Supporting Information: Using Reactive Dissipative Particle Dynamics to Understand Local Shape Manipulation of Polymer Vesicles

Qinyu Zhu, Timothy R. Scott, and Douglas R. Tree**

Chemical Engineering Department, Brigham Young University, Provo, Utah

E-mail: tree.doug@byu.edu

Description of Parallel Cell List Algorithm

Instead of using a Verlet list of neighboring particles within a certain cut-off radius, we divide the simulation space into cells of uniform size r_c and use a cell list to sort the particles and count the pairwise interactions.[S1] Here we briefly explain the basic concept of cell list structure using a 2D system, as shown in Figure S1. The cell list structure consists of a head array and a linked-list array. The system in Figure S1 is divided into nine uniform cells. Each element in the head array corresponds to one cell and stores the first particle's index that belongs to the cell. The elements in the head array also point to the address of the next particle index in the linked-list array. The elements in the linked-list represent the particle indices, and also points to the next particle in the same cell, and so on. If we follow the trace of the linked-list array, we will reach an element of -1, which means that we have iterated through all the particles in this cell.

When calculating the pairwise interactions for a particle, one only needs to check the particles in the same cell and half of the neighboring cells. For example, if we are calculating the pairwise interaction for particle 7 in Figure S1, we only need to check the particles in cells 0, 1, 2, and 5. Other pairs that involve particle 7 will be taken into account when we deal with particles in the other half of the neighboring cells. The cell list needs to be updated at every time step. The algorithm is efficient in large systems with a computational time that scales linearly with the number of particles.

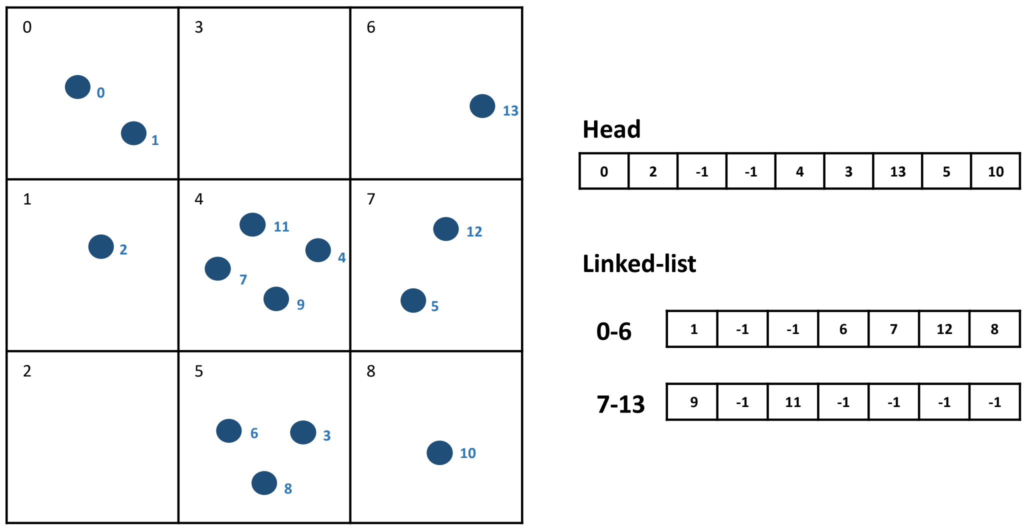

Figure S1: Demonstration of cell list structure in a 2D system.

However, constructing the cell list using the algorithm above is intrinsically a serial process, since we need to successively loop through each and every particle in the system to place them in the right location. Constructing the cell list on the CPU and then transferring it to the GPU is one less desirable option. This memory transfer process can be the rate-limiting step for a parallel computation. To avoid this, we used the the compare_and_swap function, which is an atomic operation in CUDA,[S2] to set up the cell list on GPU. The pseudocode for this procedure is found in Algorithm 1. To better interpret the algorithm, we define several functions here that we later refer to in Algorithm 1,

1. **Initialize**($head, clist$): initialize the head array ($head$) and linked-list array ($clist$) and set all the elements in both arrays to -1.

2. **Identify_Cell(*idx*)**: identify the cell index of a corresponding particle index *idx*.

3. **compare_and_swap($array[n], old\_val, new\_val$)**: check if element $array[n]$ has the value of *old_val*. If so, then the *new_val* is assigned to this element; otherwise, the element remain unchanged.

