The Journal of Open Source Software

ARBFN: Arbitrary Externally-Computed Closed-Loop Force Fixes in LAMMPS

Jordan Dehmel [1] **and Jarrod Schiffbauer** [2,3]

1 Dept of Computer Science, Colorado Mesa University, Grand Junction, CO **2** Dept of Physical and Environmental Sciences, Colorado Mesa University, Grand Junction, CO **3** Dept of Mechanical Engineering, University of Colorado Boulder, Grand Junction, CO

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 16 July 2025
Published: unpublished

Statement of Need

The molecular dynamics simulation software LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) (Thompson et al., 2022) provides a scripting language for the easy implementation of experiments: However, it is not all-encompassing. There are many situations in which LAMMPS alone is not sufficient and some external computation must be used (for example, quantum effects (Kohlmeyer et al., 2014) and machine learning control (Rohskopf et al., 2023)). Previous researchers, when confronted with these limitations, have implemented custom interfaces for specific programs. This project outlines the development of a generic protocol for this process. Specifically, we introduce an externally-controlled arbitrary atomic forcing fix within the existing MPI (Message Passing Interface) LAMMPS framework (The Open MPI Project, 2003--2025). This involves an arbitrary-language controller program being instantiated alongside LAMMPS in an MPI runtime, then communicating with all LAMMPS instances whenever the desired fix must be computed. The protocol communicates in JSON (JavaScript Object Notation) strings and assumes no controller linguistic properties beyond a valid MPI implementation.

Introduction

The molecular dynamics simulation software LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) was originally developed by a collaboration between government and industry for the direct, all-atom, simulation of materials and biomolecular properties (see for example, (Thompson et al., 2022), (Petilla et al., 2024), (Bozorgpour, 2024).) Since that time, LAMMPS has been extended in scale and scope to include not only atomistic simulations, but a variety of coarse-grained simulations (Luo & Sommer, 2009), implicit-solvent based simulations (Wang et al., 2016), including multi-particle dissipative (Langevin) dynamics for both passive and active (Dias, 2021), (in 't Veld et al., 2008) colloidal systems.

Since this latter application was the original motivation for the present undertaking, a brief introduction to the underlying problem will be given here to provide context for the application of the present work, both to the simulation of active colloidal systems as well as other potential applications. For a more comprehensive overview of active matter, including experimental systems as well as the Active Brownian Particle (ABP) and other theoretical models, there are a number of reviews available (Boymelgreen et al., 2022), (Vrugt & Wittkowski, 2025). For a more pragmatic introduction to LAMMPS for the simulation of ABPs can be found in the paper by Dias (Dias, 2021). In brief, an active particle (or agent) is an entity which consumes energy and generates its own velocity vector locally, typically via some overall symmetry-breaking, with physical examples ranging from the microscopic, e.g., motile bacteria, synthetic swimmers, sperm cells, to the macroscopic, e.g., animals or vehicles. At the smaller scale, models for so-called active Brownian motion can be implemented by solving the equations of motion for

the particles in an implicit solvent. For a system of N active particles in a viscous bath at temperature T, LAMMPS solves a set of 3N coupled ordinary differential equations of the Langevin type written here in 2D for convenience,

$$m_i\ddot{\vec{r}}_i = F_a\vec{e}_i - \nabla_r V(\vec{r}) - \frac{m_i}{\tau_t}\dot{\vec{r}}_i + \sqrt{\frac{2m_i k_B T}{\tau_t}}\xi(t)$$

and

$$I\ddot{\vec{e}}_i = -\nabla_\theta V(\vec{e}_i) - \frac{\alpha I}{\tau_t}\dot{\vec{e}}_i + \sqrt{\frac{2\alpha I k_B T}{\tau_t}}\xi(t)$$

Here, the activity is imposed as a force, F_a, acting along the orientation vector for the i-th particle, $\vec{e}_i$, where the particle has mass m_i and moment of inertia I_i. Other forces arising from interactions with particles, external fields, or boundaries can be encoded in the position-dependent potential, $V(\vec{r}_i, \vec{e}_i)$. The fluid bath interacts through viscous damping, represented by the translational damping time, τ_t, and related to rotational damping through the factor, α, and subject to fluctuations via the Gaussian noise term, ξ. This treats the fluid implicitly as a passive damping bath that viscously dissipates the motion of the particles and does not include hydrodynamic interactions between particles or particles and boundaries.

