ARMY RESEARCH LABORATORY

LAMMPS Implementation of Constant Energy Dissipative Particle Dynamics (DPD-E)

by James P. Larentzos, John K. Brennan, Joshua D. Moore, and William D. Mattson

ARL-TR-6863 **March 2014**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5069

ARL-TR-6863 **March 2014**

LAMMPS Implementation of Constant Energy Dissipative Particle Dynamics (DPD-E)

James P. Larentzos
Dynamics Research Corporation (DRC)

John K. Brennan, Joshua D. Moore, and William D. Mattson
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
March 2014	Final	10 January 2012–13 September 2013

4. TITLE AND SUBTITLE
LAMMPS Implementation of Constant Energy Dissipative Particle Dynamics (DPD-E)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
James P. Larentzos,* John K. Brennan, Joshua D. Moore, and William D. Mattson

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-WML-B
Aberdeen Proving Ground, MD 21005-5069

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-6863

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
*Dynamics Research Corporation (DRC), High Performance Technologies Group at the U.S. Army Research Laboratory, Aberdeen Proving Ground, MD 21005

14. ABSTRACT
A general framework is presented for implementing the constant-energy Dissipative Particle Dynamics (DPD-E) method into the highly scalable Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) simulation software to efficiently model systems under isoenergetic conditions. The current LAMMPS velocity-Verlet (VV) integration scheme for isothermal DPD simulations is extended to the isoenergetic case. A description is given of the implementation of the Shardlow-splitting algorithm (SSA) to enable longer time steps with comparable accuracy. In addition, a description of example benchmark problems is provided, along with a discussion about the tradeoffs between the DPD version of the VV and the SSA integration schemes in terms of performance and stability.

15. SUBJECT TERMS
dissipative particle dynamics, Shardlow splitting, parallelization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			John K. Brennan
Unclassified	Unclassified	Unclassified	UU	44	19b. TELEPHONE NUMBER (*Include area code*) (410) 306-0678

Contents

List of Figures

List of Tables

Acknowledgments

We acknowledge Martin Lísal (Institute of Chemical Process Fundamentals of the ASCR and J. E. Purkinje University, Ústí nad Labem, Czech Republic), Timothy Sirk (U.S. Army Research Laboratory), and Timothy Mattox (Dynamics Research Corporation) for useful discussions in parallelizing the Shardlow splitting algorithm. We also acknowledge the computational resources and Productivity Enhancement, Technology Transfer and Training (PETTT) software support from the High Performance Computing Modernization office. Joshua D. Moore acknowledges support in part by an appointment to the U.S. Army Research Laboratory Postdoctoral Fellowship Program administered by the Oak Ridge Associated Universities through a cooperative agreement with the U.S. Army Research Laboratory. John K. Brennan and Joshua Moore acknowledge support in part by the Office of Naval Research and the Department of Defense High Performance Computing Modernization Program Software Application Institute for Multiscale Reactive Modeling of Insensitive Munitions.

1. Introduction

Dissipative Particle Dynamics (DPD) is an attractive method for simulating soft condensed matter, including polymers, surfactants, and colloids at the mesoscale. DPD is a particle-based mesoscopic method that can be understood as a coarse-graining technique that correctly predicts the hydrodynamic nature of a fluid. The general idea behind DPD is that particles interact with each other through a set of stochastic differential equations with conservative, dissipative, and random forces.

While the original DPD formalism is isothermal (*1*, *2*), it has been extended to the isoenergetic (DPD-E) (*3*, *4*) case, which is of practical interest for simulating materials at nonisothermal conditions. One challenge requiring special consideration is the numerical integration of the stochastic equations-of-motion (EOMs). The most commonly applied numerical integration scheme for DPD applications originates from the velocity-Verlet (VV) integrator used for molecular dynamics applications and is extended to DPD by including the dissipative and random forces in the overall force expression (*5*). This approach is currently used in the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) software package (*6*) but is limited only to the constant-temperature DPD method. An alternative integration scheme using the Shardlow-splitting algorithm (*7*) separates the EOMs into stochastic and deterministic integration steps. It is readily extendable to all DPD variants and has been found to be the most effective integration scheme available to date, especially when considering DPD simulations under isoenergetic conditions to attain the length and time scales necessary to model complex systems.

In this work, we present a general framework for implementing the DPD-E method into the highly scalable LAMMPS simulation software to efficiently model systems under isoenergetic conditions. We extend the current LAMMPS VV integration scheme for isothermal DPD simulations to the isoenergetic case. In addition, we describe the implementation of the Shardlow-splitting algorithm (SSA) to enable longer time steps with comparable accuracy. Finally, a description of example benchmark problems is provided, along with a discussion about the tradeoffs between the DPD version of the VV and the SSA integration schemes in terms of performance and stability.

2. Methods

2.1 Constant Energy Dissipative Particle Dynamics (DPD-E)

In the DPD-E method (*3, 4*) particles are defined by a mass m_i, position $\mathbf{r}_i$, momentum $\mathbf{p}_i$, and particle internal energy u_i. The variation of $\mathrm{d}u_i$ is taken as the sum of the two contributions that correspond to the mechanical work done on the system $\mathrm{d}u_i^{mech}$ and the heat conduction between particles $\mathrm{d}u_i^{cond}$. For constant-energy and constant-volume conditions, the evolution of DPD particles in time t is governed by the following EOM:

$$
\begin{aligned}
\mathrm{d}\mathbf{r}_i &= \frac{\mathbf{p}_i}{m_i}\mathrm{d}t \\
\mathrm{d}\mathbf{p}_i &= \sum_{j\neq i}\mathbf{F}_{ij}^{C}\mathrm{d}t - \gamma_{ij}\omega^{D}\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}}\mathrm{d}t + \sigma_{ij}\omega^{R}\frac{\mathbf{r}_{ij}}{r_{ij}}\mathrm{d}W_{ij} \\
\mathrm{d}u_i^{mech} &= \frac{1}{2}\sum_{j\neq i}\gamma_{ij}\omega^{D}\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)^2\mathrm{d}t - \frac{d\sigma_{ij}^2}{2}\left(\frac{1}{m_i}+\frac{1}{m_j}\right)\left(\omega^{R}\right)^2\mathrm{d}t \qquad \left(i=1,\ldots,N\right) \\
&\quad - \sigma_{ij}\omega^{R}\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\mathrm{d}W_{ij} \\
\mathrm{d}u_i^{cond} &= \sum_{j\neq i}\kappa_{ij}\left(\frac{1}{\theta_i}-\frac{1}{\theta_j}\right)\omega^{Dq}\mathrm{d}t + \alpha_{ij}\omega^{Rq}\mathrm{d}W_{ij}^{q}
\end{aligned}
\tag{1}
$$

where $\mathbf{r}_{ij}=\mathbf{r}_i-\mathbf{r}_j$ is the separation vector between particle i and particle j, and $r_{ij}=\left|\mathbf{r}_{ij}\right|$. $\mathbf{v}_{ij}=\frac{\mathbf{p}_i}{m_i}-\frac{\mathbf{p}_j}{m_j}$ is the relative velocity between the particle pair. γ_{ij} and σ_{ij} are the friction coefficient and noise amplitude between particle i and particle j, respectively. $\mathrm{d}W_{ij}=\mathrm{d}W_{ji}$ are the increments of the Wiener processes associated with momentum variations. The weight functions $\omega^{D}(r)$ and $\omega^{R}(r)$ vanish for $r\geq r_c$, where r_c is the cutoff radius, and are typically chosen as

$$
\omega^{D}(r) = \left[\omega^{R}(r)\right]^2 = \begin{cases}\left(1-\frac{r}{r_c}\right)^2, & r<r_c \\ 0, & r\geq r_c\end{cases}
\tag{2}
$$