We first initialize the head array and linked-list array on GPU and obtain the thread id from the GPU, as suggested in the first two lines. Each thread is then assigned to a single particle, and the cell index that the particle belongs to is identified. In line 5, we use the **compare_and_swap** function to attempt to swap the corresponding element in the head array with the particle index. If the index is successfully swapped, this thread returns and proceeds to process the next particle in the queue. Otherwise, we locate the next address in the linked-list and repeat the *compare_and_swap* function until a swap is successful, as suggested in the while loop from line 10 to line 18.

Algorithm 1 Parallel Cell List Construction

```
 1: Initialize($head, clist$)
 2: Set $idx = threadIdx.x + blockDim.x * blockIdx.x$  #Thread id from GPU, also referred
    to as the particle index that the thread is processing
 3: Set $idcell$ = Indentify_Cell($idx$)
 4: compare_and_swap($head[idcell], -1, idx$)
 5: if $head[idcel] = idx$ then
 6:    return and proceed to next particle
 7: else
 8:    Set $swap$ = True
 9:    while $swap$ = True do
10:       $nextid \leftarrow head[idcell]$
11:       compare_and_swap($clist[nextid], -1, idx$)
12:       if $clist[nextid]) = idx$ then
13:          $swap$ = False
14:       else
15:          $nextid \leftarrow clist[nextid]$
16:       end if
17:    end while
18: end if
```

Relaxation Time of the Diblock Copolymer Solution

To characterize the relaxation time, we performed a DPD simulation using the same parameters as our polymer vesicle system described in the main text starting from random initial conditions. We ran the simulation for 5×10^5 steps, and the average radius of gyration (R_g) was obtained every 50 steps. The normalized autocorrelation function is plotted in Figure S2, where it shows an initially exponential decay before it becomes noisy due to insufficient statistics. Accordingly, we fit the data from $\tau = 0$ to $\tau = 20000$ using the following model

$$\ln E(\tau) = -\frac{\tau}{\theta} \tag{1}$$

where E is the normalized autocorrelation function of Rg, and θ is the autocorrelation time. The autocorrelation time obtained from the fit is $\theta \approx 1.43 \times 10^4$ timesteps. As mentioned in the main text, we typically ran for 1×10^6 timesteps for the final equilibration of the vesicle structure, which is approximately $70\times$ the autocorrelation time. Thus, it is reasonable to conclude that our final system is effectively relaxed.

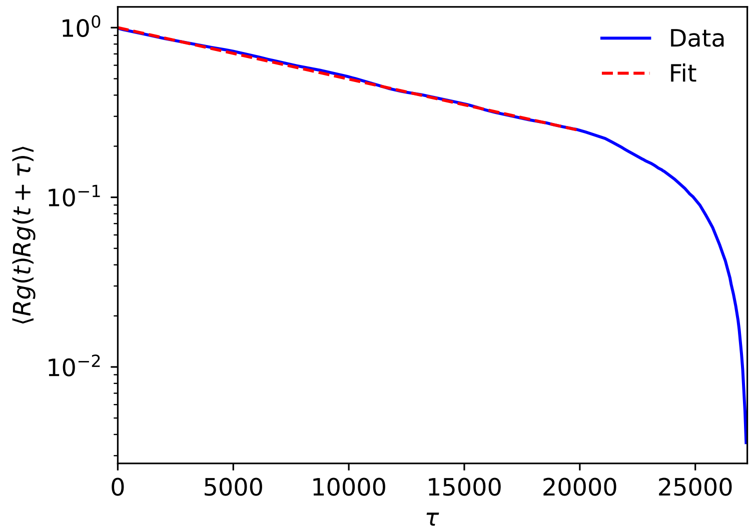

Figure S2: Normalized autocorrelation function of R_g.

Lateral Diffusion Coefficient of Polymer Chains in the Vesicle

In the main manuscript, we presented the result of mean squared displacement (MSD) of DPD solvent beads as a function of time and fitting the data points obtain tracer diffusion of the solvent $\mathrm{S_A}$, $D_{\mathrm{S_A}} = 0.2296$ in DPD unit. Here, We performed a similar calculation to obtain the lateral diffusion coefficient of polymer chains within the vesicle by tracking and averaging the MSD of center of mass of the polymer chains as a function of time and fitting the data point to

$$MSD = 6D_{\mathrm{Lateral}}t \tag{2}$$

where D_{Lateral} is the lateral diffusion coefficient of the polymer chains in DPD unit. The MSD is plotted against time in Fig. S3. The fitted lateral diffusion coefficient is $D_{\mathrm{Lateral}} = 0.0073$, which is approximately 30 times slower than that of the individual solvent bead.