Typically, for low-Reynolds number motion, translational, and often rotational, inertia are ignored, and moreover, there is no straightforward way to include the hydrodynamics of the bath. In many instances, what is interesting about active matter systems is their collective behavior, e.g., emergent non-equilibrium phase-separation (Omar et al., 2021) and anomalous fluid-like viscosity (Wilson et al., 2009) in large systems of interacting active agents. Such behavior occurs for models like the ABP model, and while modeling explicit hydrodynamic interactions between confining walls or adjacent active particles directly is possible for small systems and short times, practical simulation of collective behavior is computationally impossible. However, some important and interesting features of such interactions on the dynamics of the active particles themselves could be captured by a heuristic model for the hydrodynamic interactions, which can be implemented as a spatiotemporal modulation of the active free-space velocity of each agent, which may, for example, depend on the evolving particle density distribution itself. External forcing or other parametric inputs are required to include the effects of applied fields or to qualitatively capture the effect of complex hydrodynamic interactions. These can sometimes be facilitated by utilizing LAMMPS fixes. However, while existing LAMMPS fixes are powerful, they are sometimes insufficient. Moreover, a more generalized fix could also be implemented to permit external control of the simulation parameters, e.g., to tie spatiotemporal properties to an external and/or dynamic look-up table, or interaction with an external PINN or AI control system, using the current state of the system as an input.

Although modifying LAMMPS' source code allows efficient implementation of arbitrary fixes, it requires a technical background and imposes `C++` as the language of choice. Especially when working with heavyweight or language-specific systems, it would be far easier to write fixes externally. Thus, we have developed a system for external force fixes as functions of the entire simulated system. Furthermore, in cases where such inter-process-communication-heavy computation would be overkill, we provide a custom externally-determined forcing field.

If we want to have an external system act as a "controller" over our LAMMPS particles, we will need to define a `C++ fix` class which can then be applied in scripts. Instances of this class will need to be able to communicate externally: The easiest way to do this is via the Message Passing Interface (MPI), which LAMMPS already uses. These "workers" will send MPI packets to the controller whenever they need an update, then receiving a result and applying it. For readability and encoding-independence, we will send packets using JavaScript Object Notation

(JSON). To avoid gridlock, we will allow the user to specify a maximal time to await controller response before an error is thrown.

We will call this fix type **`fix arbfn`** (for "arbitrary function" of the state of the simulation).

Note: JSON incurs overhead cost proportional to the size of the message because of its syntax. It would be faster and smaller to send raw encodings of the values used, at the cost of imposing additional restrictions upon the controller language. Thus, we have chosen to pay the overhead for JSON.

In the aforementioned special cases wherein constant MPI communication is unnecessary, we will also define the **`arbfn/ffield`** fix, which determines a static force field via MPI communication at instantiation, then trilinearly interpolating atom positions onto a finite position grid in order to find their forcing values at runtime.

Our final LAMMPS interface for `fix arbfn` is exemplified by the following code.

```
# Every timestep, send all atomic data and
# receive fix data.  If the controller
# takes longer than 50 ms to respond, error.
fix name_1 all arbfn maxdelay 50.0

# Every 100 timesteps, do as above with no
# time limit
fix name_2 all arbfn every 100
```

Likewise, `fix arbfn/ffield` is shown below.

```
# At initialization, retrieve a mesh of 101
# by 201 by 301 nodes. Every timestep,
# perform trilinear interpolation of the
# received force field
fix name_3 all arbfn/ffield 100 200 300

# Every 100 timesteps, send all atom data
# to the controller and refresh the grid
fix name_4 all arbfn/ffield 10 10 10 every 100
```

Details

`fix arbfn`

The first fix provided by the package is `fix arbfn`. It is the most powerful and the slowest. Every time this fix is called, its atoms are sent off to the controller over MPI. The controller then determines some amount of force to add to each atom, sending it back to LAMMPS to implement. The controller is also allowed to send a "waiting" packet indicating that LAMMPS should wait another few milliseconds. If no response is received within some specified time limit, LAMMPS will error. Since there may be arbitrarily many LAMMPS instances running, the controller may choose to await all the data or to send back data immediately. It is slightly faster to send back data one controller at a time, but limits the capabilities of the fix (for instance, a fix pushing atoms towards the center of mass could not be implemented).

