

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

Jordan Dehmel and Jarrod Schiffbauer

*Colorado Mesa University, 1100 North Avenue, Grand Junction, CO 81501-3122*

The molecular dynamics simulation software LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) 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, including quantum effects [1] and machine learning control [2] of a simulation. Such situations provide some custom interface for a specific program to apply forces on the simulated particles. 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 [3] [4]. 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.

## I. INTRODUCTION

LAMMPS fixes are powerful, but sometimes insufficient. Although modifying LAMMPS’ source code allows efficient implementation of arbitrary fixes, it requires a technical background and for C++ to be the language used. 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 arbfm` (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 `arbfm/ffield` fix, which determines a static force field via MPI communication at instantiation, then trilinearly interpolating atom positions onto a finite grid in order to find their forcing values at runtime.

Our final LAMMPS interface for `fix arbfm` 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 arbfm maxdelay 50.0

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

Likewise, `fix arbfm/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 arbfm/ffield 100 200 300

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

More about the implementation of these fixes will be given in section II.

## II. DETAILS

### A. `fix arbfm`

The first fix provided by the package is `fix arbfm`. 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 arbfm` 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.

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.

### B. `fix arbfm/ffield`

Evolving from the aforementioned MPI delays is the `arbfm/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 `arbfm/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.

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

### C. `fix arbfm/ffield with every n`

Our final protocol addresses the holes in what `fix arbfm/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 IIA. 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 arbfm/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 IIA.

II B is a special case of II C 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* IIA: 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.

## D. Running Simulations

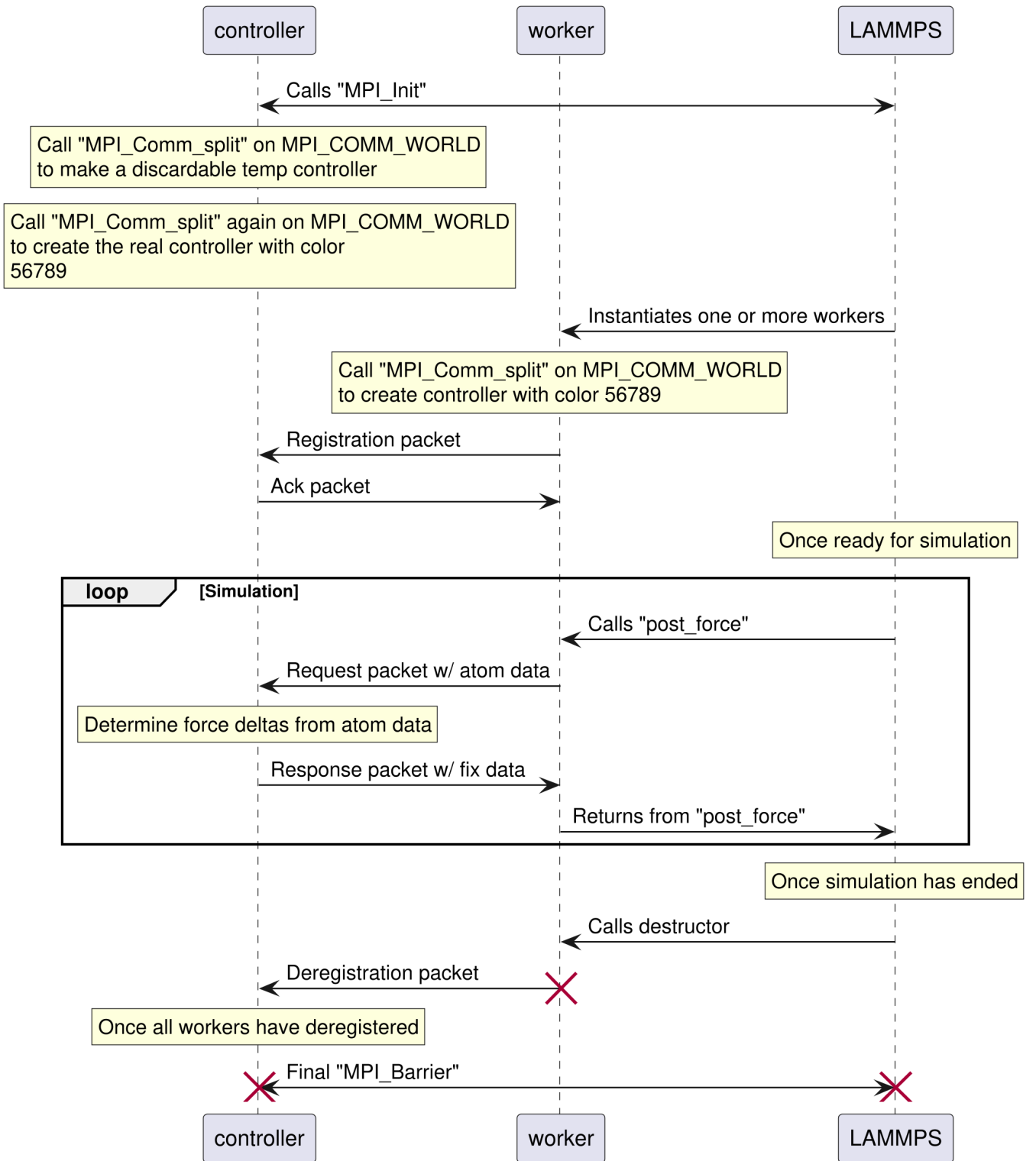
In order to keep ARBFN MPI communication from interfering with internal LAMMPS communication, we must run LAMMPS with the `-mpicolor ...` command-line argument. The number following this must be anything **except** 56789 (this color is reserved for internal package communication). On UNIX systems, this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