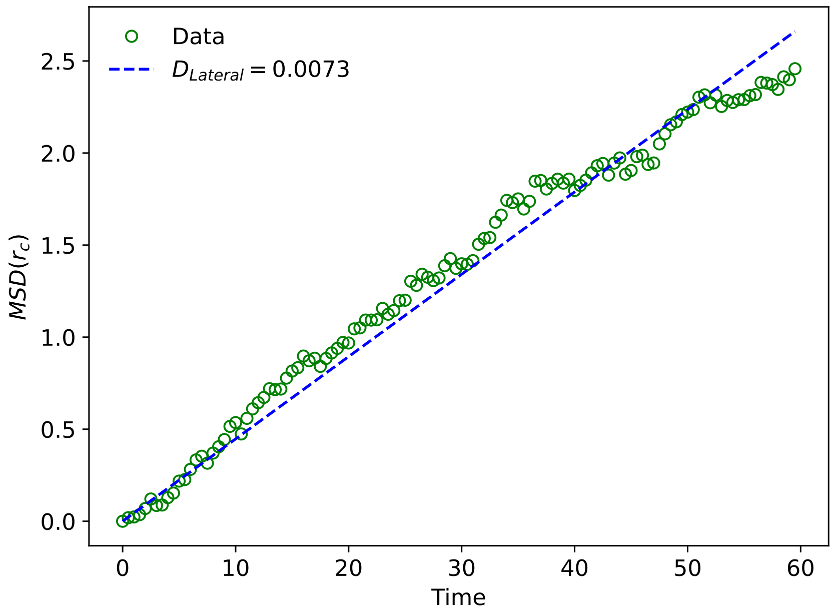

Figure S3: The MSD of polymer chains as a function of time.

Run-Time Comparison of DPD and *RDPD* Codes

In the main text, we show a bar chart that compares the run time of serial and parallel versions of DPD and *RDPD* codes. Here we attach the data we used to create the bar chart.

Table S1: Time comparison

Code Version	Run Time (min)
Serial DPD	24.62
Serial *RDPD* with Naive Algorithm	39.21
Serial *RDPD* with SRBD	32.54
Parallel DPD	1.27
Parallel *RDPD* with Naive Algorithm	1.60
Parallel *RDPD* with SRBD	1.43

Validation of SRBD Equilibria at Other Reaction Rates

In the main text, we validate the SRBD equilibrium by simulating the following catalytic conversion reaction,

$$S_{\mathrm{A}} + \mathrm{E} \rightleftharpoons S_{\mathrm{B}} + \mathrm{E}. \tag{3}$$

where $k_f = 0.1$ and $k_r = 0.05$. Here we show additional sets of forward and reverse reaction rates that also demonstrate similar equilibria. The simulations started with a random initial condition of equal mole fractions of A and B. All the simulations have the same forward reaction rate of $k_f = 0.1$. The reverse reaction rate in Figure S4 a, b, c and d are $k_r = 0.02$, 0.1, 0.2, and 0.5 respectively. The red lines represent the theoretical equilibrium mole fraction x_{A} for the corresponding reaction rate parameters. In Figure S4, the value of $x_{S_{\mathrm{A}}}$ fluctuates around the equilibrium mole fraction after $t \approx 1500$, in agreement with the expected values of equilibrium concentration with respect to the various sets of forward and reverse reaction rates.