This fix can be used to implement frame-by-frame control of the forces of atoms based on the state of some external system. As a frivolous example, imagine a LAMMPS simulation where forces could be applied by using a physical joystick. It can also be used to apply forces based on attributes not feasibly implementable solely within LAMMPS.

The protocol for `fix arbfn` is shown in Figure 1. Note that communication between LAMMPS and the "worker" (fix object instance) is virtually free, while communication between the

worker and the controller is very expensive.

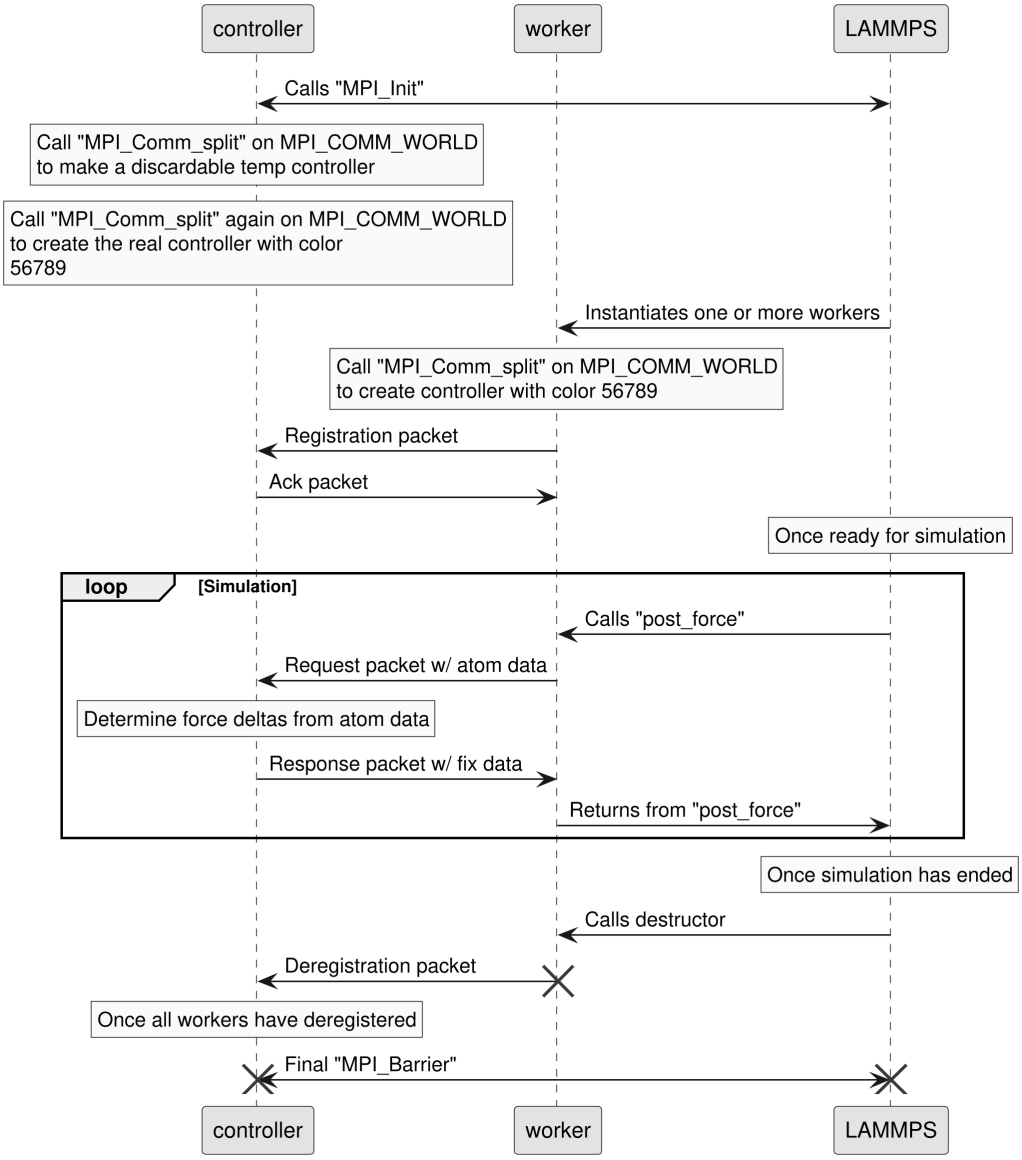

Figure 1: `fix arbfn` protocol.

