ARBFN: Arbitrary Externally-Computed 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 and machine learning control of a simulation. Such situations provide some 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 [1] [2] [3]. 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 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 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 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, send all atomic data
# without expecting any fix data back.
fix name_2 all arbfn every 100 dumponly
```

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
```

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

II. RELATED WORK

The source code of this package draws from the QMMM package [4] and the LAMMPS codebase [5] for the framework of MPI communication. It also used the `BROWNIAN` package and "Extending and Modifying LAMMPS" [1] as a basis for force modifications.

The external interfacing of our package is similar to the built-in LAMMPS `python` wrapper (used to similar effect in e.g. [6]), but allows more linguistic generality.

III. DETAILS

A. `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.

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 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.

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 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 III A. 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 III A.

III B is a special case of III 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* III A: 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
```

IV. 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 directly. 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 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 usecase is more narrow.

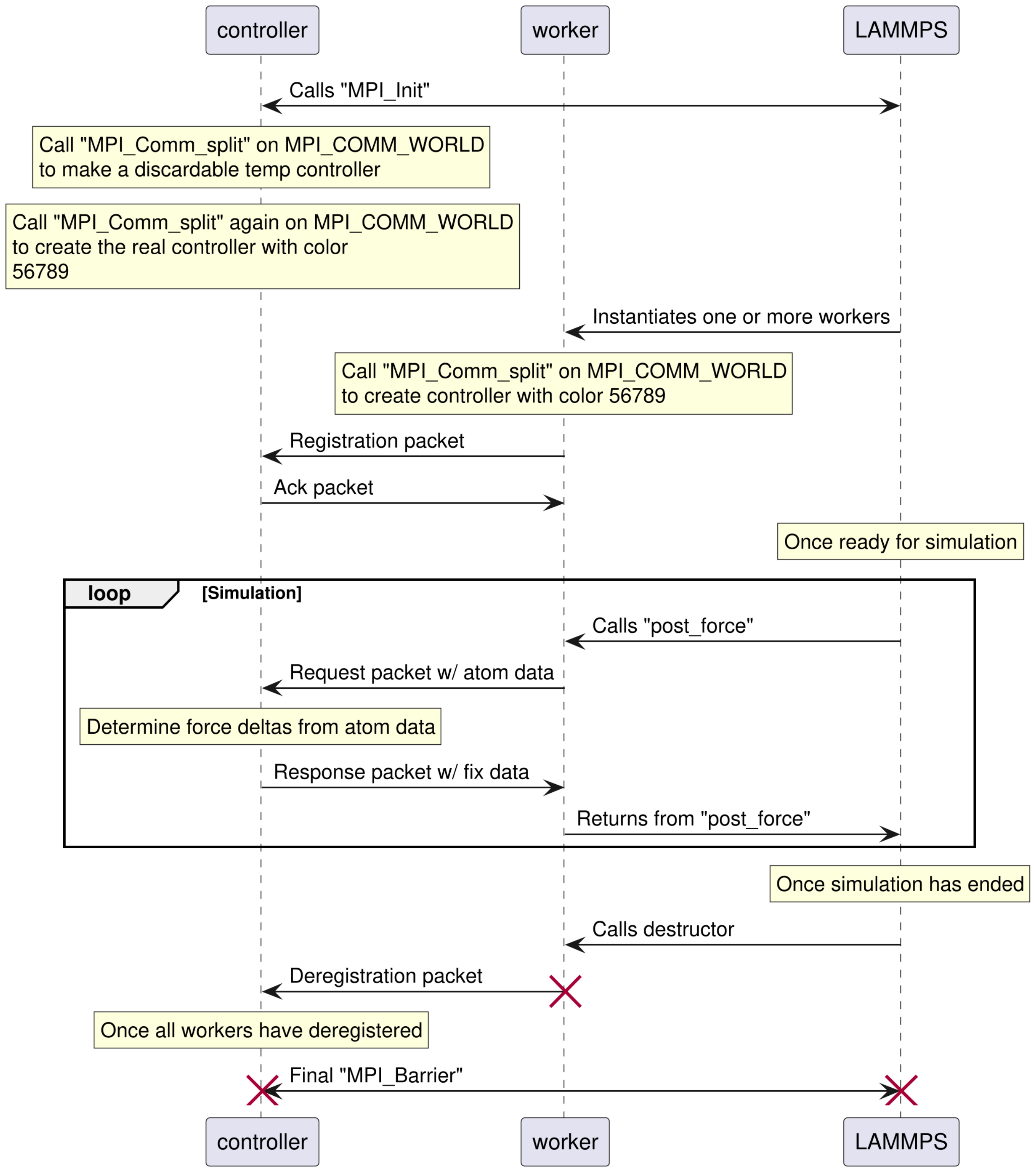

controller
worker
LAMMPS
Calls "MPI_Init"
Call "MPI_Comm_split" on MPI_COMM_WORLD to make a discardable temp controller
Call "MPI_Comm_split" again on MPI_COMM_WORLD to create the real controller with color 56789
Instantiates one or more workers
Call "MPI_Comm_split" on MPI_COMM_WORLD to create controller with color 56789
Registration packet
Ack packet
Once ready for simulation
loop
[Simulation]
Calls "post_force"
Request packet w/ atom data
Determine force deltas from atom data
Response packet w/ fix data
Returns from "post_force"
Once simulation has ended
Calls destructor
Deregistration packet
Once all workers have deregistered
Final "MPI_Barrier"
controller
worker
LAMMPS

FIG. 1: `fix arbfn` protocol.

V. 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

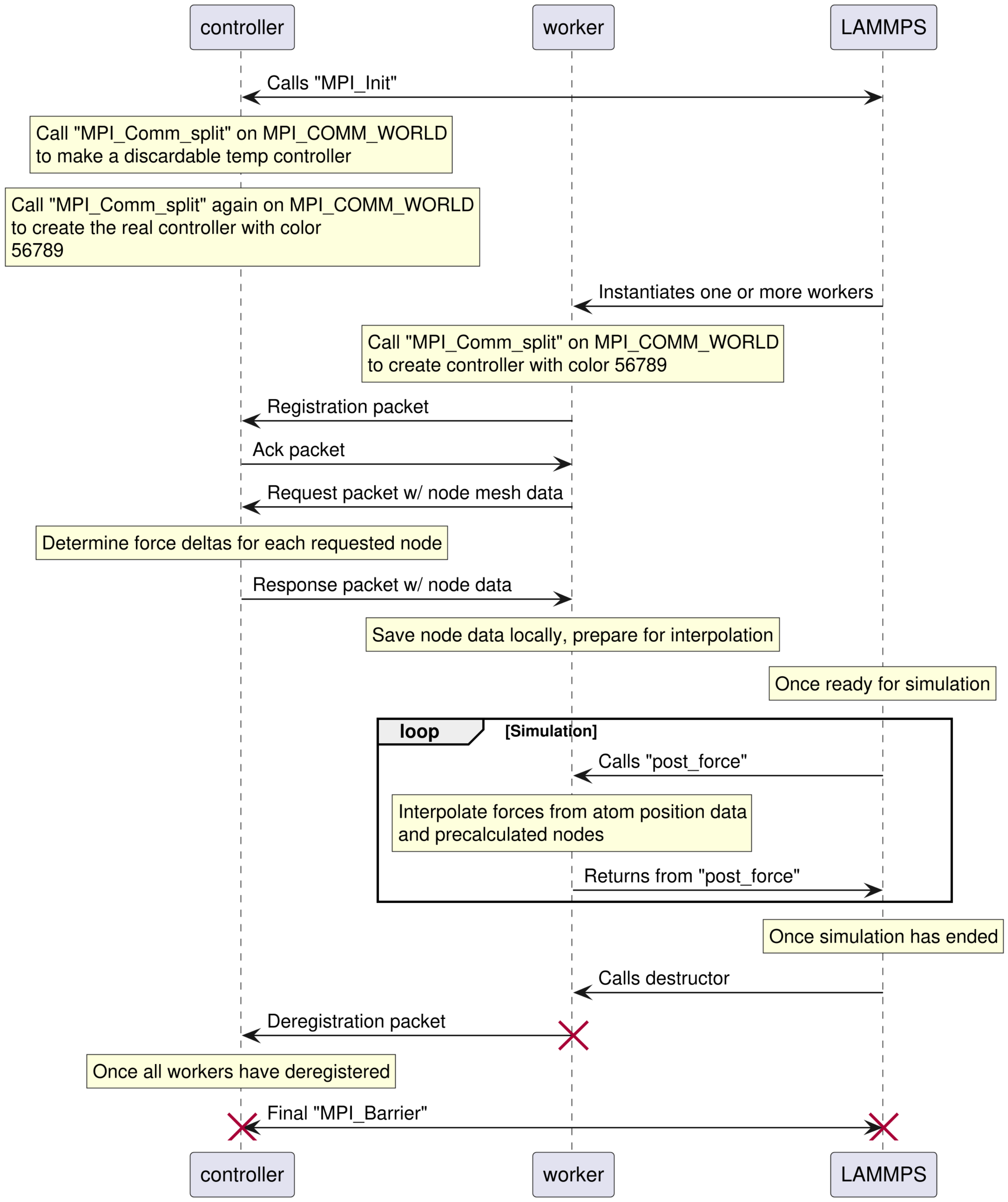

controller
worker
LAMMPS
Calls "MPI_Init"
Call "MPI_Comm_split" on MPI_COMM_WORLD to make a discardable temp controller
Call "MPI_Comm_split" again on MPI_COMM_WORLD to create the real controller with color 56789