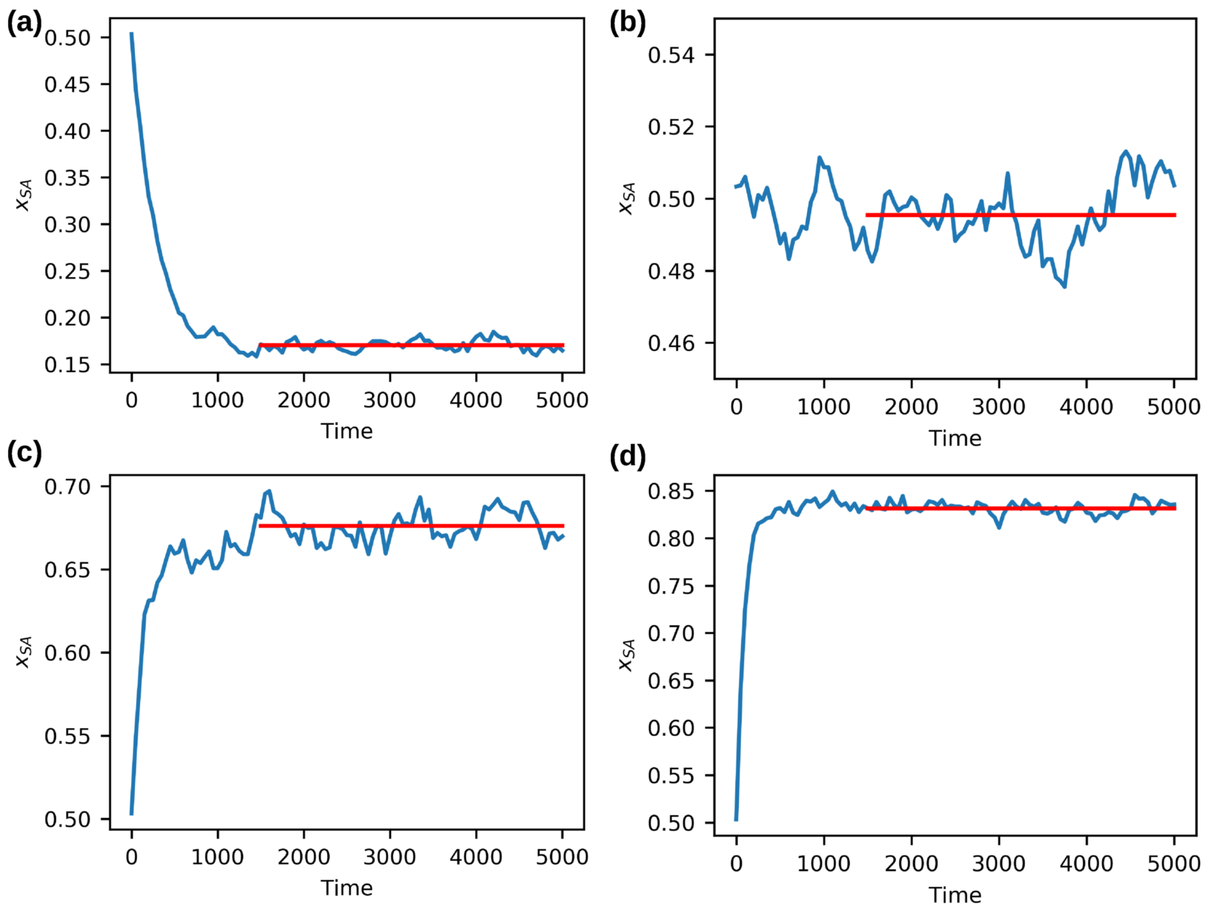

Figure S4: Mole fraction x_{S_A} as a function of time for reversible catalytic reactions at $k_f = 0.1$ and (a) $k_r = 0.02$, (b) $k_r = 0.1$, (c) $k_r = 0.2$, (d) $k_r = 0.5$.

Characterizing the monomer conversion in Fig. 12

To better demonstrate the process of dynamically changing polymer chemistry, as shown in Fig. 12 in the main text, we ran 20 replicates of the corresponding RDPD simulation, and plotted the number of converted B beads against the simulation time in Fig. S5. As suggested by the trend of the curve, the conversion rate slowly decreases as the reaction proceeds, which is reasonable because the consumption of reactants would decrease the probability of successful collision that leads to the chemical reactions. On average, 3681 B particles were converted at the end of the reaction (at $t = 500$).