Similarly, the weight functions $\omega^{Dq}(r)$ and $\omega^{Rq}(r)$ are defined in an equivalent manner as equation 2. θ_i is the particle internal temperature and is related to u_i through a mesoparticle equation of state. κ_{ij} and α_{ij} are the mesoscopic thermal conductivity and the noise amplitude

between particle i and particle j, respectively, and $\mathrm{d}W_{ij}^{q} = -\mathrm{d}W_{ji}^{q}$ are the increments of the Wiener processes associated with thermal conduction. The conservative force $\mathbf{F}_{ij}^{C}$ is given as the negative derivative of a coarse-grain potential, u_{ij}^{CG}, and is expressed in LAMMPS as

$$F_{ij}^{C} = -\frac{du_{ij}^{CG}}{dr_{ij}} = A_{ij}\omega^{R} \tag{3}$$

where A_{ij} is the force constant. Bonet Avalos and Mackie (*3*) demonstrated that thermodynamic consistency requires the following fluctuation-dissipation relations to be satisfied

$$\begin{aligned} \sigma_{ij}^{2} &= 2\gamma_{ij}k_{\mathrm{B}}\Theta_{ij} \\ \omega^{D}(r) &= \left[\omega^{R}(r)\right]^{2} \\ \alpha_{ij}^{2} &= 2k_{\mathrm{B}}\kappa_{ij} \\ \omega^{Dq}(r) &= \left[\omega^{Rq}(r)\right]^{2} \end{aligned} \tag{4}$$

where the relevant temperature is $\Theta_{ij}^{-1} = \frac{1}{2}\left(\frac{1}{\theta_i} + \frac{1}{\theta_j}\right)$. In the following outline, the EOMs are formulated for the VV integration scheme under isoenergetic conditions.

1. For $i = 1,\ldots,N$
   a. $\mathbf{p}'_i \leftarrow \mathbf{p}_i$
   b. $\mathbf{p}_i \leftarrow \mathbf{p}_i + \frac{\Delta t}{2}\mathbf{F}_i$, where $\mathbf{F}_i = \sum_{j\neq i}\mathbf{F}_{ij}^{C} + \mathbf{F}_{ij}^{D} + \mathbf{F}_{ij}^{R}$
   c. $\mathbf{r}_i \leftarrow \mathbf{r}_i + \Delta t\,\frac{\mathbf{p}_i}{m_i}$
   d. $u_i^{cond} \leftarrow u_i^{cond} + \sum_{j\neq i}\kappa_{ij}\left(\frac{1}{\theta_i} - \frac{1}{\theta_j}\right)\omega^{Dq}\frac{\Delta t}{2} + \alpha_{ij}\omega^{Rq}\varsigma_{ij}^{q}\frac{\sqrt{\Delta t}}{2}$
   e. $u_i^{mech} \leftarrow u_i^{mech} - \frac{1}{2}\sum_{j\neq i}\gamma_{ij}\omega^{D}\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)^{2}\frac{\Delta t}{2} - \frac{\sigma_{ij}^{2}}{2}\left(\frac{1}{m_i} + \frac{1}{m_j}\right)\left(\omega^{R}\right)^{2}\frac{\Delta t}{2} - \sigma_{ij}\omega^{R}\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\varsigma_{ij}\frac{\sqrt{\Delta t}}{2}$
2. *Force Calculation:* $\{\mathbf{F}_i\}_{i=1}^{N}$, where $\mathbf{F}_i = \sum_{j\neq i}\mathbf{F}_{ij}^{C} + \mathbf{F}_{ij}^{D} + \mathbf{F}_{ij}^{R}$
3. For $i = 1,\ldots,N$
   a. $\mathbf{p}'_i \leftarrow \mathbf{p}_i$

b. $\mathbf{p}_i \leftarrow \mathbf{p}_i + \frac{\Delta t}{2}\mathbf{F}_i$, where $\mathbf{F}_i = \sum_{j\neq i} \mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R$

c. $\mathbf{r}_i \leftarrow \mathbf{r}_i + \Delta t \frac{\mathbf{p}_i}{m_i}$

d. $u_i^{cond} \leftarrow u_i^{cond} + \sum_{j\neq i} \kappa_{ij}\left(\frac{1}{\theta_i} - \frac{1}{\theta_j}\right)\omega^{Dq}\frac{\Delta t}{2} + \alpha_{ij}\omega^{Rq}\zeta_{ij}^q\frac{\sqrt{\Delta t}}{2}$

e. $u_i^{mech} \leftarrow u_i^{mech} - \frac{1}{2}\sum_{j\neq i}\gamma_{ij}\omega^D\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)^2\frac{\Delta t}{2} - \frac{\sigma_{ij}^2}{2}\left(\frac{1}{m_i}+\frac{1}{m_j}\right)\left(\omega^R\right)^2\frac{\Delta t}{2}$
$- \sigma_{ij}\omega^R\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\zeta_{ij}\frac{\sqrt{\Delta t}}{2}$

Here, the EOMs are formulated for the VV integration scheme using the Shardlow-splitting algorithm (VV-SSA) (*8*).

1. *Stochastic Integration*: For all $i-j$ pairs of particles

a. $\mathbf{p}'_i \leftarrow \mathbf{p}_i$, $\mathbf{p}'_j \leftarrow \mathbf{p}_j$

b. $\mathbf{p}_i \leftarrow \mathbf{p}_i - \frac{\Delta t}{2}\gamma_{ij}\omega^D\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} + \sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\sqrt{\Delta t}}{2}$

c. $\mathbf{p}_j \leftarrow \mathbf{p}_j + \frac{\Delta t}{2}\gamma_{ij}\omega^D\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} - \sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\sqrt{\Delta t}}{2}$

d. $\mathbf{v}_{ij} \leftarrow \frac{\mathbf{p}_i}{m_i} - \frac{\mathbf{p}_j}{m_j}$

e. $\mathbf{p}_i \leftarrow \mathbf{p}_i - \frac{\Delta t}{2}\frac{\gamma_{ij}\omega^D}{1+\frac{\mu_{ij}}{2}\gamma_{ij}\omega^D\Delta t}\left[\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} + \frac{\mu_{ij}}{2}\sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\sqrt{\Delta t}\right] + \sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\sqrt{\Delta t}}{2}$

f. $\mathbf{p}_j \leftarrow \mathbf{p}_j + \frac{\Delta t}{2}\frac{\gamma_{ij}\omega^D}{1+\frac{\mu_{ij}}{2}\gamma_{ij}\omega^D\Delta t}\left[\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} + \frac{\mu_{ij}}{2}\sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\sqrt{\Delta t}\right] - \sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\sqrt{\Delta t}}{2}$ where

$\mu_{ij} = \frac{1}{m_i} + \frac{1}{m_j}$

g. $u_i^{cond} \leftarrow u_i^{cond} + \kappa_{ij}\left(\frac{1}{\theta_i} - \frac{1}{\theta_j}\right)\omega^{Dq}\Delta t + \alpha_{ij}\omega^{Rq}\zeta_{ij}^q\sqrt{\Delta t}$

h. $u_j^{cond} \leftarrow u_j^{cond} - \kappa_{ij}\left(\frac{1}{\theta_i} - \frac{1}{\theta_j}\right)\omega^{Dq}\Delta t - \alpha_{ij}\omega^{Rq}\zeta_{ij}^q\sqrt{\Delta t}$

i. $u_i^{mech} \leftarrow u_i^{mech} - \frac{1}{2}\left[\frac{\mathbf{p}_i \cdot \mathbf{p}_i}{2m_i} + \frac{\mathbf{p}_j \cdot \mathbf{p}_j}{2m_j} - \frac{\mathbf{p}'_i \cdot \mathbf{p}'_i}{2m_i} - \frac{\mathbf{p}'_j \cdot \mathbf{p}'_j}{2m_j}\right]$

j. $u_j^{mech} \leftarrow u_j^{mech} - \frac{1}{2}\left[\frac{\mathbf{p}_i \cdot \mathbf{p}_i}{2m_i} + \frac{\mathbf{p}_j \cdot \mathbf{p}_j}{2m_j} - \frac{\mathbf{p}'_i \cdot \mathbf{p}'_i}{2m_i} - \frac{\mathbf{p}'_j \cdot \mathbf{p}'_j}{2m_j}\right]$

