Accelerating dissipative particle dynamics simulations for soft matter systems

Trung Dac Nguyen¹

National Center for Computational Sciences, Oak Ridge National Laboratory, Tennessee, 37831

Steven J. Plimpton

Sandia National Laboratories, Albuquerque, New Mexico, 87123

Abstract

Dissipative particle dynamics (DPD) is a coarse-grained particle-based simulation method that offers microscopic-scale insights into soft matter systems. We present an efficient implementation of a DPD model for graphical processing units (GPUs). As implemented in the LAMMPS molecular dynamics package, it can run effectively on current-generation supercomputers which often have hybrid nodes containing multi-core CPUs and (one or more) GPUs. Using efficient communication of information between the CPUs and GPUs, DPD interactions can be computed on the GPU while other portions of a full simulation model (boundary conditions, constraints, bonded interactions, diagnostic calculations, etc.) can be performed on the CPU. Our GPU-enhanced runs show a speedup of up to 9.5x versus many-core CPU simulations, and can run scalably across thousands of compute nodes. We briefly discuss how the new GPU implementation was validated against the CPU version for thermodynamics, diffusion, and hydrodynamic behavior. We also highlight large-scale models which the faster DPD implementation has enabled, for studies of monolayer self-assembly and thin-film instabilities.

Keywords: Dissipative particle dynamics; LAMMPS; GPU acceleration; hybrid CPU/GPU; hybrid MPI/GPU; high-performance computing.

1. Introduction

Dissipative particle dynamics (DPD) was first proposed by Hoogerbrugge and Koelman in 1992 as a particle-based method for studying hydrodynamic phenomena [1]. The statistical mechanics foundation of DPD was developed by Español and Warren in 1995 [2], followed by a rigorous analysis on the nature of its simulation parameters by Groot and Warren [3]. DPD has been widely used for studies of soft condensed matter systems, where the length scale of interest is greater than the atomistic scale but smaller than the characteristic structural dimensions, *e.g.* the scale of network connections or the spacings between assembled layers [3]. Owing to its inherent coarse-grained soft sphere model and its capability for capturing hydrodynamic interactions, DPD has also been used extensively in coupling particle-based simulation with continuum analyses, *i.e.* for multi-scale modeling. Examples include

the study of fluid flows in microchannels [4, 5, 6], thin film evolution [7, 8], self-assembly in surfactants and amphiphilic systems [9, 10, 11, 12, 13, 14, 15], structural and rheological properties of polymers [16, 17, 18] deformation of red blood cells [19] and membranes [20, 21]. We refer the readers to reference [22] for a thorough review of the applications of DPD simulations to soft matter systems.

DPD is a relatively “cheap” computational model for two reasons. Its interactions are pairwise within a relatively short cutoff distance. And the pairwise interactions have a “soft” core which allows use of a larger timestep than for other potentials such as Lennard-Jones with its $1/r^{12}$ core. As highlighted below, these features have made it an attractive model for GPU implementation by several groups; pairwise potentials are more straight-forward to parallelize for a GPU than many-body potentials [23, 24]. However, its soft core and the kinds of soft materials DPD is used to model, make it less amenable to the kinds of algorithmic acceleration techniques discussed elsewhere in this spe-

¹Correspondence should be sent to: ndactrung@gmail.com

cial issue. Methods such as parallel replica dynamics [25], temperature-accelerated dynamics [26], and kinetic Monte Carlo [27], enable dramatically longer timescale simulations by exploiting the rareness of discrete “events”, defined as the crossing of well-defined energy barriers. These methods have thus been almost exclusively applied to solid-state materials. Soft materials, modeled by soft potentials, under non-equilibrium conditions, do not typically have easily-identifiable barriers and events. In this paper we therefore address the theme of the special issue by relying less on algorithmic insight and more on exploiting new hardware (GPUs) to achieve accelerated simulation capabilities. We do this for a class of coarse-grained models which have wide application in industrial processing and more generally for simulation of liquids and soft matter at the mesoscale.

As is now well appreciated, the introduction of general-purpose graphics processing units (GPGPUs) to scientific computing (circa 2007) opened new opportunities for accelerating molecular dynamics (MD) codes. Exploiting the fine-grained parallelism offered by GPUs requires new MD methodologies. One trade-off is whether to offload only computationally intensive tasks to the GPU versus perform the entire simulation on the GPU. The former strategy (employed in this paper) is often pursued by legacy codes (*e.g.*, NAMD [28], GROMACS [29] and LAMMPS [30]), and the latter by newer codes (*e.g.*, HOOMD-Blue [31]).

Several successful efforts to accelerate DPD using GPUs have recently been described in the literature. To our knowledge, the first implementation of DPD to take full advantage of the single-instruction-multiple-threads (SIMT) environment of the GPU was the work of Phillips *et al.* [32] for HOOMD-Blue. In that work, a hash-function-based pseudo random number generator (PRNG) was used, previously developed by Steve Worley’s group, to create a micro-stream of random numbers (RNs) per thread per kernel call. Their approach addressed issues with (1) memory storage for matching random numbers for each pair, (2) broadcasting the states of the PRNG among threads, (3) expensive arithmetic operations required for generating RNs, and (4) the low quality of built-in linear congruential RNGs for large systems and long simulations. Specific details on the PRNG were subsequently described in the work of Afshar *et al.* [33]. These ideas have been extended for accelerating DPD in codes using multiple GPUs [34], using message-passing-interface (MPI) parallelism [33], and using hybrid MPI/CUDA parallelism [14]. Notably, the GPU-accelerated version of a generalized DPD model recently described by Tang

and Karniadakis [14] was shown to obtain a 10-30x speedup compared to a corresponding CPU-only version in LAMMPS [14]. This implementation was used in a large-scale mesoscopic study of amphiphilic systems in bulk and under soft confinement [15].