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

## III. 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 demonstrates the sharp slope of `fix arbfm`: 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 arbfm/ffield` results appear to scale with a much smaller coefficient, as expected: This is a much more usable fix, although its usecase is more narrow.

FIG. 1: `fix arbfm` protocol.

#### IV. 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 (`arbfm/ffield`) or frame-by-frame (`fix arbfm`). Initial

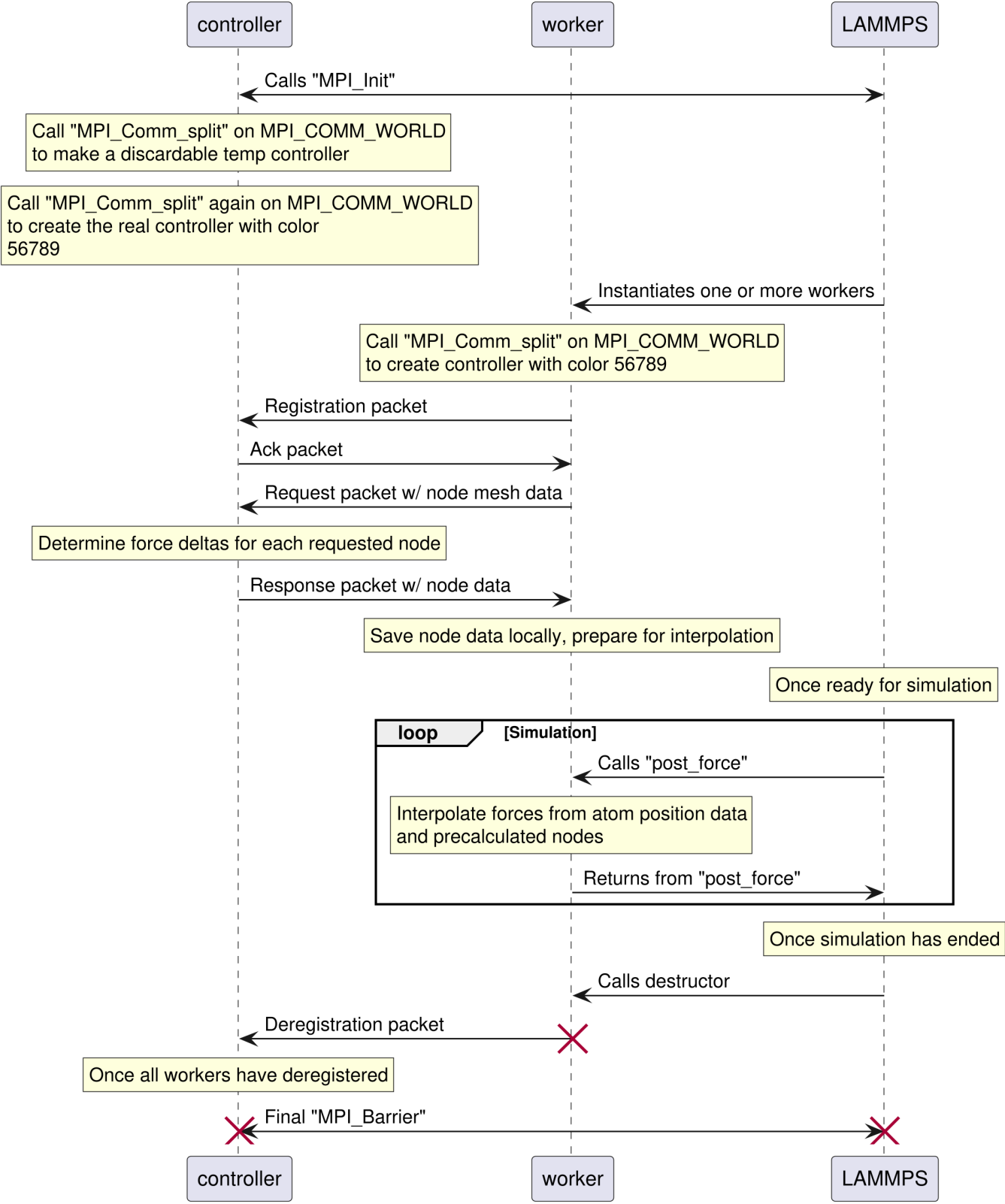


FIG. 2: `fix arbfn/ffield` protocol.