While necessary for some use cases, this fix is painfully slow: A simulation that may take only a few minutes without it will instead take hours.

`fix arbfn/ffield`

Evolving from the aforementioned MPI delays is the `arbfn/ffield` fix. This takes in some spatial grid of nodes at instantiation via MPI, then interpolates between them to find specific force field values.

This fix is *not* able to update frame-by-frame, and the interpolation it does is position-only (velocity, existing force, and orientation cannot come into play), but is in exchange about 100 times faster.

The protocol for the `arbfn/ffield` fix is shown in Figure 2. Note that there are no longer

costly MPI calls within the simulation loop, and thus the simulation will perform much better.

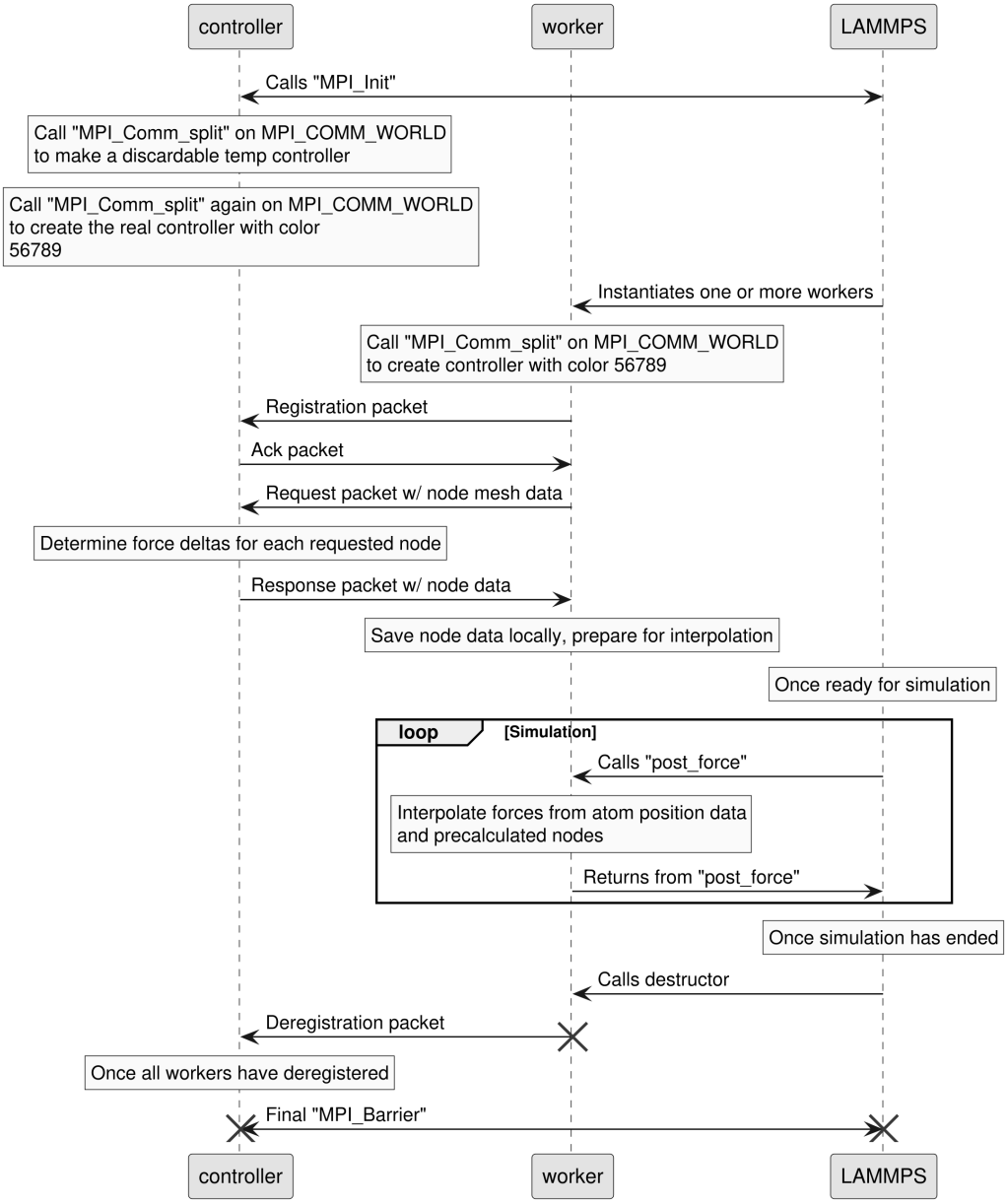

Figure 2: `fix arbfn/ffield` protocol.

Although the difference between Figure 1 and Figure 2 may seem trivial, the omission of the controller from the simulation loop allows Figure 2 to run orders of magnitude faster.

`fix arbfn/ffield` with `every n`

Our final protocol addresses the holes in what `fix arbfn/ffield` can compute. Instead of receiving a single interpolation grid at the beginning and using it for the entire simulation, the `every n` argument allows us to dynamically update the grid every n time steps. Specifically, every n-th time step, the worker sends all of its atomic data to the controller and receives a new interpolation grid in return.

This protocol allows more generality at the cost of speed, while still being (generally) faster

than `arbfn`. It allows a degree of dependency of the force field on the atomic data (e.g. center of mass) which was previously impossible.

The protocol for `fix arbfn/ffield` with the `every n` argument is shown in Figure 3. Note that this reintroduces the costly IPC during the simulation, but is still more sparse than `arbfn`.