All of these DPD GPU implementations achieve their advertised speed-up when the entire MD simulation runs continuously on the GPU(s). By contrast, there is a package in LAMMPS (called GPU), which is designed to work on hybrid platforms, *i.e.* ones with compute nodes containing both CPUs and GPUs [35]. It allows pairwise potentials (and one many-body potential, Stillinger-Weber) as well as long-range electrostatic interactions to be computed on the GPU, concurrently with other bonded interactions (bonds, angles, dihedrals, etc) computed on the CPU.

More importantly, the GPU package still achieves good performance while communicating updated atom properties (positions, forces) back-and-forth between the CPU(s) and GPU each timestep. This is because the package allows multiple CPU cores to host a single GPU, which enables a greater degree of MPI-based parallelism across the multi-core CPUs of a large machine for these kinds of simulations. As a result, GPU utilization can be maximized and data transfers from one MPI task can be overlapped with GPU computations for other MPI tasks [23]. We note that complex simulations often involve significant computations that occur each timestep or every few timesteps, which at least in LAMMPS, have not been (and may never be) implemented for the GPU. Examples include specialized boundary conditions, application of constraint forces, simulation box deformations, calculation of diagnostic properties, etc. Using the GPU package, these computations still run efficiently on the CPU, using conventional MPI-based parallelization.

Many potentials have already been implemented for the GPU package in LAMMPS, including Coulombic and dipole-dipole interactions, Gay-Berne for ellipsoidal interactions, and the EAM, DLVO, and Stillinger-Weber potentials [36, 37, 23]. In this paper, we describe how the DPD model was implemented in the GPU package framework which required some unique issues to be addressed. The next sections give brief details of the implementation and illustrate its use in large-scale modeling of soft materials. The capabilities described here have been part of the general LAMMPS open-source release since April 2014.

2. Implementation

2.1. Equation of motion in DPD

In the standard DPD method, the equation of motion for particle i is given as [1, 2, 3]:

$$m_i \frac{d^2 \mathbf{x}_i}{dt^2} = \sum_{j \neq i} \mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R, \quad (1)$$

where m_i and $\mathbf{x}_i$ are the mass and position of particle i , respectively. $\mathbf{F}_{ij}^C$, $\mathbf{F}_{ij}^D$ and $\mathbf{F}_{ij}^R$ are the conservative force, drag force and random force, respectively, between particle i and its neighbor j within the cutoff distance r_c . These forces are defined as:

$$\mathbf{F}_{ij}^C = a_{ij} w_C(r_{ij}) \mathbf{e}_{ij} \quad (2)$$

$$\mathbf{F}_{ij}^D = -\gamma w_D(r_{ij}) (\mathbf{e}_{ij} \cdot \mathbf{v}_{ij}) \mathbf{e}_{ij} \quad (3)$$

$$\mathbf{F}_{ij}^R = \sigma w_R(r_{ij}) \xi_{ij} \Delta t^{-1/2} \mathbf{e}_{ij} \quad (4)$$

where $\mathbf{r}_{ij} = r_{ij} \mathbf{e}_{ij}$ is the distance vector between particles i and j , a_{ij} is the interaction strength, $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$, Δt is the integration time step, and ξ_{ij} is a random variable with Gaussian characteristics: $\langle \xi_{ij}(t) \rangle = 0$ and $\langle \xi_{ij}(t) \xi_{kl}(t') \rangle = (\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}) \delta(t - t')$ [3]. Importantly, the random variable is symmetric for every pair ij : $\xi_{ij}(t) = \xi_{ji}(t)$. $w_C(r_{ij})$, $w_D(r_{ij})$ and $w_R(r_{ij})$ are the weighing functions designed to vanish for $r \geq r_c$. In Eq. 4, the random force magnitude is inversely proportional to the time step, $\Delta t^{-1/2}$, to take into account the time discretization of the numerical integrator [3].

For mesoscopic coarse-grained models, for which DPD was developed, the conservative weighing function $w_C(r_{ij})$ is often chosen as, but not restricted to, a soft-core potential:

$$w_C(r_{ij}) = 1 - \frac{r_{ij}}{r_c}, \quad (5)$$

which allows particles to pass through each other. It is the soft-core model that allows for substantially bigger integration time steps compared to those used for atomic- and molecular-scale models. The random and drag weighing functions, and their corresponding amplitudes, are related to each other via the fluctuation-dissipation theorem [2, 3]:

$$w_R^2(r_{ij}) = w_D(r_{ij}) \quad (6)$$

$$\sigma^2 = 2\gamma k_B T \quad (7)$$

In the original formulation, ξ_{ij} is a normally distributed random variable with zero mean and unit vari-

ance, i.e., $N(0, 1)$. However, it was shown that a uniformly distributed variable, $U(-0.5, 0.5)$, with amplitude multiplied by $\sqrt{12}$, yields indistinguishable results [3], an effect we examine in Section 3.2.