Instantiates one or more workers
Call "MPI_Comm_split" on MPI_COMM_WORLD to create controller with color 56789
Registration packet
Ack packet
Request packet w/ node mesh data
Determine force deltas for each requested node
Response packet w/ node data
Save node data locally, prepare for interpolation
Once ready for simulation
loop
[Simulation]
Calls "post_force"
Interpolate forces from atom position data and precalculated nodes
Returns from "post_force"
Once simulation has ended
Calls destructor
Deregistration packet
Once all workers have deregistered
Final "MPI_Barrier"
controller
worker
LAMMPS

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

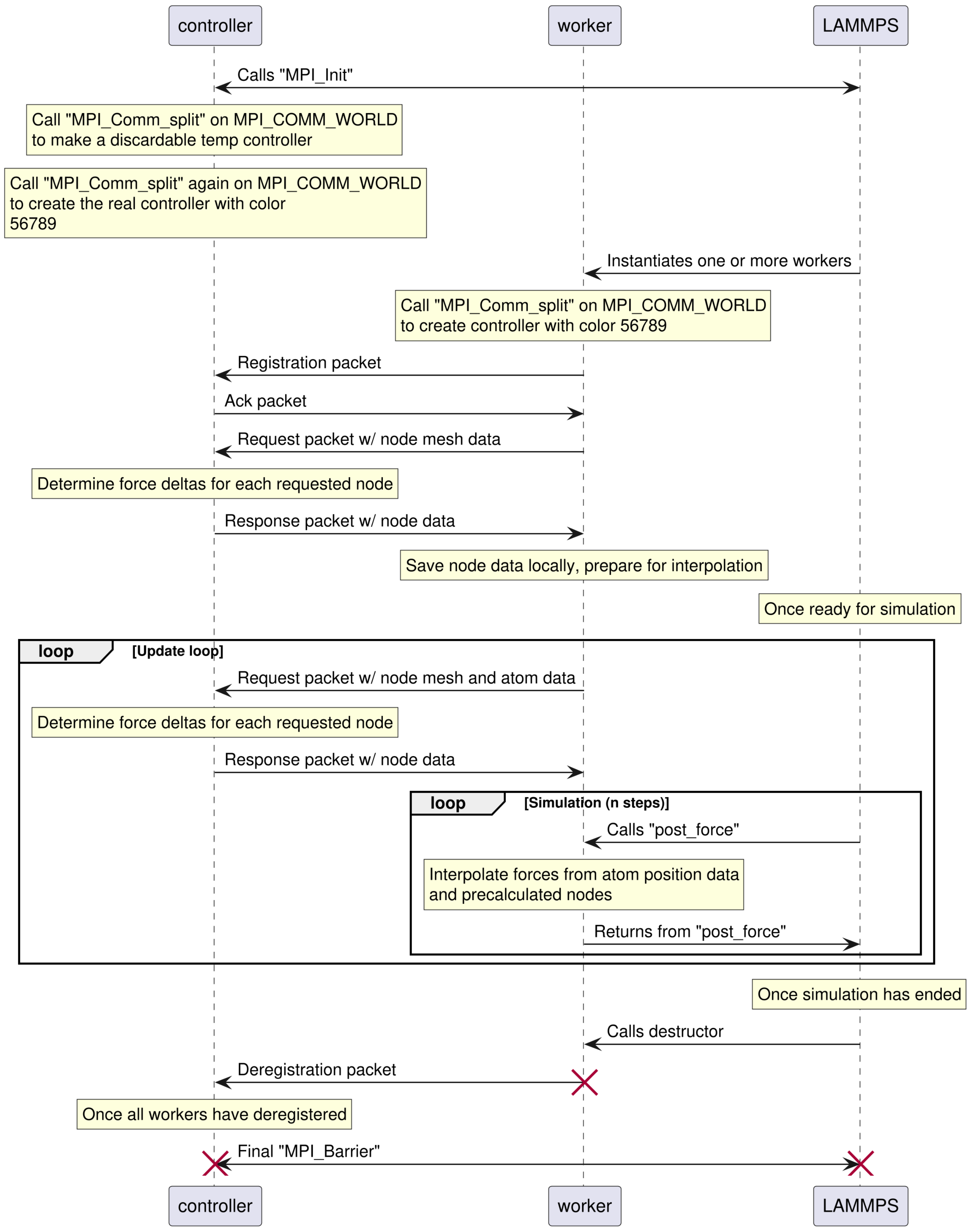
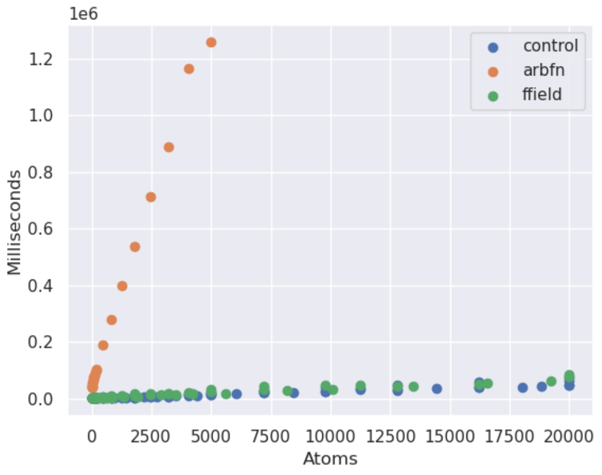

controller
worker
LAMMPS
Calls "MPI_Init"
Call "MPI_Comm_split" on MPI_COMM_WORLD to make a discardable temp controller
Call "MPI_Comm_split" again on MPI_COMM_WORLD to create the real controller with color 56789
Instantiates one or more workers
Call "MPI_Comm_split" on MPI_COMM_WORLD to create controller with color 56789
Registration packet
Ack packet
Request packet w/ node mesh data
Determine force deltas for each requested node
Response packet w/ node data
Save node data locally, prepare for interpolation
Once ready for simulation
loop [Update loop]
Request packet w/ node mesh and atom data
Determine force deltas for each requested node