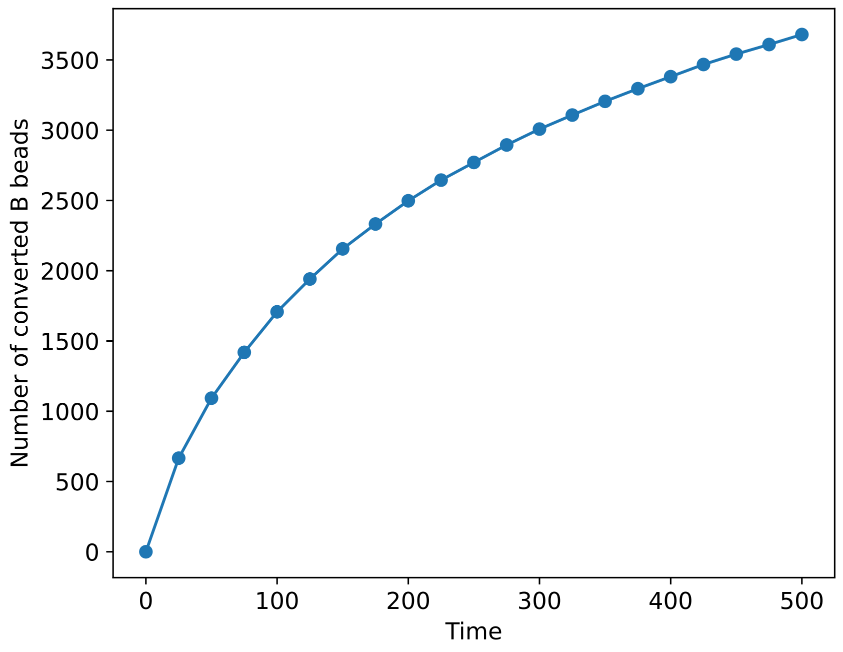

Figure S5: Number of B beads converted as a function of time.

Details of the Calculation of the Vesicle Curvature

As discussed in the main text, we estimated the vesicle curvature to show that the deformation occurs locally. The snapshots of the vesicle structure before and after dynamically changing the solvophobicity of B blocks were projected onto the Y-Z plane since the $S_{A'}$ particles were placed symmetrically with respect to the X and Z axes. The 2D space was divided into equally spaced grid points with a bin size of 0.5 in both directions. To get a smoother surface for a more accurate curvature estimation, we removed the A blocks in the outer corona, and we averaged the coordinates over 100 snapshots. Additionally, to further reduce the noise amplitude in the density calculation each particle was treated as a Gaussian distributed density pulse

$$f(y,z) = A\exp(-(\frac{(y-y_0)^2}{2{\sigma_y}^2} + \frac{(z-z_0)^2}{2{\sigma_z}^2})) \tag{4}$$

where y and z represent the grid point, and y_0 and z_0 are the coordinates of the particles. The amplitude was set as $A = 1$, and $\sigma_y = \sigma_z = 1$. A threshold between 0.5 and 75 was applied to obtain the grid points constituting the outer layer of the projection. The alpha shapes, which describe the boundary that envelops a set of points, were then defined with $\alpha = 2.0$ to obtain the outlines of the projection for curvature estimation.[S3] The curvatures of the 2D projection were estimated after parameterizing this curve as $\gamma(s) = (y(s), z(s))$ where s is a circumferential index. We then define a curvature as[S4]

$$\kappa = \left| \frac{y'z'' - z'y''}{[(y')^2 + (z')^2]^{3/2}} \right| \tag{5}$$

where κ is the absolute curvature, $y' = dy/ds$, $z' = dz/ds$, $y'' = d^2y/ds^2$, and $z'' = d^2z/ds^2$, respectively. The first and second derivatives in Eq. 5 were calculated based on a third order polynomial that was fit locally using a Savitzky–Golay filter with a window length of 23.[S5]

References

(S1) Allen, M.; Tildesley, D. *Computer Simulation of Liquids*; Oxford science publications; Clarendon Press, 1987.

(S2) Nickolls, J.; Buck, I.; Garland, M.; Skadron, K. Scalable parallel programming with CUDA. *Queue* **2008**, *6*, 40–53, DOI: 10.1145/1365490.1365500.

(S3) Edelsbrunner, H.; Kirkpatrick, D.; Seidel, R. On the shape of a set of points in the plane. *IEEE T Inform Theory* **1983**, *29*, 551–559, DOI: 10.1109/TIT.1983.1056714.

(S4) Kobayashi, S.; Nomizu, K. *Foundations of differential geometry*; New York, London, 1963; Vol. 1.

(S5) Savitzky, A.; Golay, M. J. Smoothing and differentiation of data by simplified least squares procedures. *Anal Chem* **1964**, *36*, 1627–1639, DOI: 10.1021/ac60214a047.