The relationship between the drag force and random force (Eqs. 6 and 7) allows DPD to sample from the canonical ensemble [2]. Because the drag and random forces force between two interacting particles satisfy Newton's third law, they serve as a momentum-conserving thermostat. This is in contrast to Brownian Dynamics where the random forces applied to individual particles are uncorrelated. As a result, DPD is the current method of choice for simulating mesoscopic systems where hydrodynamic interactions are relevant.

2.2. Current DPD implementation in LAMMPS

We briefly discuss the spatial decomposition algorithm used in LAMMPS to illustrate how it effects our implementation of DPD. Interested readers can find more details in Ref. [30]. For a spatial decomposition, the simulation box is divided into non-overlapping sub-domains, one per MPI task or processor. The processor is responsible for updating the forces, positions and velocities of "owned" atoms within its sub-domain. This requires communication with processors owning nearby sub-domains to acquire "ghost" atoms, which affect forces on owned atoms. For DPD, the needed properties of ghost atoms are the atom IDs, types, positions, and velocities. The CPU implementation of DPD in LAMMPS uses the Marsaglia pseudo-random number generator [38] and a polar method to create a unique stream of Gaussian random numbers on each processor. The Marsaglia PRNG provides 900M independent streams based on an initialization seed, each of length 10^{30} ; it is thus a simple matter to assign a different stream to each processor. To achieve symmetry of the random force between each pair of interacting particles, a "half" neighbor list is used, so that the (I, J) neighbor pair is stored either by atom I or J . If the two particles are on different processors, only one processor computes the force on both particles and a second round of "ghost" atom communication is performed each timestep to communicate forces on ghost atoms back to the processor that owns each atom.

2.3. Accelerating DPD for GPUs

The GPU package (mentioned in the introduction) was introduced in LAMMPS in 2010. Details are described in a series of papers by Brown *et al.* [35, 36, 37, 23]. In the package framework, computation of pairwise forces and long-range Coulombics (via a particle-particle mesh (PPPM) solver) are performed on

the GPU. At every timestep atom data (positions, types, etc.) are transferred from the CPU memory to the GPU. Neighbor list builds are performed on the GPU, every few timesteps as needed. Pairwise forces are computed on the GPU, using the neighbor lists and the forces are transferred back to the CPU, so they can be used later in the timestep, for constraint-force calculation, time integration, etc. To increase efficiency on the GPU, more than one thread can be used to iterate over each atom's list of neighbors. This is a win when the number of neighbors is large or the potential is computationally intense, which is the case for DPD (relative to a simpler LJ potential). The data transfers between CPU and GPU, as well as the GPU computations are performed asynchronously and concurrently with respect to other computations on the CPU. A critical performance feature of the GPU package is that it allows multiple MPI processes, running on CPU cores, to share a single GPU. This provides additional concurrency and effectively reduces the overhead cost of transferring data between the CPU and GPU every timestep.

In contrast to the CPU, neighbor list building on the GPU creates “full” neighbor lists, where atom J is stored as a neighbor of atom I and vice versa. This means the force F_{ij} is computed twice (by the thread assigned to each atom), but it avoids a GPU memory collision since neither thread updates the force on the other atom. This savings more than pays for the cost of double computation; there is also no longer a need for the 2nd pass of ghost-atom force communication. However, for DPD, computing the force F_{ij} twice means the random forces will be different since independently generated random number streams are uncorrelated, as they are for the Marsaglia generator [38] when used as a PRNG.

This issue was solved for DPD, originally by Phillips *et al.* who used counter-based PRNGs for generating micro-streams of random numbers per thread per kernel call [32]. The idea is to produce encoded values from unique seeds through a series of integer arithmetic operations. The PRNG used by Phillips *et al.* [32], and later by Wang *et al.* [34], as described in more details by Afshar *et al.* in a recent work [33], utilizes the atom IDs for atoms I, J and the current time step as three integer seeds. Tang and Karniadakis proposed that bit-reversed atom tags be combined with the interleaved velocity mantissas via an exclusive-or bit operation to improve seed entropy [14]. The three seeds are encoded in two 32-bit variables, and subsequently advanced for a couple of iterations using classical RNGs or another counter-based RNG for short streams of random numbers (RNs). The output is thus a uniform RN unique to atoms I, J and the current timestep. The key point is that

the same RN can be computed independently by the two GPU threads which own atom I or J.

Anytime a new PRNG (or even non-parallel RNG) is devised, it is wise to carefully test its properties. Work by Worley *et al.* reported that the Saru PRNG passed rigorous tests (SmallCrush, Crush and BigCrush) defined by the TestU01 test suite [33]. This was also verified by Phillips *et al.* [32]. However, in the context of our DPD models, we performed additional tests of our GPU implementation versus the original CPU version, to see if any subtle PRNG effects became apparent at the scale of simulation results, when large numbers of RNs were used. These tests are described in Section 3.2.

For the DPD model implementation presented here, we extended the GPU package in LAMMPS to (1) transfer atom velocities and IDs between the CPU and GPU, (2) implement a light-weight version of the Saru PRNG algorithm that is compatible with both the CUDA nvcc and standard C compilers, (3) implement both the Saru linear congruential generator (LCG) and the eight-round Tiny Encrypted Algorithm (TEA8) for generating micro-streams of RNs per thread per kernel call, and (4) allow for choice of either uniformly distributed or Gaussian RNs. As noted previously, uniform RNs can be converted to Gaussian RNs using Marsaglia's polar method, which requires extra computation. We compare the two flavors of RNs in terms of performance and simulation results in Section 3.2.