2. *Deterministic Integration #1*: For $i = 1,...,N$

   a. $\mathbf{p}_i \leftarrow \mathbf{p}_i + \frac{\Delta t}{2}\mathbf{F}_i^C$

   b. $\mathbf{r}_i \leftarrow \mathbf{r}_i + \Delta t \frac{\mathbf{p}_i}{m_i}$

3. *Conservative Force Calculation*: $\left\{\mathbf{F}_i^C\right\}_{i=1}^N$, where $\mathbf{F}_i^C = \sum_{j \neq i} \mathbf{F}_{ij}^C$

4. *Deterministic Integration #2*: For $i = 1,...,N$

   $\mathbf{p}_i \leftarrow \mathbf{p}_i + \frac{\Delta t}{2}\mathbf{F}_i^C$

2.2 Parallel Implementation of DPD-E Into the LAMMPS Framework

A detailed description of the theoretical foundations and parallelization of the DPD-E method using the VV and VV-SSA integration schemes can be found elsewhere (*8*, *9*). In this report, the relevant source code modifications that were required to implement the VV and VV-SSA integration schemes for DPD-E are presented. A summary of the code modifications is provided in appendices A–G. The source code is current with the 27 January 2014 version of LAMMPS and is available upon written request to the authors.

To implement DPD-E into LAMMPS, a new atom style was created to handle the DPD particle attributes required for DPD-E simulations. In addition, new pair styles were created to compute the DPD forces, and new fixes were created to integrate the DPD-E equations of motion through a VV and VV-SSA integration scheme. Compute functions were created to monitor the DPD particle attributes as a simulation progresses. Each new feature to LAMMPS is described in detail in the following sections.

2.2.1 Implementation of the DPD Atom Style

For all isoenergetic DPD calculations, a new atom style *dpd* is required to compute and communicate between processors the DPD particle attributes and per-atom arrays. DPD particles are specified in the LAMMPS input file via the atom_style command:

```
atom_style dpd
```

Selection of the *dpd* atom style requires the DPD internal temperature to be specified in the corresponding LAMMPS data file according to the following format:

- Column 1: particle id

- Column 2: particle type
- Column 3: Internal temperature (θ_i) of particle i
- Columns 4–6: Cartesian coordinates (x, y, z) of particle i

The implementation of the *dpd* atom style to LAMMPS requires a new AtomVecDPD class to be created in order to compute and communicate the DPD particle internal temperature (dpdTheta), equation of state flag (eos), conductive energy (uCond), mechanical energy (uMech), as well as the differences in the DPD particle's conductive energy (duCond) and mechanical energy (duMech) between two subsequent integration steps. The AtomVecDPD atom style class derives from the AtomVec parent class and is similar to and modeled after the existing AtomVecAtomic (`atom style atomic`) class.

Implementation into LAMMPS requires modification of the existing atom class (LAMMPS source files: *atom.h* and *atom.cpp*) and creation of the AtomVecDPD class (LAMMPS source files: *atom_vec_dpd.h* and *atom_vec_dpd.cpp*). The atom class code is modified to define the necessary pointers to the DPD attributes. The differences between the modified and the native LAMMPS codes are shown in appendix A. Next, the new AtomVecDPD child class was created by copying the existing AtomVecAtomic class (LAMMPS source files: *atom_vec_atomic.h* and *atom_vec_atomic.cpp*), then adjusting the data file format, the data structure sizes, and the communication buffers to handle the additional per-particle arrays. A summary of the differences between the AtomVecDPD and AtomVecAtomic classes is given in appendix B.

2.2.2 Implementation of the DPD and DPD/Atom Compute Commands

For all isoenergetic DPD calculations, it is useful to monitor all internal properties of the particles on both a system and per-particle basis. The ComputeDPD and ComputeDPDatom classes were created to monitor these properties over the course of the DPD-E simulations.

The ComputeDPD class (LAMMPS source files: *compute_dpd.h* and *compute_dpd.cpp*) computes the total particle internal conductive and mechanical energies by summing the per-particle energies. In addition, the particle internal temperature of the system is computed through a harmonic average of the per-particle internal temperatures, defined as:

$$\theta_{avg}^{-1} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\theta_i} \tag{5}$$

where *N* is the number of particles in the system. The ComputeDPD class is accessed through the LAMMPS input file via the compute command:

```
compute    1 all dpd
```

and returns a vector of size 5 that contains the following particle internal properties:

- The total conductive energy of the system
- The total mechanical energy of the system
- The sum of the conductive and mechanical energy of the system
- The harmonic averaged internal temperature of the system
- The number of particles in the group

The ComputeDpdAtom class (LAMMPS source files: *compute_dpd_atom.h* and *compute_dpd_atom.cpp*) accesses the per-particle internal energies and internal temperature. The compute is specified through the LAMMPS input file via the compute command

```
compute    1 all dpd/atom
```

and enables access to the following particle properties:

- The per-particle conductive energy
- The per-particle mechanical energy
- The per-particle internal temperature

2.2.3 Implementation of DPD-E With the VV Integration Scheme

Implementation of the isoenergetic DPD using the VV numerical integration scheme is similar to the existing isothermal DPD LAMMPS implementation. The major difference is the calculation of the particle internal energies within the *dpde* pair style compute command and the integration of the internal energies within the fix *dpde* command. The new pair style and fix commands are presented together since the classes interact with one another to perform all stages of the VV integration.

2.2.3.1 Implementation of the DPDE Pair Style

Implementation of constant energy DPD with the VV integration scheme requires a new pair style to be defined, which computes the conservative, dissipative, and random forces as well as the change in particle internal conductive energy and mechanical energy from one integration step to the next. The new PairDPDE class (LAMMPS source files: *pair_dpde.h* and *pair_dpde.cpp*) is specified through the LAMMPS input file via the pair_style command

```
pair_style dpde <kappa flag> <cutoff> <random number seed>
```

```
pair_coeff   i j A_DPD  sigma_ij  kappa <cutoff>
```

An energy-independent or energy-dependent (*10*) kappa model is specified in the pair_style command by setting the kappa_flag to 0 or 1, respectively. In the energy-independent model (kappa_flag = 0), κ_{ij} is explicitly given as a pair coefficient in the LAMMPS input file. In the energy-dependent model (kappa_flag = 1), κ_{ij} is given by the equation

$$\kappa_{ij} = \frac{\kappa_0}{k_B}\left(\frac{u_i+u_j}{2}\right)^2 \quad (6)$$

where the κ_0 pair coefficient is specified in the LAMMPS input file, u_i is the total internal energy of particle i, and k_B is Boltzmann's constant.

The PairDPDE pair style class derives from the Pair parent class and is very similar to and modeled after the existing PairDPD pair style class, except that it contains additional per-atom arrays that update the DPD particle internal temperature (dpdTheta), conductive energy (uCond), mechanical energy (uMech), heat capacity (cv), and density (rho), as well as the differences in the DPD particle's conductive energy (duCond) and mechanical energy (duMech) between two subsequent integration steps. The main differences between the native PairDPD and modified PairDPDE classes are summarized in appendix C.