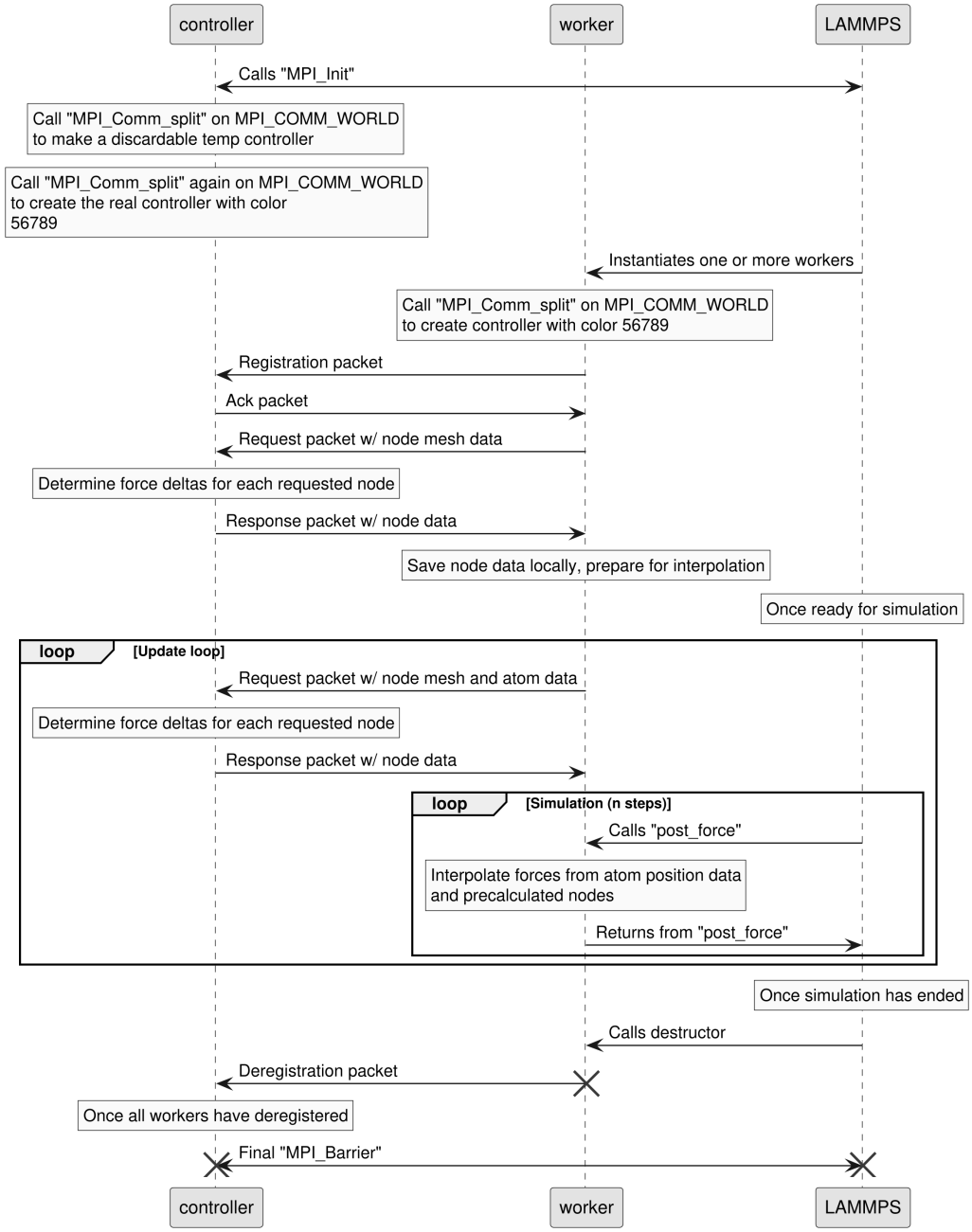

Figure 3: `fix arbfn/ffield` protocol when used with the `every n` argument.

`arbfn/ffield` is a special case of `arbfn/ffield every n` where n is larger than the length of the simulation. Internally, this is represented by `every 0`. The other limit case, `every 1`, is *nearly* `arbfn`: It communicates every frame (and therefore is at least as slow), but the atom forces are ultimately still interpolated according to the grid, rather than directly controlled.

Running Simulations

In order to keep ARBFN MPI communication from interfering with internal LAMMPS communication, we must run LAMMPS with the `-mpicolor ...` command-line argument. The number following this must be anything **except** 56789 (this color is reserved for internal package communication). On UNIX systems, this takes the form that follows.

```
mpirun -n 1 ./controller : \
    -n ${num_lmp_threads} \
    lmp -mpicolor 123 \
    -in input_script.lmp
```

Performance Testing

Our performance experiments were carried out with small simulations. We kept a constant particle density of 2 atoms per unit area in a 2D system (where the square box's edge length was varied) to avoid minimization problems arising from varying particle count directly. The controllers used were minimal so as to test LAMMPS performance rather than external computation time: Although they always applied at least some forces to some atoms, no significant computation was involved. These experiments are intended to demonstrate the trend of the running times in small simulations, as well as to corroborate that our fixes scale proportionally to a no-fix system. A python script was used to automate the running of scripts.