3. Results

3.1. Performance

To compare performance of the GPU version of DPD with the original CPU version, we give strong and weak scaling results for a DPD fluid at number density $\rho = 6.0\sigma_0^{-3}$ and temperature $k_B T = 2.0$ (with $k_B = 1$). The interaction cutoff is $r_c = 1.5\sigma_0$, where $\sigma_0 = 1$ is the length unit. The repulsion strength is $a = 25.0$ and the friction coefficient $\gamma = 1.0$. For strong scaling, the number of particles (N) is fixed while the number of compute nodes is varied. Ideal strong scaling means that the number of time steps per second (TPS) increases linearly with the number of nodes. For weak scaling, the number of particles per node is fixed while the number of compute nodes is varied. Ideal weak scaling means that TPS is constant as the number of nodes increases. We compare two GPU versions of DPD (for mixed and double precision) against the original CPU version. Mixed precision mode means positions are stored and used in calculations as single precision (4-byte) values on the GPU; all other quantities (energy,

force and virial) are double precision (8-byte). Double precision mode means all quantities are stored and used on the GPU as 8-byte values, the same as the CPU version.

All runs in the following sections were performed on the Titan Cray XK7 supercomputer at Oak Ridge National Laboratory. It has 18,688 compute nodes with a Gemini interconnect. Each node has a 16-core CPU (2.2 GHz AMD Opteron 6274, 32 GB of memory (ECC DDR3 SDRAM), and a single NVIDIA Kepler K20X GPU. All of the per-node CPU results reported below were for runs using all 16 cores of each node, i.e. 16 MPI tasks/node. Thus we are comparing performance of each GPU to a 16-node CPU.

There are two parameters that influence the performance of the GPU DPD implementation, the number of GPU threads per particle, N_{threads} , used for computing forces, and the number of MPI processes sharing the GPU on each compute node, both of which we varied to achieve the fastest run times. We found a value of $N_{\text{threads}} = 8$ gave best performance for nearly all the performance data presented here. We also found that using 12 or 16 MPI processes per GPU was fastest when there were more than 64K DPD particles per node. For less than 10K particles per node, it was fastest to only use 1 or 2 MPI processes per node.

Figure 1 show strong scaling results for 1 to 1024 Titan compute nodes, using 3 different system sizes, $N = 256\text{K}$, 2M and 4M particles. Each system was equilibrated for 10000τ before the benchmark data was gathered. Each benchmark consisted of a 1000-timestep run, which is sufficiently long to give reliable simulation timings and to hide the overhead of loading dynamically-linked CUDA runtime libraries at the beginning of the run. The plots use million particle-steps (MPS) per second to quantify performance on the y-axis, i.e. if a million-particle system ran at a rate of 10 timesteps per second, its y-value would be 10 MPS per second.

On a single node, the GPU Mixed version of DPD is 9.5x faster than the CPU version for the $N = 256\text{K}$ particle system. The GPU Double Precision version is 1.6 times slower than Mixed, but still faster than the CPU version by 5.8x. Similar speed-ups result for the larger problem sizes at small node counts (left side of plots). As the number of compute nodes increases for each problem, the speedup decreases: e.g. for the smallest problem on 64 nodes, the GPU Mixed version is faster than the CPU version by only 2.6x. This is expected for strong scaling, since when the number of atoms per node becomes small, GPU computation is less efficient due to reduced occupancy of the device. The plots and

Figure 1: Strong scaling results for DPD systems at number density $\rho = 6.0\sigma_0^{-3}$ and $k_B T = 2.0$: (a) $N = 256\text{K}$ particles; (b) $N = 2\text{M}$ particles; (c) $N = 4\text{M}$ particles. Black solid lines represent ideal linear scaling slopes.

table show that parallel efficiency remains above 80% if the number of particles per node is greater than 32K, a relatively small number. We also observed that using uniformly distributed RNs was faster than Gaussian RNs by 7-15%, depending on the number of particles per node.

Table 1 shows performance for the largest problem size (6.9M atoms) for GPU Mixed vs CPU. It also includes a ‘‘p’’ column indicating how many MPI tasks per GPU (i.e. per node) gave optimal performance. For example, on 1024 nodes, 8 MPI tasks per node (8192 total MPI tasks) was the fastest, since there are few particles per GPU. The GPU version on 1024 nodes can

thus complete a typical self-assembly simulation, which requires approximately 100 million timesteps (1 million LJ τ) in 87 hours. We also benchmarked DPD systems at lower density, $\rho = 0.8\sigma_0^{-3}$, instead of $\rho = 6.0\sigma_0^{-3}$. This reduces the number of neighbors per particle, resulting in faster simulations. The ratio of GPU/CPU performance changed little for the low-density case, about a 15% drop as compared to the data in Figure 1 and Table 1.

No. of nodes	GPU	p	CPU	Speedup
16	123	16	12	10.2
32	162	16	30	5.4
64	312	12	67	4.6
128	601	8	136	4.4
256	880	16	220	4.0
512	1058	8	367	2.9
1024	2197	8	855	2.6

Table 1: Performance in MPS per second for $N = 6.9$ -million particles. p is the optimal number of MPI processors per node for the GPU runs.

Figure 2: Weak scaling results for DPD systems at number density $\rho = 6.0\sigma_0^{-3}$ and $k_B T = 2.0$: (a) $N = 32K$ particles per node; (b) $N = 256K$ particles per node. Black solid lines represent ideal linear scaling slopes.