2.2.3.2 Implementation of DPDE Fix Commands

Implementation of constant energy DPD with the VV integration scheme also requires a new fix class to be defined, which accounts for the integration of the conductive and mechanical energy and applying a mesoparticle equation of state. The new fix is defined as the FixDPDE class (LAMMPS source files: *fix_dpde.h* and *fix_dpde.cpp*) and is specified in the LAMMPS input file via the fix command

```
fix     1 all dpde
```

The FixDPDE is very similar to and modeled after the existing FixNVE class, except that it contains the integration of the conductive and mechanical energy. The main differences between the native FixNVE and modified FixDPDE classes are summarized in appendix D.

Once the updated conductive and mechanical energy is computed, the DPD particle internal temperatures are updated through the mesoparticle equation of state that relates the total particle internal particle energy to the internal temperature. The mesoparticle equation of state must be specified with a separate fix command. Currently, only one choice of the mesoparticle equation of state has been implemented. It relates the total internal energy to the internal temperature through the heat capacity according to the relation

$$u_i = C_{V,i}\theta_i \quad (7)$$

The new fix is defined as the FixEOScv class (LAMMPS source files: *fix_eos_cv.h* and *fix_eos_cv.cpp*) and is specified in the LAMMPS input file via the fix command:

```
fix     1 all eos/cv <cv>
```

where cv is the value of the heat capacity. The FixEOScv class is summarized in appendix E.

2.2.4 Implementation of DPD-E With the VV-SSA Integration Scheme

Implementation of the isoenergetic DPD using the VV-SSA integration scheme requires a splitting of the stochastic and deterministic EOMs. The conservative force is computed within the *dpde/conservative* pair style and is integrated deterministically through the VV integration scheme with the fix *dpde/shardlow* command. The random and dissipative forces are computed and integrated through the SSA within the fix *dpde/shardlow* prior to the deterministic integration of the conservative force. The new pair style and fix commands are presented together since the classes interact with one another to perform all stages of the VV-SSA integration. Additional implementation details of the SSA can be found elsewhere (*9*).

2.2.4.1 Implementation of the DPDE/Conservative Pair Style

Implementation of constant energy DPD using the VV-SSA integration scheme requires a new pair style that computes the conservative force to be defined. The new PairDPDEConservative class (LAMMPS source files: *pair_dpde_conservative.h* and *pair_dpde_conservative.cpp*) is specified in the LAMMPS input file via the pair_style command

```
pair_style dpde/conservative <kappa flag> <cutoff> <random number seed>
```

```
pair_coeff   i j A_DPD  sigma_ij  kappa <cutoff>
```

An energy-independent or energy-dependent (*10*) kappa model is specified in the pair_style command by setting the kappa_flag to 0 or 1, respectively. In the energy-independent model (kappa_flag = 0), κ_{ij} is explicitly given as a pair coefficient in the LAMMPS input file. In the energy-dependent model (kappa_flag = 1), κ_{ij} is related to the κ_0 pair coefficient as given in equation 6.

The PairDPDEConservative pair style class derives from the Pair parent class and is very similar to and modeled after the existing PairDPD pair style class. The main difference is that random and dissipative computations are removed, while the required data structures to compute the DPD particle attributes are included. The main differences between the PairDPD and PairDPDEConservative classes are summarized in section 2.2.4.2 and are explicitly shown in appendix F.

2.2.4.2 Implementation of DPDE/Shardlow Fix Command

Implementation of constant energy DPD using the VV-SSA integration also requires a new fix class to be defined, which accounts for the stochastic integration of the random and dissipative forces, the deterministic integration of the conservative force, and the integration of the conductive and mechanical energy for constant energy simulations. The new fix is defined as the FixDPDEShardlow class (LAMMPS source files: *pair_dpde_shardlow.h* and *pair_dpde_ shardlow.cpp*) and is specified in the LAMMPS input file via the fix command:

```
fix     1 all dpde/shardlow
```

The FixDPDEShardlow class is very similar to the existing FixNVE class, except that it contains an additional function to integrate the stochastic equations of motion, the conductive energy, and the mechanical energy. These integrations are handled via a function called stochastic_integrate within the FixDPDEShardlow class. The DPD particle internal temperatures are updated through the mesoparticle equation of state that relates the total particle internal particle energy to the internal temperature. The mesoparticle equation of state must be specified with a separate fix command. Currently, only one choice of the mesoparticle equation of state has been implemented and can be specified in the LAMMPS input file via the fix command

```
fix     1 all eos/cv <cv>
```

The main differences between the native FixNVE and modified FixDPDEShardlow classes are summarized in appendix G.

3. Example Input Files for VV and VV-SSA Integration

Example input files running a DPD-E simulation in LAMMPS with the VV and VV-SSA integration schemes are provided in figures 1 and 2, respectively.

Figure 1. Example input file for simulating a DPD fluid under isoenergetic conditions with the VV integration scheme.

```
# Input File for DPD fluid under isoenergetic conditions using the VV integration scheme
boundary p p p

units         metal  # ev, ps
atom_style      dpd
read_data       initial.conf.DPDfluid

communicate single vel yes
mass          1 125.9
pair_style      dpde 1 8.60 234324
pair_coeff      1 1 0.0752 0.0223 4.55E-05 8.60

neighbor        2.0 bin
neigh_modify    every 1 delay 0 check no once no

# Time in ps for metal units
timestep        0.001

compute         dpdU all dpd

variable        totEnergy equal pe+ke+c_dpdU[3]

thermo          1
thermo_style    custom step temp pe ke c_dpdU[1] c_dpdU[2] c_dpdU[3] v_totEnergy c_dpdU[4]
thermo_modify   format float %15.10E

fix           1 all dpde
fix           2 all eos/cv  0.00517041
run           100
```

Figure 2. Example input file for simulating a DPD fluid under isoenergetic conditions with the VV-SSA integration scheme.

```
# Input File for DPD fluid under isoenergetic conditions using the VV-SSA integration scheme
boundary p p p

units         metal  # ev, ps
atom_style      dpd
read_data       initial.conf.DPDfluid

communicate single vel yes
mass          1 125.9
pair_style      dpde/conservative 1 8.60 234324
pair_coeff      1 1 0.0752 0.0223 4.55E-05 8.60

neighbor        2.0 bin
neigh_modify    every 1 delay 0 check no once no

# Time in ps for metal units
timestep        0.001

compute         dpdU all dpd

variable        totEnergy equal pe+ke+c_dpdU[3]

thermo          1
thermo_style    custom step temp pe ke c_dpdU[1] c_dpdU[2] c_dpdU[3] v_totEnergy c_dpdU[4]
thermo_modify   format float %15.10E

fix           1 all dpde/shardlow
fix           2 all eos/cv  0.00517041
run           100
```

The example input files contain the commands to simulate a 10,125 particle system in a cubic box with volume 129.0 $Å^3$ at a temperature of 300 K under isoenergetic conditions with a time step of 0.001 ps. The eos/cv equation of state is specified along with an energy-dependent kappa model. The neighbor lists are reconstructed after every time step, and the thermodynamics are output every 0.001 ps. The total particle internal energies and internal temperature are included in the thermodynamics via the compute *dpd* command. The DPD-E parameters are summarized in table 1.

Table 1. Model and system parameters for DPD-E simulations of the DPD fluid model.

System Property	Real Units
ρ_{DPD}	4.72×10^{-3} Å^3
$T_{SP} = T_{init}$	300 K
$mass$	125.9 g mol^{-1}
A_{DPD}	7.52×10^{-2} eV Å^{-1}
r_{cut}	8.6 Å
σ	2.23×10^{-2} eV ps$^{1/2}$ Å^{-1}
κ_0	4.55×10^{-5} ps^{-1}
C_V	5.17×10^{-3} eV K^{-1}

4. Results

The example benchmarks described in section 5 are compared in terms of performance and accuracy in Lísal et al. (*8*), where DPD fluid calculations were performed under isoenergetic conditions with 10,125 particles at time steps ranging from 0.001 to 0.4 ps for a total of 1 ns of simulation time. Lísal et al. (*8*) shows that the VV-SSA integrator is considerably more stable than the VV integrator, where comparable accuracy is achieved with time steps that are *10–100 times longer than the traditional VV integrator*. However, the VV-SSA implementation is shown to be more computationally expensive than the serial VV implementation on a per time step basis and carries additional parallelization overhead. Analysis of the trade-off between computation performance and time-step size stability showed that the VV-SSA integration scheme can decrease the overall time-to-solution by a factor of 10–100 for a DPD-E simulation, thereby justifying practical and regular implementation of the approach.