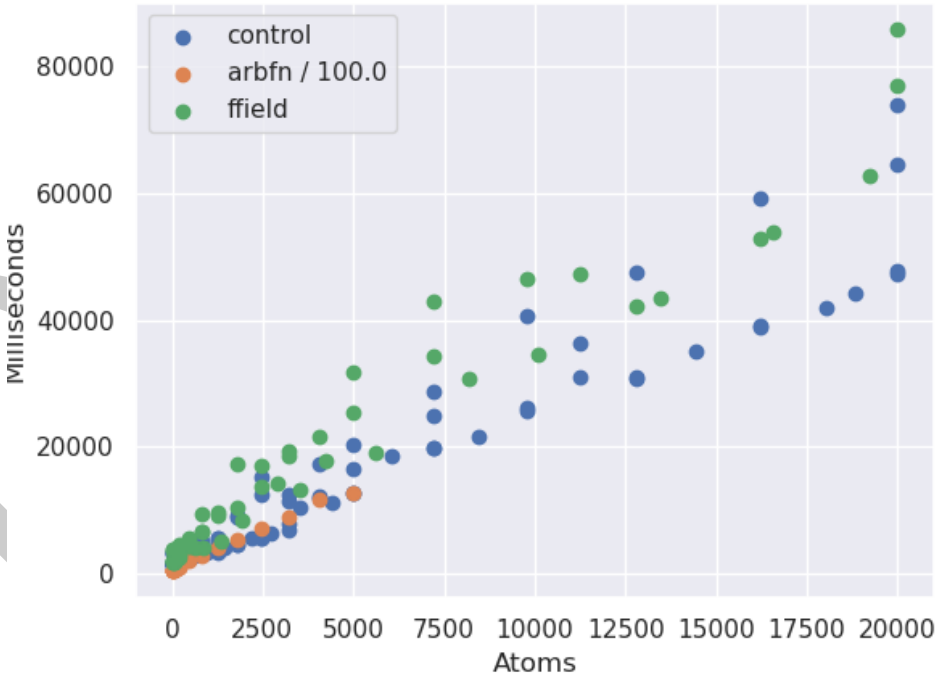

Figure 4: Comparing `fix arbfn` to control and `fix arbfn/ffield`. The runtime of the former has been divided by 100 to fit it on the graph. While still appearing linear with respect to the number of atoms, it runs *much* slower than the latter two (which are about the same speed).

Figure 4 demonstrates the sharp slope of `fix arbfn`: IPC is extremely costly, and frame-by-frame updates should be avoided whenever possible. That being said, the time usage appears to scale proportionally to control as expected. The `fix arbfn/ffield` results appear to scale

with a much smaller coefficient, as expected: This is a much more usable fix, although its use case is more narrow.

Conclusion

We discussed the implementation of the `ARBFN` package for LAMMPS, including a brief analysis of its performance as simulations scaled. This package allows LAMMPS fixes which are determined at runtime by arbitrary external "controllers", either one-and-done (`arbfn/ffield`) or frame-by-frame (`fix arbfn`). Initial testing shows that the former performs nearly as well as unmodified LAMMPS, while the latter performs about 2 orders of magnitude worse. We also provided an option to update the interpolation force field as a function of atom data periodically throughout the simulation's life cycle. The package is provided as supplementary material with this paper.

Related Work

The source code of this package draws from the QMMM package (Kohlmeyer et al., 2014) and the LAMMPS codebase (Thompson et al., 2022) for the framework of MPI communication. It also used the `BROWNIAN` package, "Extending and Modifying LAMMPS" (Mubin & Li, 2021), and (Albano et al., 2021) as a basis for force modifications. Packages like QMMM and FitSNAP (Rohskopf et al., 2023) use custom `C++` to interface with external controllers: Our package seeks to standardize such communication. The external interfacing of our package is similar to the built-in LAMMPS `python` wrapper (used to similar effect in e.g. (Do & Hurák, 2024)), but allows more linguistic generality. Indeed, `python` has proved a popular choice in the composure and control of MD simulations: Source (Balasubramanian et al., 2016) uses template `python` scripts to control and scale simulations.

Visualization has been hand-in-hand with control as a primary concern in molecular dynamics. Systems like (Koutek et al., 2002) and (Stone et al., 2001) allow hand-modification of particle forces within their systems, even though their concern is primarily in human-simulation interface rather than software-simulation.

Acknowledgements

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 2126451 and 2430509, as well as by the National Aeronautics and Space Administration (NASA) MOSAICS grant No. 80NSSC24K0415. The authors gratefully acknowledge the NSF's, NASA's, and Colorado Mesa University's support.

References

Albano, A., Guillou, E. le, Danzé, A., Moulitsas, I., Sahputra, I. H., Rahmat, A., Duque-Daza, C. A., Shang, X., Ching Ng, K., Ariane, M., & Alexiadis, A. (2021). How to modify LAMMPS: From the prospective of a particle method researcher. *ChemEngineering*, *5*(2). https://doi.org/10.3390/chemengineering5020030

Balasubramanian, V., Bethune, I., Shkurti, A., Breitmoser, E., Hruska, E., Clementi, C., Laughton, C., & Jha, S. (2016). Extasy: Scalable and flexible coupling of md simulations and advanced sampling techniques. *2016 IEEE 12th International Conference on e-Science (e-Science)*, 361–370.