For weak scaling, we benchmarked systems of size $N = 32K$ and $N = 256K$ particles/node, with results

shown in Figure 2. The GPU version exhibits similar behaviors for both system sizes, maintaining a high parallel efficiency up to 1000 nodes. For the smaller problems, the GPU Mixed version was 6x faster on 1 node and 4.8x faster on 1000 nodes. For the larger problem, the speed-up ranges from 9.5x on 1 node to 7.5x on 1000 nodes. The largest problem run was for 256M particles on 1000 nodes, which ran at a rate of 25 timesteps/second, or 900τ per hour, with the GPU version.

3.2. Validation

We performed various validation tests using full equilibrium and non-equilibrium simulations to ascertain that the GPU version of DPD gives results consistent with the original CPU version. We felt this was particularly important since a different random number generator (RNG) is used in the two versions and the degree of parallelism required for the parallel RN streams produced by the GPU RNG is much higher.

In the first test, we examine equilibrium thermodynamic quantities such as temperature, pressure and potential energy from a simulation in the canonical ensemble (constant temperature and volume). In the second, we monitor the mean-square displacement of liquid particles at different temperatures to evaluate self-diffusion coefficients. In the third, we model a non-equilibrium counter Poiseuille flow [4] in both the transient and steady-state regimes and compute velocity profiles, a methodology used to measure the shear viscosity of a fluid.

Figure 3 shows the time history of temperature, potential energy and pressure in a $N = 256K$ particle system, started from the same configuration for both the GPU and CPU versions. The results are statistically indistinguishable over a long time period of 1000τ , where $\tau = \sigma_0 \sqrt{m/k_B T}$ is the time unit, $\sigma_0 = 1$ is the length unit and m is the particle mass. We repeated the simulations at lower temperatures of 0.5 and 1.0 with similar results.

Figure 4 shows the mean-squared displacement (MSD) at different temperatures for a $N = 256K$ particle system. The slope of the MSD versus time is proportional to the self-diffusion coefficient D of the system, i.e. $D = \frac{1}{2d} \lim_{t \rightarrow \infty} \frac{\partial \langle \Delta r(t)^2 \rangle}{\partial t}$, where $d = 3$ dimensions. The GPU and CPU versions again give highly consistent results for individual data points and the slopes at early and late times.

The third test is a steady-state double-Poiseuille flow model used for viscosity measurements [4]. Fluid velocity profiles in both transient and steady states are

Figure 3: Thermodynamic quantities obtained from the GPU vs. CPU DPD implementation for a system consisting of 256K particles at number density $\rho = 6.0\sigma_0^{-3}$ and $k_B T = 2.0$: Instantaneous temperature (top), potential energy normalized by the number of particles (middle), and pressure (bottom).

shown in Figure 5. The simulation parameters are the repulsion strength $a = 25.0$, number density $\rho = 6.0\sigma_0^{-3}$, $k_B T = 0.1$ and the random force magnitude $\sigma = 3.0$. The simulation box dimensions are $14\sigma_0$ by $14\sigma_0$ by $28\sigma_0$ with periodic boundary conditions in all three directions. Particles with $z > 0$ are subject to an external force of $g_z = 0.055$, and those with $z < 0$ to $g_z = -0.055$ (in force units) along the z direction which induces the two-directional flow. The DPD interaction cutoff is $r_c = 1.0\sigma_0$ and the time step is $\Delta t = 0.01\tau$.

The transient velocity profiles show a parabolic form, which is qualitatively consistent with analytical models for a time-dependent Poiseuille flow [39]. Because the GPU and CPU versions give very similar transient profiles, we show only the GPU data in Figure 5(a) at various time points. It is noteworthy that such velocity profiles can only be obtained with $k_B T/\sigma_0$ smaller

Figure 4: Mean-squared displacement for DPD systems at number density $\rho = 6.0\sigma_0^{-3}$ and different temperatures. The number of particles is $N = 256K$.

Figure 5: Double-Poiseuille flows: (a) Transient and (b) steady state. The number of particles is $N = 16464$.

than, or comparable to, g_z ; otherwise, the profiles would be dominated by thermal noise. The steady-state velocity profiles produced by both uniformly distributed and Gaussian RNs with the GPU version also match remarkably well with the CPU version, as shown in Figure 5(b). The steady-state profiles can also be fit against an analytical model as shown in the figure, namely $v_x(z) = \rho g_z(zD + z^2)/2\eta$, where ρ is the fluid number density, η is fluid dynamic viscosity and $2D = L_z = 28\sigma_0$ is the width of the channel.

3.3. Soft matter simulations

We now highlight three large-scale mesoscopic systems that were simulated using the DPD GPU implementation, of the ORNL Titan machine, a prototypical hybrid supercomputer with nodes containing both CPUs and GPUs.