5. Conclusions

In this report we have provided the user with the following information and resources:

- A detailed description of the implementation of the DPD-E method using the VV and VV-SSA integration schemes into LAMMPS.
- Documentation that highlights the major modifications to the source code.
- Example input scripts to run the software.
- A benchmark study to give an indication of the performance and accuracy of DPD-E with the VV and VV-SSA integration schemes.

6. References

1. Hoogerbrugge, P. J.; Koelman, J. M. V. A. Simulating Microscopic Hydrodynamic Phenomena With Dissipative Particle Dynamics. *Europhys. Lett*. **1992,** *19,* 155.
2. Koelman, J. M. V. A.; Hoogerbrugge, P. J. Dynamic Simulation of Hard Sphere Suspensions Under Steady Shear. *Europhys. Lett*. **1993,** *21,* 363.
3. Bonet Avalos, J.; Mackie, A. D. Dissipative Particle Dynamics With Energy Conservation. *Europhys. Lett*. **1997,** *40* (2), 141.
4. Español, P. Dissipative Particle Dynamics With Energy Conservation. *Europhys. Lett*. **1997,** *40* (6), 631.
5. Groot, R. D.; Warren, P. B. Dissipative Particle Dynamics: Bridging the Gap Between Atomistic and Mesoscopic Simulation. *J. Chem. Phys*. **1997,** *107,* 4423.
6. Plimpton, S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comp. Phys*. **1995,** *117,* 1–19.
7. Shardlow, T. Splitting for Dissipative Particle Dynamics. *SIAM J. Sci. Comput*. **2003,** *24,* 1267.
8. Lísal, M.; Brennan, J. K.; Avalos, J. B. Dissipative Particle Dynamics at Isothermal, Isobaric, Isoenergetic, and Isoenthalpic Conditions Using Shardlow-Like Splitting Algorithms. *J. Chem. Phys*. **2011,** *135,* 204105.
9. Larentzos, J. P.; Brennan, J. K.; Moore, J. D.; Lísal, M.; Mattson, W. D. Parallel Implementation of Isothermal and Isoenergetic Dissipative Particle Dynamics Using Shardlow-Like Splitting Algorithms. *Comput. Phys. Commun*., in press.
10. Hernando, R. Kinetic Theory of Dissipative Particle Dynamics Models. Ph.D Thesis, Universidad Nacional de Educación a Distancia, 2002.

Appendix A. Code Differences Between the Atom Class Files as Compared to the Native LAMMPS Code

This appendix appears in its original form, without editorial change.

```
diff --git a/src/atom.h b/src/atom.h
index cc044f1..1d122ec 100644
--- a/src/atom.h
+++ b/src/atom.h
@@ -50,6 +50,10 @@ class Atom : protected Pointers {
  int *type,*mask;
  imageint *image;
  double **x,**v,**f;
+  double *uCond, *uMech;
+  double *duCond, *duMech;
+  double *dpdTheta;
+  int *eos;

  tagint *molecule;
  int *molindex,*molatom;
```

```
diff --git a/src/atom.cpp b/src/atom.cpp
index 86ae09b..888aebe 100644
--- a/src/atom.cpp
+++ b/src/atom.cpp
@@ -72,6 +72,10 @@ Atom::Atom(LAMMPS *lmp) : Pointers(lmp)
  type = mask = NULL;
  image = NULL;
  x = v = f = NULL;
+  uCond = uMech = NULL;
+  duCond = duMech = NULL;
+  dpdTheta = NULL;
+  eos = NULL;

  molecule = NULL;
  molindex = molatom = NULL;
```

Appendix B. Summary of LAMMPS Code Differences Between the AtomVecDPD Class as Compared to the Native AtomVecAtomic Class

This appendix appears in its original form, without editorial change.

- AtomVecDPD Constructor:
    - Initialized uCond, uMech, duCond, duMech and eos pointers to NULL
    - Updated comm_x_only and comm._f_only to 1 to allow for forward communication of rho, cv and dpdTheta
    - Updated data buffer size definitions for the forward and border communications (size_forward=9, size_border=12) to account for cv, dpdTheta, eos, uCond and uMech
    - Updated the number of columns in the data file (size_data_atom = 6). The data file contains the following six columns:
        - Column 1: particle id
        - Column 2: particle type
        - Column 3: Internal temperature (θ_i) of particle i
        - Columns 4-6: Cartesian coordinates (x, y, z) of particle i
    - Updated the column the x data is located (xcol_data = 4)
    - Set the rho_flag to be on (atom->rho_flag = 1)
- void AtomVecDPD::grow(int n)
    - Added memory allocation for rho, drho, cv, dpdTheta, eos, uCond, uMech, duCond and duMech
- void AtomVecDPD::grow_reset()
    - Added pointers for rho, drho, cv, dpdTheta, eos, uCond, uMech, duCond and duMech
- void AtomVecDPD::copy(int i, int j, int delflag)
    - Copies particle i rho, drho, cv, dpdTheta, eos, uCond, uMech to particle j
- int AtomVecDPD::pack_comm(int n, int *list, double *buf, int pbc_flag, int *pbc)
    - Packs rho, cv, dpdTheta, eos, uCond and uMech in the forward communication buffer
- int AtomVecDPD::pack_comm_vel(int n, int *list, double *buf, int pbc_flag, int *pbc)
    - Packs rho, cv, dpdTheta, eos, uCond and uMech in the forward communication buffer, along with velocities
- void AtomVecDPD::unpack_comm(int n, int first, double *buf)
    - Unpacks rho, cv, dpdTheta, eos, uCond and uMech from forward communication buffer
- void AtomVecDPD::unpack_comm_vel(int n, int first, double *buf)
    - Unpacks rho, cv, dpdTheta, eos, uCond and uMech from forward communication buffer, along with velocities
- int AtomVecDPD::pack_border(int n, int *list, double *buf, int pbc_flag, int *pbc)
    - Packs rho, cv, dpdTheta, eos, uCond and uMech in the border communication buffer.
- int AtomVecDPD::pack_border_vel(int n, int *list, double *buf, int pbc_flag, int *pbc)
    - Packs rho, cv, dpdTheta, eos, uCond and uMech in the border communication buffer, along with velocities
- void AtomVecDPD::pack_comm_hybrid(int n, int *list, double *buf)
    - Packs rho, cv, dpdTheta, eos, uCond and uMech for the hybrid atom style communication buffer
- void AtomVecDPD::pack_border_hybrid(int n, int *list, double *buf)
    - Packs rho, cv, dpdTheta, eos, uCond and uMech in the hybrid atom style border communication buffer
- void AtomVecDPD::unpack_border(int n, int first, double *buf)
    - Unpacks rho, cv, dpdTheta, eos, uCond and uMech from the border communication buffer
- void AtomVecDPD::unpack_border_vel(int n, int first, double *buf)
    - Unpacks rho, cv, dpdTheta, eos, uCond and uMech from the border communication buffer, along with velocities
- void AtomVecDPD::unpack_comm_hybrid(int n, int first, double *buf)
    - Unpacks rho, cv, dpdTheta, eos, uCond and uMech from the hybrid atom style communication buffer
- void AtomVecDPD::unpack_border_hybrid(int n, int first, double *buf)
    - Unpacks rho, cv, dpdTheta, eos, uCond and uMech from the hybrid atom style border communication buffer
- int AtomVecDPD::pack_exchange(int i, double *buf)
    - Packs rho, cv, dpdTheta, eos, uCond and uMech in the exchange communication buffer.
- int AtomVecDPD::unpack_exchange(double *buf)

- Unpacks rho, cv, dpdTheta, eos, uCond and uMech from exchange communication buffer.