Boymelgreen, A., Schiffbauer, J., Khusid, B., & Yossifon, G. (2022). Synthetic electrically driven colloids: A platform for understanding collective behavior in soft matter. *Current Opinion in Colloid and Interface Science*, *60*. https://doi.org/10.1016/j.cocis.2022.101603

Bozorgpour, R. (2024). *Computational explorations in biomedicine: Unraveling molecular dynamics for cancer, drug delivery, and biomolecular insights using LAMMPS simulations*. https://arxiv.org/abs/2311.13000

Dias, C. S. (2021). *Molecular dynamics simulations of active matter using LAMMPS*. http://arxiv.org/abs/2102.10399

Do, L., & Hurák, Z. (2024). Feedback control in molecular dynamics simulations using LAMMPS. *2024 European Control Conference (ECC)*, 1065–1070.

in 't Veld, P. J., Plimpton, S. J., & Grest, G. S. (2008). Accurate and efficient methods for modeling colloidal mixtures in an explicit solvent using molecular dynamics. *Computer Physics Communications*, *179*(5), 320–329. https://doi.org/https://doi.org/10.1016/j.cpc.2008.03.005

Kohlmeyer, A., Ippolito, M., & Cavazzoni, C. (2014). *QM/MM support library*.

Koutek, M., Hees, J. van, Post, F. H., Bakker, A., & others. (2002). Virtual spring manipulators for particle steering in molecular dynamics on the ResponsiveWorkbench. *EGVE*, 53–62.

Luo, C., & Sommer, J.-U. (2009). Coding coarse grained polymer model for LAMMPS and its application to polymer crystallization. *Computer Physics Communications*, *180*(8), 1382–1391. https://doi.org/https://doi.org/10.1016/j.cpc.2009.01.028

Mubin, Dr. S., & Li, J. (2021). *Extending and modifying LAMMPS writing your own source code: A pragmatic guide to extending LAMMPS as per custom simulation requirements*. Packt Publishing. ISBN: 9781800562264

Omar, A. K., Klymko, K., Grandpre, T., & Geissler, P. L. (2021). Phase diagram of active brownian spheres: Crystallization and the metastability of motility-induced phase separation. *Physical Review Letters*, *126*. https://doi.org/10.1103/PhysRevLett.126.188002

Petilla, C. E., Cruz, C. J. D., & Mahinay, C. L. (2024). Mechnical properties of si(1-x)-c(x): Strength and stiffness of materials using LAMMPS molecular dynamics simulation. *Jap. J. Appl. Phys.*, *63*. https://doi.org/10.35848/1347-4065/ad66d7

Rohskopf, A., Sievers, C., Lubbers, N., Cusentino, M. a., Goff, J., Janssen, J., McCarthy, M., Zapiain, D. M. O. de, Nikolov, S., Sargsyan, K., Sema, D., Sikorski, E., Williams, L., Thompson, A. p., & Wood, M. a. (2023). FitSNAP: Atomistic machine learning with LAMMPS. *Journal of Open Source Software*, *8*(84), 5118. https://doi.org/10.21105/joss.05118

Stone, J. E., Gullingsrud, J., & Schulten, K. (2001). A system for interactive molecular dynamics simulation. *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, 191–194.

The Open MPI Project. (2003--2025). *Open MPI v5.0.x* (Version v5.0.x). https://docs.open-mpi.org/en/v5.0.x/#.

Thompson, A. P., Aktulga, H. M., Berger, R., Bolintineanu, D. S., Brown, W. M., Crozier, P. S., Veld, P. J. in 't, Kohlmeyer, A., Moore, S. G., Nguyen, T. D., Shan, R., Stevens, M. J., Tranchida, J., Trott, C., & Plimpton, S. J. (2022). LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp. Phys. Comm.*, *271*, 108171. https://doi.org/10.1016/j.cpc.2021.108171

Vrugt, M. te, & Wittkowski, R. (2025). Metareview: A survey of active matter reviews. *Euro. Phys. J. E*, *48*. https://doi.org/10.1140/epje/s10189-024-00466-z

Wang, Y., Sigurdsson, J. K., & Atzberger, P. J. (2016). Fluctuating hydrodynamics methods for dynamic coarse-grained implicit-solvent simulations in LAMMPS. *SIAM Journal on Scientific Computing*, *38*(5), S62–S77. https://doi.org/10.1137/15M1026390

Wilson, L. G., Harrison, A. W., Schofield, A. B., Arlt, J., & Poon, W. C. K. (2009). Passive

and active microrheology of hard-sphere colloids. *Journal of Physical Chemistry B*, *113*, 3806–3812. https://doi.org/10.1021/jp8079028