The first model is a binary mixture of surfactants with different chain lengths as shown in Figure 6(a). The A species are the short chains; B are the long chains. The head group particles of both species are constrained to move in a plane (representing a substrate), at a grafting density of 4 beads per σ_0^2 . Using a DPD model, Singh *et al.* observed that the head groups assemble into parallel stripes for a chain length ratio of 4:7 and a repulsion strength between like species of $a_{AA} = a_{BB} = 18.75$ and between unlike species of $a_{AB} = 33.75$ [11]. Our simulation results produce both the disordered state in (b) and the stripe state in (c) with $a_{AB} = 29.0$ and $a_{AB} = 18.75$, respectively. To facilitate the formation of the equilibrium structure at $a = 18.75$, we used an NPH ensemble to maintain pressure tensor components in both the x and y directions so that the grafting density of head groups is unchanged. The stripes are parallel to the x axis, indicating that they are not consequences of periodic boundary conditions. The simulated system consists of $N = 200K$ -particles, about 4 times bigger than that in the work of Singh *et al.* It took approximately 150 million timesteps, or $1.5 \times 10^6 \tau$, to perform a step-wise cooling from $a_{AA} = a_{BB} = 33.75$ to $a_{AA} = a_{BB} = 18.75$ to arrive at the stripe structure in Figure 6. At the rate of 430 timesteps/second on 9 Titan compute nodes with total 36 MPI tasks, the entire simulation took approximately 100 hours. This is a prototypical example that illustrates how a DPD model can rapidly relax a mesoscale system to equilibrium without the expense of accounting for detailed atomistic and molecular interactions. It also demonstrates the compatibility of the GPU package with other features in LAMMPS including bonds and external constraints.

The second model is a melt of A3B7 block copolymers, in which each polymer is modeled as a linear

Figure 6: (a) Model of a 1:1 mixture of two incompatible types of polymer chains whose the head groups are grafted to a substrate and the chain length ratio of 4:7 (model due to Singh *et al.* [11]). Adjacent beads in one chain are connected via harmonic springs with stiffness $k = 30.0$ and equilibrium length $r_0 = 1.0\sigma_0$. (b) Head groups randomly mix at high repulsion strength $a = 29.0$. (c) Head groups form stripes at $a = 18.75$. This is consistent with the equilibrium self-assembled structures for $a = 18.75$ reported by Ref. [11].

chain of 3 beads of type A and 7 beads of type B (Figure 7(a), inset). In a previous study, Horsch *et al.* observed the formation of the hexagonal cylinder phase at the given block fraction of species A, i.e., $\phi_A = 0.3$ [40]. We wish to examine if the simulated system size in the work of Horsch *et al.* would be sufficient to remove finite size effects. For this purpose, we simulated a system consisting of 156K particles, which is 5.7 times bigger than that in the previous study.

Figure 7: (a) Hexagonal cylinder phase formed in a A3B7 block copolymer melt consisting of 15625 chains ($N = 156250$ particles) at number density $\rho = 0.85r_c^{-3}$. Each chain is composed of 3 beads of type A (yellow) and 7 beads of type B (blue) (inset). (b) The lattice of the cylinders viewed along the cylinder axes with the B blocks removed for clarity. The repulsion strength between unlike species $a_{AB} = 42.0$ and between like species $a_{AA} = a_{BB} = 25.0$, or equivalently, $\chi N = 23.7$.

We used the same model and interaction parameters as those in Ref. [40] and performed a “cooling” procedure by increasing the repulsion strength between unlike species, a_{AB} , from 30.0 to 43.0. According to Groot and Warren [3], this is equivalent to increasing χN_C from 6.9 to 24.9, where χ is the effective Flory-Huggins parameter and N_C is the chain length, $N_C = 10$ in this case. Our simulation reveals that the melt undergoes a transition from disorder to the hexagonal cylinder phase at $a_{AB} = 39.0$. In the work of Horsch *et al.*, the phase was observed at $a_{AB} = 42.0$, which is consistent with our result. This indicates that the system size in the previous study is sufficient for predicting the formation of the hexagonal cylinder phase for the given block copolymer melt. However, only with the system size in our study do the cylinders in the triangular lattice align parallel to the x-y plane (Figure 7(b)) so that the lattice spacing along the z direction is not influenced by periodic boundary conditions. Further detailed analysis such as measurements in the diameter of the cylinders and the spacing between them should be done, but we leave them for later studies.

The last example is a thin film of block copolymers deposited on a solid substrate. As pointed out by previous studies [41, 42], large-scale simulations are required to capture the long-wave capillary modes in such films. By correlating the growth of those modes with the formation of the holes, if any, we can identify the dewetting mechanism of the film. We simulated a film of the A4B6 block copolymers, each of which is modeled as a linear chain as shown in Figure 8(a). The interaction between the monomers is modeled as the 12-6 Lennard Jones potential, instead of the soft-core potential in the standard DPD formulation. In this particular example, we employ the random and drag forces in DPD as a momentum conserving thermostat for the coarse-grained monomers. An equilibrated film with the initial thickness of approximately 6.0σ and other dimensions of 297σ by 297σ is deposited on a solid substrate, where σ is the length scale of the Lennard Jones potential. The substrate particles are arranged into a face-centered cubic lattice and kept immobile throughout the simulation. A vacuum space of 20σ is added on top of the film to avoid the periodic boundary effects in the z direction.

The total number of particles simulated is 1011800, of which 450K are the monomers. The simulation ran on 25 Titan compute nodes at the rate of 125 timesteps, or 0.625τ , per second.

Figure 8: (a) A section of a film of A4B6 block copolymers on a face-centered cubic substrate. Each chain is composed of 4 beads of type A (green) and 6 beads of type B (orange). The bonds between the monomers in the same chain are modeled as FENE springs. (b) Local film thickness after 1000τ relaxation and the corresponding 2D Fourier spectrum (inset).