- **int AtomVecDPD::size_restart()**
  - Adjusted the size of the restart data to include rho, cv, dpdTheta, eos, uCond and uMech
    - Changed from 11 to 17

- **int AtomVecDPD::pack_restart(int i, double *buf)**
  - Packs rho, cv, dpdTheta, eos, uCond and uMech in the restart buffer

- **int AtomVecDPD::unpack_restart(double *buf)**
  - Unpacks rho, cv, dpdTheta, eos, uCond and uMech from the restart buffer

- **void AtomVecDPD::create_atom(int itype, double *coord)**
  - Creates a new particle and sets default rho, drho, cv, dpdTheta, eos, uCond, uMech, duCond and duMech values

- **void AtomVecDPD::data_atom(double *coord, tagint imagetmp, char **values)**
  - Read internal temperature (dpdTheta) from data file; initialize rho, cv, uCond[i] and uMech[i] to zero and eos to -1

- **int AtomVecDPD::data_atom_hybrid(int nlocal, char **values)**
  - Read internal temperature (dpdTheta) from hydrid atom style, data file

- **void AtomVecDPD::pack_data(double **buf)**
  - Packs internal temperature (dpdTheta) into the data file buffer

- **int AtomVecDPD::pack_data_hybrid(int i, double *buf)**
  - Packs internal temperature (dpdTheta) into the hydrid atom style, data file buffer

- **void AtomVecDPD::write_data(FILE *fp, int n, double **buf)**
  - Writes internal temperature (dpdTheta) to the data file

- **int AtomVecDPD::write_data_hybrid(FILE *fp, double *buf)**
  - Writes internal temperature (dpdTheta) to the hybrid atom info to data file

- **bigint AtomVecDPD::memory_usage()**
  - Added memory checks for rho, drho, cv, dpdTheta, eos, uCond, uMech, duCond and duMech

INTENTIONALLY LEFT BLANK.

Appendix C. Summary of LAMMPS Code Differences Between the PairDPDE Class as Compared to the Native PairDPD Class

This appendix appears in its original form, without editorial change.

- PairDPDE constructor: PairDPDE::PairDPDE(LAMMPS *lmp) : Pair(lmp)
  - Set the size of the pair forward communication to 3 ($duCond_i$, $duMech_i$, $dpdTheta_i$)
  - Set the size of the pair reverse communication to 2 ($duCond_i$, $duMech_i$)
- PairDPDE destructor: PairDPDE::~PairDPDE()
  - Deallocate kappa array; remove gamma deallocation
- void PairDPDE::compute(int eflag, int vflag)
  - Initialize duCond and duMech by setting the arrays to 0.0 followed by a forward communication
  - Define w^R and w^D: $\omega^D(r) = [\omega^R(r)]^2 = \begin{cases} \left(1 - \frac{r}{r_c}\right)^2, & r < r_c \\ 0, & r \geq r_c \end{cases}$

    where r_c is the cutoff radius for the pair of interacting particles.

    NOTE: In PairDPD class, wd definition differs
  - Compute theta_ij: $\theta_{ij}^{-1} = \frac{1}{2}\left(\frac{1}{\theta_i} + \frac{1}{\theta_j}\right)$
  - Compute gamma_ij: $\gamma_{ij} = \frac{\sigma_{ij}^2}{2k_B\theta_{ij}}$
  - Compute the conservative, dissipative and random forces

$$\mathbf{F}_{ij} = \mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R$$

$$\boldsymbol{F}_{ij}^C = A_{DPD}\omega^R(r)\frac{\boldsymbol{r_{ij}}}{r_{ij}}$$

$$\mathbf{F}_{ij}^D = -\gamma_{ij}\omega^D\left(r_{ij}\right)\left(\frac{\mathbf{r}_{ij}}{r_{ij}} \cdot \mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}}$$

$$\boldsymbol{F}_{ij}^R = \sigma_{ij}\omega^R(r)\varsigma_{ij}\frac{\boldsymbol{r_{ij}}}{r_{ij}}\frac{1}{\sqrt{\Delta t}}$$

    where ς_{ij} is a random number sampling a Gaussian distribution and Δt is the timestep used in the simulation
  - Compute kappa_ij: κ_{ij} or $\kappa_{ij} = \frac{\kappa_0}{k_B}\left(\frac{u_i+u_j}{2}\right)^2$
  - Compute alpha_ij: $\alpha_{ij} = \sqrt{2k_B\kappa_{ij}}$
  - Compute mu_ij: $\mu_{ij} = \left(\frac{1}{m_i} - \frac{1}{m_j}\right)$
  - Compute $duMech_i$ and $duMech_j$

$$du_i^{mech} = \frac{1}{2}\left[\gamma_{ij}\omega^D\left(\frac{\boldsymbol{r_{ij}}}{r_{ij}} \cdot \boldsymbol{v_{ij}}\right)^2 - \frac{\sigma_{ij}^2}{2}\mu_{ij}\omega^D - \sigma_{ij}\omega^R\left(\frac{\boldsymbol{r_{ij}}}{r_{ij}} \cdot \boldsymbol{v_{ij}}\right)\varsigma_{ij}\frac{2}{\sqrt{\Delta t}}\right]$$

$$du_j^{mec\,h} = du_i^{mech}$$

  - Compute $duCond_i$ and $duCond_j$

$$du_i^{cond} = \kappa_{ij}\left(\frac{1}{\theta_i} - \frac{1}{\theta_j}\right)\omega^{Dq} + \alpha_{ij}\omega^{Rq}\varsigma_{ij}^q\frac{1}{\sqrt{\Delta t}}$$

$$du_j^{cond} = -du_i^{cond}$$

  - Compute the interaction energy between the particle pair
    - evdwl = 0.5*a0[itype][jtype]*cut[itype][jtype] * wd;
  - Reverse communication of duCond and duMech
- void PairDPDE::allocate()
  - Allocate memory for kappa array; Remove gamma allocation
- void PairDPDE::settings(int narg, char **arg)

- Read 3 arguments associated with pair style (kappa_flag, cut_global and seed); Remove temperature
- Ensure kappa_flag is set to either 0 or 1

- **void PairDPDE::coeff(int narg, char **arg)**
  - Reads in 3-4 three pair coefficients
    - A_{DPD}
    - Sigma (σ_{ij}); removed gamma
    - Kappa_ij (κ_{ij}) or Kappa0 (κ_0)
    - cutoff (optional)
- **void PairDPDE::init_style()**
  - Ensures that the Newton pair is set to "on" for DPD calculations
- **double PairDPDE::init_one(int i, int j)**
  - Initializes the pair coefficients for all particle pairs; Add kappa, sigma and remove gamma
- **void PairDPDE::write_restart(FILE *fp)**
  - Updates the pair coefficient information to the restart file; Add kappa, sigma and remove gamma
- **void PairDPDE::read_restart(FILE *fp)**
  - Read pair coefficients from restart file and broadcast across processors; Add kappa, sigma and remove gamma
- **void PairDPDE::write_restart_settings(FILE *fp)**
  - Removed the temperature variable
- **void PairDPDE::read_restart_settings(FILE *fp)**
  - Removed temperature variable
- **void PairDPD::write_data(FILE *fp)**
  - Removed gamma; Added kappa and sigma to list of variable to write to the data file
- **void PairDPD::write_data_all(FILE *fp)**
  - Removed gamma; Added kappa and sigma to list of variable to write to the data file
- **double PairDPDE::single(int i, int j, int itype, int jtype, double rsq, double factor_coul, double factor_dpd, double &fforce)**
  - Updated definition of wr and wd
- **int PairDPDE::pack_comm(int n, int *list, double *buf, int pbc_flag, int *pbc)**
  - Packs the buffer for forward communication (duCond, duMech and dpdTheta)
- **void PairDPDE::unpack_comm(int n, int first, double *buf)**
  - Unpacks the buffer for forward communication (duCond, duMech and dpdTheta)
- **int PairDPDE::pack_reverse_comm(int n, int first, double *buf)**
  - Packs the buffer for reverse communication (duCond and duMech)
- **void PairDPDE::unpack_reverse_comm(int n, int *list, double *buf)**
  - Unpacks the buffer for reverse communication (duCond and duMech)

INTENTIONALLY LEFT BLANK.