Response packet w/ node data
loop [Simulation (n steps)]
Calls "post_force"
Interpolate forces from atom position data and precalculated nodes
Returns from "post_force"
Once simulation has ended
Calls destructor
Deregistration packet
Once all workers have deregistered
Final "MPI_Barrier"
controller
worker
LAMMPS

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



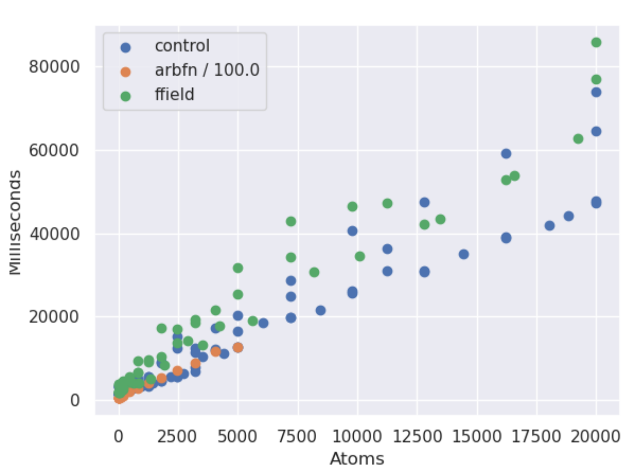


FIG. 4: Comparing `fix arbfn` to control and `fix arbfn/ffield`. While still approximately linear with respect to the number of atoms, it runs *much* slower than the latter two (which are about the same speed).

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.

VI. FUTURE WORK

FUTURE WORK GOES HERE

ACKNOWLEDGMENTS

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

[1] 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).

[2] The Open MPI Project, Open mpi v5.0.x, `https://docs.open-mpi.org/en/v5.0.x/#` (2003–2025).

[3] Sandia Corporation, Lammps documentation (4 feb 2025 version), `https://docs.lammps.org/Manual.html` (2003–2025).

[4] A. Kohlmeyer, M. Ippolito, and C. Cavazzoni, Qm/mm support library (2014).

[5] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, 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).

[6] L. Do and Z. Hurák, Feedback control in molecular dynamics simulations using lammps, in *2024 European Control Conference (ECC)* (IEEE, 2024) pp. 1065–1070.