To characterize the undulations of the film free surface, we divide the substrate area into a grid of 256 by 256 bins and determine the number of monomers in each bin based on their coordinates. The film thickness in a bin is calculated from the number of the monomers in the bin, the bin cross-sectional area and the monomer number density ($\rho = 0.85\sigma^{-3}$). For storage and computational efficiency, the statistics for this quantity are gathered on-the-fly using a spatial binning diagnostic in LAMMPS. As shown in Figure 8(b), after 1000τ relaxation, the local thickness of the films, although isotropic, exhibits a dominant correlation length. This correlation length, also called the characteristic wavelength of the surface undulation modes, is indicated by the bright ring in the 2D Fourier spectrum (Figure 8(b), inset). For the given simulated film, the wavelength

to be approximately 100σ , which is about one third of the film dimensions in x and y directions. Although this wavelength completely diminished after 5000τ and the film did not dewet after 50000τ , we emphasize that without large-scale simulations surface undulation modes would have been missed.

4. Conclusion

We have developed an accelerated version of dissipative particle dynamics (DPD) which runs on GPUs within the LAMMPS molecular dynamics package. For sufficiently large models (i.e., 200K or more particles per GPU) a single GPU can run simulation models nearly 10x faster than a 16-core CPU. (This is for the current-generation Kepler K20X GPUs and AMD Opteron 6274 CPUs on the Titan supercomputer at ORNL.) The implementation also shows good parallel scalability to thousands of compute nodes on the same machine.

The DPD model was added to the existing “GPU” package in LAMMPS, which allows multiple MPI tasks (on CPU cores) to use a single GPU and efficiently transfers particle data between the CPUs and GPU every timestep. This means that other features of a complex simulation (boundary conditions, constraints, diagnostics, etc) defined in LAMMPS can run efficiently on the CPU while the GPU computes the DPD interactions and neighbor lists.

This capability is an example of how large-scale coarse-grained mesoscopic models can be run efficiently on current-generation hybrid clusters and supercomputers containing both multi-core CPUs and accelerator chips (e.g. GPUs). Similar acceleration methodologies could be applied to other mesoscale models suitable for soft and hard materials, since many such models rely on relatively simple pairwise interactions. Examples include smoothed-particle hydrodynamics (SPH), PeriDynamics (PD), and potentials for granular materials.

5. Acknowledgments

The GPU and CPU DPD models described here are part of the open-source LAMMPS distribution, available for download at <http://lammps.sandia.gov>. T.D.N. thanks W. M. Brown for helpful discussion regarding the implementation of the GPU package and Arnold N. Tharrington for comments on the random number generators on the GPU. This research used resources of the Leadership Computing Facility at Oak Ridge National

Laboratory and was conducted under the auspices of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

1. P. J. Hoogerbrugge and J. M. V. A. Koelman. Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics. *Europhys. Lett.*, 19(3):155–160, 1992.
2. P. Español and P. B. Warren. Statistical mechanics of dissipative particle dynamics. *Europhys. Lett.*, 30(4):191, 1995.
3. R. D. Groot and P. B. Warren. Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation. *J. Chem. Phys.*, 107(11):4423–4435, 1997.
4. J. A. Backer, C. P. Lowe, H. C. J. Hoefsloot, and P. D. Iedema. Poiseuille flow to measure the viscosity of particle model fluids. *J. Chem. Phys.*, 122:154503, 2005.
5. D. Fedosov, G. E. Karniadakis, and B. Caswell. Dissipative particle dynamics simulation of depletion layer and polymer migration in micro- and nanochannels for dilute polymer solutions. *J. Chem. Phys.*, 128:144903, 2008.
6. M. J. A. Hore and M. Laradji. Dissipative particle dynamics simulation of the interplay between spinodal decomposition and wetting in thin film binary fluids. *J. Chem. Phys.*, 132:024908, 2010.
7. W. Dzwinel and D. A. Yuen. Dissipative particle dynamics of the thin-film evolution in mesoscale. *Mol. Simulat.*, 22:369–395, 1999.
8. S. Merabia and J. B. Avalos. Dewetting of a stratified two-component liquid film on a solid substrate. *Phys. Rev. Lett.*, 101:208304, 2008.
9. F. Martinez-Veracoechea and F. A. Escobedo. Simulation of the gyroid phase in off-lattice models of pure diblock copolymer melts. *J. Chem. Phys.*, 125:104907, 2006.
10. C. Singh, P. K. Ghorai, M. A. Horsch, A. M. Jackson, R. G. Larson, F. Stellacci, and S. C. Glotzer. Entropy-mediated patterning of surfactant-coated nanoparticles and surfaces. *Phys. Rev. Lett.*, 99:226106, 2007.
11. C. Singh, A. M. Jackson, F. Stellacci, and S. C. Glotzer. Exploiting substrate stress to modify nanoscale SAM patterns. *J. Am. Chem. Soc.*, 131:16377–16379, 2009.
12. J. R. Spaeth, I. G. Kevrekidis, and A. Z. Panagiotopoulos. Dissipative particle dynamics simulations of polymer-protected nanoparticle self-assembly. *J. Chem. Phys.*, 135:184903, 2011.
13. I. Pons-Siepermann and S. C. Glotzer. Design of patchy particles using quaternary self-assembled monolayers. *ACS Nano*, 6:3919–3924, 2012.
14. Y.-H. Tang and G. E. Karniadakis. Accelerating dissipative particle dynamics simulations on GPUs: Algorithms, numerics and applications. *arXiv:1311.0402v1*, 2013.
15. X. Li, Y.-H. Tang, H. Liang, and G. E. Karniadakis. Large-scale dissipative particle dynamics simulations of self-assembled amphiphilic systems. *Chem. Commun.*, 50:8306–8308, 2014.
16. Y. Kong, C. W. Manke, W. G. Madden, and A. G. Schlijper. Effect of solvent quality on the conformation and relaxation of polymers via dissipative particle dynamics. *J. Chem. Phys.*, 197:592–602, 1997.
17. G. Pan and C. W. Manke. Effects of solvent quality on the dynamics of polymer solutions simulated by dissipative particle dynamics. *J. Rheology*, 46:1221–1237, 2002.
18. V. Symeonidis, G. E. Karniadakis, and B. Caswell. Dissipative particle dynamics simulations of polymer chains: Scaling laws and shearing response compared to DNA experiments. *Phys. Rev. Lett.*, 95:076001, 2005.
19. T. Ye, N. Phan-Thien, B. C. Khoo, and C. T. Lim. Stretching and relaxation of malaria-infected red blood cells. *Biophys. J.*

