieger and G. Vriend, "New ways to boost molecular dynamics simulations," *J. Comput. Chem.*, vol. 36, pp. 996–1007, May 2015.

[8] P. Larsson, B. Hess, and E. Lindahl, "Algorithm improvements for molecular dynamics simulations," *WIREs Comput Mol Sci*, vol. 1, pp. 93–108, Jan. 2011.

[9] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, C. R. Ho, D. J. Ierardi, I. Kolossvry, J. L. Klepeis, T. Layman, C. McLeavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang, "Anton, a Special-purpose Machine for Molecular Dynamics Simulation," *Commun. ACM*, vol. 51, pp. 91–97, July 2008.

[10] B. Garzn, S. Lago, and C. Vega, "Reaction field simulations of the vapor-liquid equilibria of dipolar fluids," *Chemical Physics Letters*, vol. 231, pp. 366–372, Dec. 1994.

[11] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, pp. 325–348, Dec. 1987.

[12] T. Darden, D. York, and L. Pedersen, "Particle mesh Ewald: An Nlog(N) method for Ewald sums in large systems," *The Journal of Chemical Physics*, vol. 98, pp. 10089–10092, June 1993.

[13] D. Wolf, P. Keblinski, S. R. Phillpot, and J. Eggebrecht, "Exact method for the simulation of Coulombic systems by spherically truncated, pairwise r^{-1} summation," *The Journal of Chemical Physics*, vol. 110, pp. 8254–8282, May 1999.

[14] S. Artemova and S. Redon, "Adaptively Restrained Particle Simulations," *Phys. Rev. Lett.*, vol. 109, p. 190201, Nov. 2012.

[15] S. Redon, G. Stoltz, and Z. Trstanova, "Error Analysis of Modified Langevin Dynamics," *J Stat Phys*, pp. 1–37, June 2016.

[16] Z. Trstanova and S. Redon, "Estimating the speed-up for Adaptively Restrained Langevin Dynamics," *Journal of Computational Physics*, 2017.

[17] Krishna Kant Singh and Stephane Redon, "Adaptively Restrained Molecular Dynamics in LAMMPS," *Modelling Simul. Mater. Sci. Eng.*, p. Accepted, 2017.

[18] P. Gonnet, "A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations," *J. Comput. Chem.*, vol. 28, pp. 570–573, Jan. 2007.

[19] G. Sutmann and V. Stegailov, "Optimization of neighbor list techniques in liquid matter simulations," *Journal of Molecular Liquids*, vol. 125, pp. 197–203, Apr. 2006.

[20] W. M. Brown, P. Wang, S. J. Plimpton, and A. N. Tharrington, "Implementing molecular dynamics on hybrid high performance computers short range forces," *Computer Physics Communications*, vol. 182, pp. 898–911, Apr. 2011.