Appendix D. Summary of LAMMPS Code Differences Between the FixDPDE Class as Compared to the Native FixNVE Class

This appendix appears in its original form, without editorial change.

- FixDPDE constructor
    - Ensure 3 keywords are given in the fix dpde command
- int FixDPDE::setmask()
    - Removed all reference to RESPA
- void FixDPDE::init()
    - Ensure that an equation of state is specified in another fix; otherwise, print error message and stop
    - Remove all references to RESPA
- void FixDPDE::initial_integrate(int vflag)
    - Integrate uCond and uMech by the relations:

$$uCond_i = uCond_i + \frac{1}{2} * dt * duCond_i$$

$$uMech_i = uMech_i + \frac{1}{2} * dt * duMech_i$$

- void FixDPDE::final_integrate()
    - Integrated uCond and uMech by the relations:

$$uCond_i = uCond_i + \frac{1}{2} * dt * duCond_i$$

$$uMech_i = uMech_i + \frac{1}{2} * dt * duMech_i$$

Appendix E. Summary of the FixEOScv Class

This appendix appears in its original form, without editorial change.

- **FixEOScv constructor**
  - Ensure 4 keywords are specified in the fix eos/cv command
  - Read the heat capacity, cv
- **int FixEOScv::setmask()**
  - Set the mask to POST_INTEGRATE and END_OF_STEP
- **void FixEOScv::init()**
  - Initialize the cv array
  - Initialize the eos array
  - Initialize uCond and uMech to be ½ * cv[i] * dpdTheta[i]
- **void FixEOScv::post_integrate()**
  - Apply the EOS to compute the DPD particle internal temperature. The eos/cv defines the following relation:

    $$\theta_i = (uCond_i + uMech_i) / C_{v,i}$$

    where θ_i is the internal particle temperature (dpdTheta).
- **void FixEOScv::end_of_step()**
  - Apply the EOS to compute the DPD particle internal temperature. The eos/cv defines the following relation:

    $$\theta_i = (uCond_i + uMech_i) / C_{v,i}$$

    where θ_i is the internal particle temperature (dpdTheta).

Appendix F. Summary of LAMMPS Code Differences Between the PairDPDE Conservative Class as Compared to the Native PairDPD Class

This appendix appears in its original form, without editorial change.

- PairDPDEConservative destructor: PairDPDE::~PairDPDE()
  - Deallocate kappa array; remove gamma deallocation
- void PairDPDEConservative::compute(int eflag, int vflag)
  - Removed dissipative and random force calculations
  - Define w^R and w^D: $\omega^D(r) = [\omega^R(r)]^2 = \begin{cases} \left(1 - \frac{r}{r_c}\right)^2, & r < r_c \\ 0, & r \geq r_c \end{cases}$

    where r_c is the cutoff radius for the pair of interacting particles.

    NOTE: In PairDPD class, wd definition differs
  - Compute the conservative force: $F_{ij}^C = A_{DPD}\omega^R(r)$
  - Compute the interaction energy between the particle pair
    - evdwl = 0.5*a0[itype][jtype]*cut[itype][jtype] * wd;
- void PairDPDEConservative::allocate()
  - Allocate memory for kappa array; Remove gamma allocation
- void PairDPDEConservative::settings(int narg, char **arg)
  - Read 3 pair style arguments (kappa_flag, cut_global and seed); Remove the temperature argument
  - Ensure kappa_flag is set to either 0 or 1
- void PairDPDEConservative::coeff(int narg, char **arg)
  - Reads in 3-4 three pair coefficients
    - A_{DPD}
    - Sigma (σ_{ij}); removed gamma
    - Kappa_ij (κ_{ij}) or Kappa0 (κ_0)
    - cutoff (optional)
- void PairDPDEConservative::init_style()
  - Ensures that ghost velocities are stored and Newton pair is set to “on” for DPD calculations
- double PairDPDEConservative::init_one(int i, int j)
  - Initializes the pair coefficients for all particle pairs; Add kappa, sigma and remove gamma
- void PairDPDEConservative::write_restart(FILE *fp)
  - Updates the pair coefficient information to the restart file; Add kappa, sigma and remove gamma
- void PairDPDEConservative::read_restart(FILE *fp)
  - Reads pair coefficients from restart file and broadcast across processors; Add kappa, sigma and remove gamma
- void PairDPDEConservative::write_restart_settings(FILE *fp)
  - Remove the temperature variable
- void PairDPDEConservative::read_restart_settings(FILE *fp)
  - Remove temperature variable
- void PairDPDEConservative::write_data(FILE *fp)
  - Removed gamma; Added kappa and sigma to list of variable to write to the data file
- void PairDPDEConservative::write_data_all(FILE *fp)
  - Removed gamma; Added kappa and sigma to list of variable to write to the data file
- double PairDPDEConservative::single(int i, int j, int itype, int jtype, double rsq, double factor_coul, double factor_dpd, double &fforce)
  - changed definition of wd and wr to be consistent with manuscript

Appendix G. Summary of LAMMPS Code Differences Between the FixDPDEShardlow Class as Compared to the Native FixNVE Class

This appendix appears in its original form, without editorial change.

- FixDPDEShardlow Constructor
  - Ensure 3 keywords are given in the fix dpde/shardlow command
  - Set the size of the fix forward communication to 10 (dvSSA[i][0-2], v[i][0-2], duCond[i], duMech[i], uCond[i], uMech[i])
  - Set the size of the fix reverse communication to 5 (dvSSA[i][0-2],duCond[i], duMech[i])
- int FixDPDEShardlow::setmask()
  - Remove all references to RESPA
- void FixDPDEShardlow::init()
  - Ensure that an equation of state is specified in another fix; otherwise, print error message and stop
  - Set the pairDPDE pointer to access the pairDPDEConservative class data structures
- void FixDPDEShardlow::initial_integrate(int vflag)
  - Added a shardlow_integrate() function prior to the velocity integration
  - Integrate the velocity and position at a half time step (Same as FixNVE)
- void FixDPDEShardlow::final_integrate()
  - Integrate the velocities at a half time step (Same as FixNVE)
- void FixDPDEShardlow::shardlow_integrate(): Member function to perform the stochastic integration of velocities
  - Compute the length of the bounding box dimensions (bbx, bby, bbz)
    - Ensure all bounding box dimensions are larger than interaction cutoff radius; Print error if not satisfied
  - Allocate memory for velocity (dvSSA[j][0-2])
  - Initialize (dvSSA[j][0-2], duCond[j], duMech[j]) arrays and forward communicate to ensure all processors are initialized
  - Allocate memory for the particle pair active interaction regions (jbin)
    - Count the number of pairs (count), then allocate active interaction region bins, jbin[count]
    - Assign each pair to an active interaction region depending on the coordinates of the pair. One atom will be in central box; assign the AIR based on the 2nd atom ,which may be in the central box or a neighboring box
  - Loop over the eight active interaction regions (Stages)
    - If the pair lies in the current active interaction region, then …
      - Compute the pair separation distance, r_{ij}, and check if it is less than the interaction cutoff
        - Store the current velocities (vx0i, vy0i, vz0i, vx0j, vy0j, vz0j)
        - Compute the velocity difference between i and j: $\mathbf{v}_{ij} \leftarrow \frac{\mathbf{p}_i}{m_i} - \frac{\mathbf{p}_j}{m_j}$ = v_i - v_j
        - Compute the dot product, dot = $r_{ij} \cdot v_{ij}$
        - Compute w^R and w^D: $\omega^D(r) = [\omega^R(r)]^2 = \begin{cases} \left(1 - \frac{r}{r_c}\right)^2, & r < r_c \\ 0, & r \geq r_c \end{cases}$ where r_c is the cutoff radius for the pair of interacting particles
        - Apply EOS to compute the current internal temperature.
        - Compute theta_ij: $\theta_{ij}^{-1} = \frac{1}{2}\left(\frac{1}{\theta_i} + \frac{1}{\theta_j}\right)$
        - Compute gamma_ij: $\gamma_{ij} = \frac{\sigma_{ij}^2}{2k_B\theta_{ij}}$
        - Obtain a random number ζ_{ij} from a Gaussian distribution
        - Compute the momentum (velocity) update