- 105:1103–1109, 2013.
20. R. D. Groot and K. L. Rabone. Mesoscopic simulation of cell membrane damage, morphology change and rupture by nonionic surfactants. *Biophys. J.*, 81:725–736, 2001.
21. X. Li, I. Pivkin, H. Liang, and G. E. Karniadakis. Shape transformations of membrane vesicles from amphiphilic triblock copolymers: A dissipative particle dynamics simulation study. *Macromolecules*, 42:3195–3200, 2009.
22. E. Moeendarbary, T. Y. Ng, and M. Zangeneh. Dissipative particle dynamics in soft matter and polymeric applications – A review. *Int. J. Appl. Mech.*, 2:161–190, 2010.
23. W. M. Brown and Y. Masako. Implementing molecular dynamics on hybrid high performance computers - three-body potentials. *Comp. Phys. Comm.*, 184:2785–2793, 2013.
24. S. J. Plimpton and A. Thompson. Computational aspects of many-body potentials. *MRS Bulletin*, 37:513–521, 2012.
25. A. F. Voter. Parallel replica method for dynamics of infrequent events. *Phys. Rev. B*, 57:13985, 1998.
26. M. R. Sorensen and A. F. Voter. Temperature-accelerated dynamics for simulation of infrequent events. *J. Chem. Phys.*, 112:9599, 2000.
27. A. B. Bortz, M. H. Kalos, and J. L. Lebowitz. A new algorithm for monte carlo simulation of ising spin systems. *J. Comput. Phys.*, 17:10–18, 1975.
28. J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten. Scalable molecular dynamics with NAMD. *J. Comput. Chem.*, 26:1781–1802, 2005.
29. H. J. C. Berendsen, D. van der Spoel, and R. van Drunen. Gromacs: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91:43–56, 1995.
30. S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comp. Phys.*, 117:1–19, 1995.
31. J. A. Anderson, C. D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J. Comp. Phys.*, 227:5342–5359, 2008.
32. C. L. Phillips, J. A. Anderson, and S. C. Glotzer. Pseudo-random number generation for brownian dynamics and dissipative particle dynamics simulations on GPU devices. *J. Comp. Phys.*, 230:7191–7201, 2011.
33. Y. Afshar, F. Schmid, A. Pishevar, and S. Worley. Exploiting seeding of random number generators for efficient domain decomposition parallelization of dissipative particle dynamics. *Comp. Phys. Comm.*, 184:1119–1128, 2013.
34. S. Wang, J. Xu, and H. Wen. Accelerating dissipative particle dynamics with multiple GPUs. *Comp. Phys. Comm.*, 184:2454–2461, 2013.
35. W. M. Brown, P. Wang, S. J. Plimpton, and A. N. Tharrington. Implementing molecular dynamics on hybrid high performance computers - short range forces. *Comp. Phys. Comm.*, 182:898–911, 2011.
36. W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington. Implementing molecular dynamics on hybrid high performance computers - particle-particle particle-mesh. *Comp. Phys. Comm.*, 183:449–459, 2012.
37. W. M. Brown, T. D. Nguyen, M. Fuentes-Cabrera, J. D. Fowlkes, P. D. Rack, M. Berger, and A. S. Bland. An evaluation of Molecular Dynamics performance on the hybrid Cray XK6 supercomputer. *Procedia Comput. Sci.*, 9:186–195, 2012.
38. F. James. A review of pseudorandom number generators. *Comp. Phys. Comm.*, 60:329–344, 1990.
39. L. D. G. Sigalotti, J. Klapp, E. Sira, Y. Melean, and A. Hasmy. SPH simulations of time-dependent Poiseuille flow at low Reynolds numbers. *J. Comp. Phys.*, 191:622–638, 2003.
40. M. A. Horsch, Z. Zhang, C. R. Iacovella, and S. C. Glotzer. Hy-

- drodynamics and microphase ordering in block copolymers: Are hydrodynamics required for ordered phases with periodicity in more than one dimension? *J. Chem. Phys.*, 121:11455–11461, 2004.
41. T. D. Nguyen, J.-M. Y. Carrillo, M. A. Matheson, and W. M. Brown. Rupture mechanism of liquid crystal thin films realized by large-scale molecular simulation. *Nanoscale*, 6:3083–3096, 2014.
42. T. D. Nguyen, M. Fuentes-Cabrera, J. D. Fowlkes, and P. D. Rack. Coexistence of spinodal instability and thermal nucleation in thin film rupture: Insights from molecular levels. *Phys. Rev. E*, 89:032403, 2014.