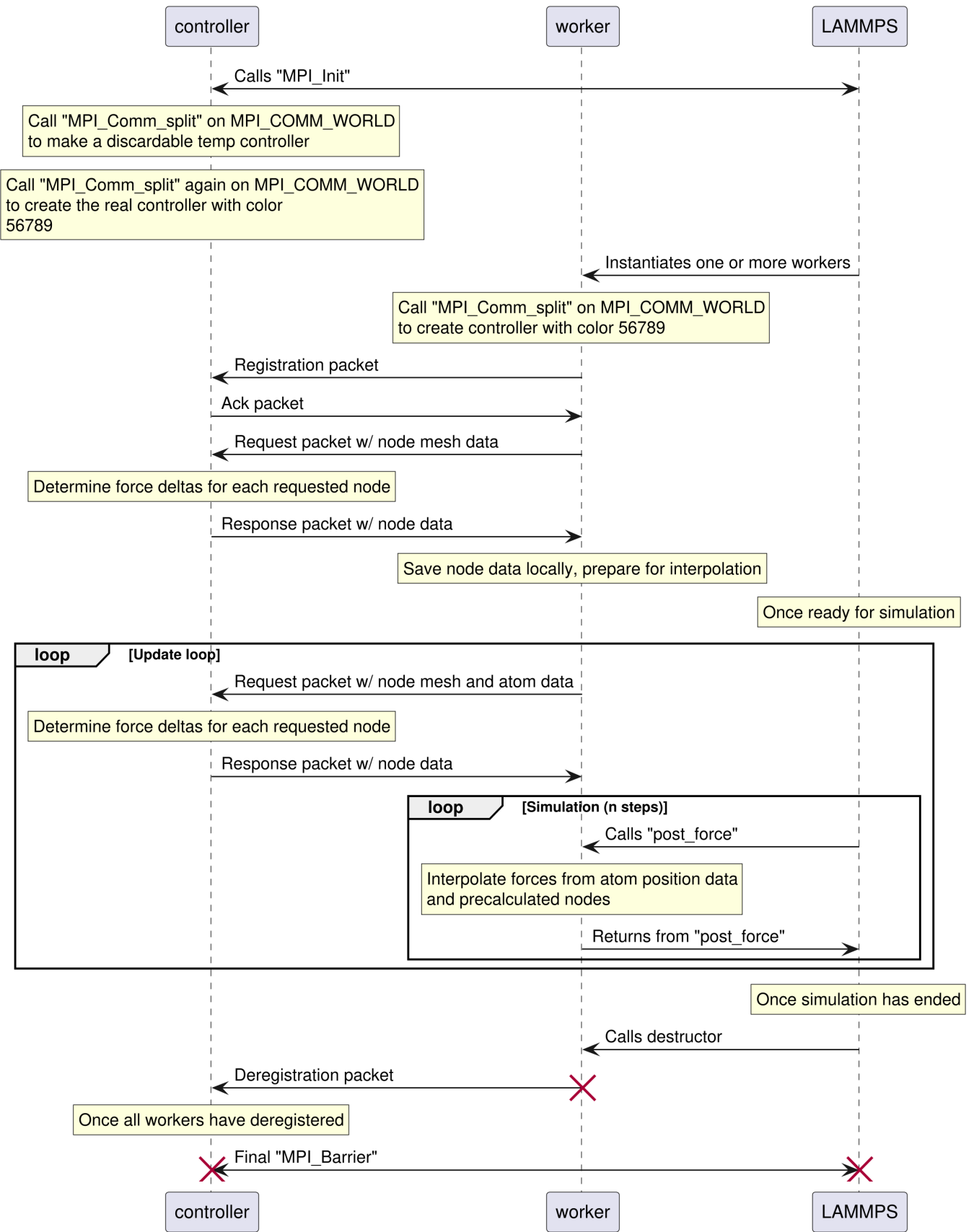


FIG. 3: `fix arbfn/ffield` protocol when used with the `every n` argument.