$$\mathbf{p}_i \leftarrow \mathbf{p}_i - \frac{\Delta t}{2}\gamma_{ij}\omega^D\left(\frac{\mathbf{r}_{ij}}{r_{ij}} \cdot \mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} + \sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\sqrt{\Delta t}}{2}$$

$$\mathbf{p}_j \leftarrow \mathbf{p}_j + \frac{\Delta t}{2}\gamma_{ij}\omega^D\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} - \sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\sqrt{\Delta t}}{2}$$

- Re-Compute the velocity difference between i and j: $\mathbf{v}_{ij} \leftarrow \frac{\mathbf{p}_i}{m_i} - \frac{\mathbf{p}_j}{m_j}$ = v_i - v_j
- Re-compute the dot product: dot = $r_{ij} \cdot v_{ij}$
- Compute mu_ij: $\mu_{ij} = \left(\frac{1}{m_i} - \frac{1}{m_j}\right)$
- Compute the momentum (velocity) update

$$\mathbf{p}_i \leftarrow \mathbf{p}_i - \frac{\Delta t}{2}\frac{\gamma_{ij}\omega^D}{1+\frac{\mu_{ij}}{2}\gamma_{ij}\omega^D\Delta t}\left[\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} + \frac{\mu_{ij}}{2}\sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\sqrt{\Delta t}\right] + \sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\sqrt{\Delta t}}{2}$$

$$\mathbf{p}_j \leftarrow \mathbf{p}_j + \frac{\Delta t}{2}\frac{\gamma_{ij}\omega^D}{1+\frac{\mu_{ij}}{2}\gamma_{ij}\omega^D\Delta t}\left[\left(\frac{\mathbf{r}_{ij}}{r_{ij}}\cdot\mathbf{v}_{ij}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} + \frac{\mu_{ij}}{2}\sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\sqrt{\Delta t}\right] - \sigma_{ij}\omega^R\zeta_{ij}\frac{\mathbf{r}_{ij}}{r_{ij}}\frac{\sqrt{\Delta t}}{2}$$

- Compute a new random number, ζ_{ij}^q
- Compute kappa_ij: $\kappa_{ij} = \frac{\kappa_0}{k_B}\left(\frac{u_i+u_j}{2}\right)^2$ (for T-dependent model)
- Compute alpha_ij: $\alpha_{ij} = \sqrt{2k_B\kappa_{ij}}$
- Compute the conductive energy update

$$u_i^{cond} \leftarrow u_i^{cond} + \kappa_{ij}\left(\frac{1}{\theta_i} - \frac{1}{\theta_j}\right)\omega^{Dq}\Delta t + \alpha_{ij}\omega^{Rq}\zeta_{ij}^q\sqrt{\Delta t}$$

$$u_j^{cond} \leftarrow u_j^{cond} - \kappa_{ij}\left(\frac{1}{\theta_i} - \frac{1}{\theta_j}\right)\omega^{Dq}\Delta t - \alpha_{ij}\omega^{Rq}\zeta_{ij}^q\sqrt{\Delta t}$$

- Compute the mechanical energy update

$$u_i^{mech} \leftarrow u_i^{mech} - \frac{1}{2}\left[\frac{\mathbf{p}_i\cdot\mathbf{p}_i}{2m_i} + \frac{\mathbf{p}_j\cdot\mathbf{p}_j}{2m_j} - \frac{\mathbf{p'}_i\cdot\mathbf{p'}_i}{2m_i} - \frac{\mathbf{p'}_j\cdot\mathbf{p'}_j}{2m_j}\right]$$

$$u_j^{mech} \leftarrow u_j^{mech} - \frac{1}{2}\left[\frac{\mathbf{p}_i\cdot\mathbf{p}_i}{2m_i} + \frac{\mathbf{p}_j\cdot\mathbf{p}_j}{2m_j} - \frac{\mathbf{p'}_i\cdot\mathbf{p'}_i}{2m_i} - \frac{\mathbf{p'}_j\cdot\mathbf{p'}_j}{2m_j}\right]$$

- Reverse communicate the velocity updates.
- Update the velocities and energies (v[j][0-2], uCond[j], uMech[j])
- Reinitialize the arrays to zero (dvSSA[j][0-2], duCond[j], duMech[j])
- Forward communicate the velocity/energy updates

- Delete the memory allocations for jbin and dvSSA

- int FixDPDEShardlow::sort_bin(double rx, double ry, double rz)
  - Compare the particle position to the reference points of the bounding box
    - Determine whether the point lies inside the bounding box, or outside the bounding box
    - Return a number to indicate the active interaction region
      - 0 = Inside bounding box
      - 1 = Top
      - 2 = Right

        - 3 = Top-right and Bottom-right
        - 4 = Back
        - 5 = Top-back and Bottom-back
        - 6 = Left-back and Right-back
        - 7 = Back Corners
- **int FixDPDEShardlow::pack_comm(int n, int *list, double *buf, int pbc_flag, int *pbc)**
    - Packs the buffer for forward communication (dvSSA, v, duCond, duMech, uCond, uMech)
- **void FixDPDEShardlow::unpack_comm(int n, int first, double *buf)**
    - Unpacks the buffer for forward communication (dvSSA, v, duCond, duMech, uCond, uMech)
- int FixDPDEShardlow::pack_reverse_comm(int n, int first, double *buf)
    - Packs the buffer for reverse communication (dvSSA, duCond, duMech)
- void FixDPDEShardlow::unpack_reverse_comm(int n, int *list, double *buf)
    - Unpacks the buffer for reverse communication (dvSSA, duCond, duMech)

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB IMAL HRA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	U OF MD-COLLEGE PARK DEPT MECH ENG P CHUNG
32 (PDF)	DIR USARL RDRL CIH C L MUNDAY RDRL CIH S J LARENTZOS RDRL WM P BAKER B FORCH J MCCAULEY RDRL WML B I BATYREV J BRENNAN S BUNTE E BYRD S IZVEKOV W MATTSON J MOORE J MORRIS R PESCE-RODRIGUEZ B RICE R SAUSA N TRIVEDI N WEINGARTEN RDRL WML C S AUBERT K MCNESBEY T PIEHLER RDRL WML D R BEYER RDRL WML H J NEWILL RDRL WMM J ZABINSKI

NO. OF COPIES	ORGANIZATION
	RDRL WMP C R BECKER J CLAYTON M GREENFIELD RDRL WMM G J ANDZELM T SIRK Y SLIOZBERG RDRL WMP G S KUKUCK B HOMAN

INTENTIONALLY LEFT BLANK.