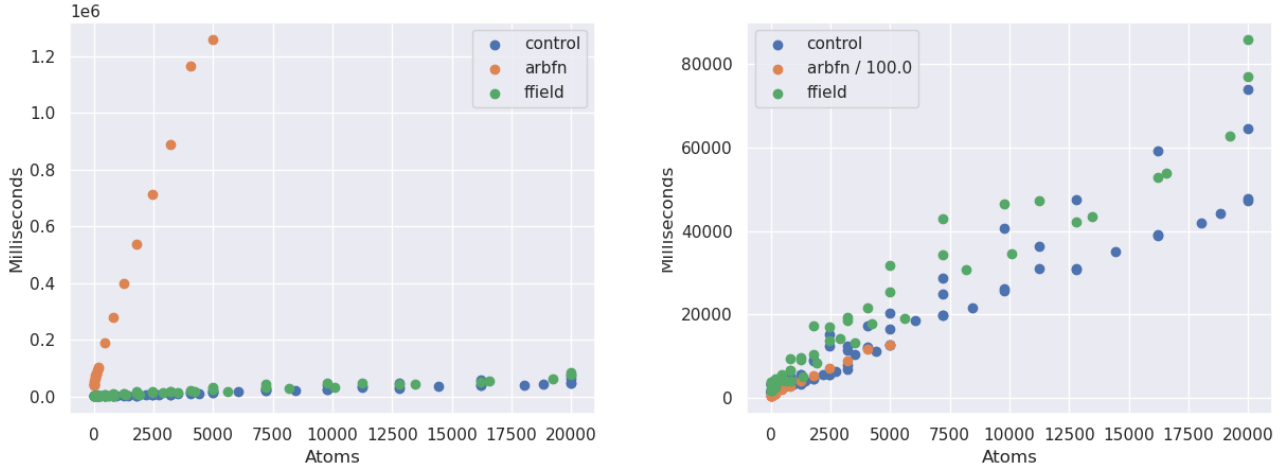


FIG. 4: Comparing `fix arbfm` to `control` and `fix arbfm/ffield`. While still appearing linear with respect to the number of atoms, it runs *much* slower than the latter two (which are about the same speed).

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.

## V. FUTURE WORK

FUTURE WORK GOES HERE

## VI. RELATED WORK

The source code of this package draws from the QMMM package [1] and the LAMMPS codebase [4] for the framework of MPI communication. It also used the BROWNIAN package, “Extending and Modifying

LAMMPS” [5], and [6] as a basis for force modifications. Packages like QMMM [1] and FitSNAP [2] 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. [7]), but allows more linguistic generality. Indeed, python has proved a popular choice in the composure and control of MD simulations: Source [8] 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 [9] and [10] allow hand-modification of particle forces within their systems, even though their concern is primarily in human-simulation interface rather than software-simulation.

## ACKNOWLEDGMENTS

Supported by NSF grant 2126451. Based upon work partially funded by Colorado Mesa University.

- 
- [1] A. Kohlmeier, M. Ippolito, and C. Cavazzoni, Qm/mm support library (2014).
  - [2] A. Rohskopf, C. Sievers, N. Lubbers, M. Cusentino, J. Goff, J. Janssen, M. McCarthy, D. M. O. de Zapain, S. Nikolov, K. Sargsyan, D. Sema, E. Sikorski, L. Williams, A. Thompson, and M. Wood, Fitsnap: Atomistic machine learning with lammps, *Journal of Open Source Software* **8**, 5118 (2023).
  - [3] The Open MPI Project, Open mpi v5.0.x, <https://docs.open-mpi.org/en/v5.0.x/#> (2003–2025).
  - [4] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolinteanu, W. M. Brown, P. S. Crozier, P. J. in ’t Veld, A. Kohlmeier, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton,

LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales, *Comp. Phys. Comm.* **271**, 108171 (2022).

- [5] D. S. Mubin and J. Li, *Extending and Modifying LAMMPS Writing Your Own Source Code: A pragmatic guide to extending LAMMPS as per custom simulation requirements* (Packt Publishing, 2021).
- [6] A. Albano, E. le Guillou, A. Danzé, I. Moulitsas, I. H. Sahputra, A. Rahmat, C. A. Duque-Daza, X. Shang, K. Ching Ng, M. Ariane, and A. Alexiadis, How to modify lammps: From the prospective of a particle method researcher, *ChemEngineering* **5**, 10.3390/chemengineering5020030 (2021).
- [7] L. Do and Z. Hurák, Feedback control in molecular dy-

- namics simulations using lammmps, in *2024 European Control Conference (ECC)* (IEEE, 2024) pp. 1065–1070.
- [8] V. Balasubramanian, I. Bethune, A. Shkurti, E. Breitmöser, E. Hruska, C. Clementi, C. Laughton, and S. Jha, Extasy: Scalable and flexible coupling of md simulations and advanced sampling techniques, in *2016 IEEE 12th International Conference on e-Science (e-Science)* (IEEE, 2016) pp. 361–370.
- [9] M. Koutek, J. van Hees, F. H. Post, A. Bakker, *et al.*, Virtual spring manipulators for particle steering in molecular dynamics on the responsiveworkbench., in *EGVE* (2002) pp. 53–62.
- [10] J. E. Stone, J. Gullingsrud, and K. Schulten, A system for interactive molecular dynamics simulation, in *Proceedings of the 2001 symposium on Interactive 3D graphics* (2001) pp. 191–194.