Accelerating a Novel Particle-based Fluid Simulation on the GPU

Zhilu Chen+, James Kingsley*, Xinming Huang+, Erkan Tüzel*
+Department of Electrical and Computer Engineering, Worcester Polytechnic Institute
*Department of Physics, Worcester Polytechnic Institute

***Abstract*—Stochastic Rotation Dynamics (SRD) is a novel particle-based simulation method that can be used to model complex fluids [1], [2], such as binary and ternary mixtures [3], and polymer solutions [4]–[6], in either two or three dimensions. Although SRD is efficient compared to traditional methods, it is still computationally expensive for large system sizes, e.g. when using a large array of particles to simulate dense polymer solutions. Recently, as the power offered by Graphics Processing Units (GPUs) has risen, General Purpose GPU (GPGPU) computing has been introduced as an effective way to improve performance for parallel computation tasks. This work focuses on the acceleration of SRD simulations using Nvidia's GPGPU architecture, CUDA. We find that while the speed improvements delivered by GPU acceleration vary with the simulation version and parameters used, our GPU implementation runs around 10 times faster than the CPU version for basic simulations, and up to 50 times faster for polymers in solution.**

***Index Terms*—CUDA, GPU, SRD, MPCD, parallel computing**

I. Introduction

Computer simulations are widely used in scientific research. Quite often simulations become computationally intensive and thus require large cluster computing architectures with expensive hardware. Even with these powerful computation resources, large scale simulations of fluids with embedded objects, such as modeling of red blood cells in flow [7], [8], or spermatozoa in channels [9], [10], can take up to several months to obtain satisfactory results.

Compute Unified Device Architecture (CUDA) programming [11]–[13] makes it possible to parallelize a program to run on the many cores of a GPU. In a GPU implementation, the program is divided into small parts that run in many threads independently and simultaneously. Thus it is possible to accelerate the simulation by parallelizing the computing tasks on a GPU instead of using expensive computer clusters [14]–[16].

In this paper we focus on accelerating a novel particle-based fluid simulation technique called Stochastic Rotation Dynamics (SRD). Also known as Multi-Particle Collision Dynamics (MPCD), SRD was originally proposed by Malevanets and Kapral [1], [2]. SRD is an ideal computational tool for solving the underlying thermo-hydrodynamic equations by providing a "hydrodynamic heat bath" which incorporates thermal fluctuations and has the correct hydrodynamic interactions between embedded particles or polymers. The method does not suffer from instabilities found in methods such as Lattice-Boltzmann, finite-difference and finite-element approaches [6]. Furthermore, its simplicity has made it possible to obtain analytic expressions for the transport coefficients [17]–[20], something that is often very difficult to do for other mesoscale particle-based algorithms. SRD has been an excellent tool to study various problems in soft matter physics in the past decade. Notable applications include polymer solutions [2], [4], [6], colloids including sedimentation [21], [22], vesicles and star polymers in shear flow [23], and modeling of swimming of fish [24], and spermatozoa [10]. In this paper, we show how this technique can be implemented on the GPU to improve its performance.

II. The algorithm

A. The solvent

In the SRD method, the fluid is divided into boxes with side length l. If we denote the number of boxes in one dimension with N, and the number of particles in each box with n, there are nN^2 particles in the simulation box in two dimensions. Each particle has a mass m, coordinate $\vec{r} = (x, y)$, and velocity $\vec{v} = (v_x, v_y)$. We can calculate which box a particle belongs to using its coordinates. Fig. 1 illustrates a sample simulation grid with the randomly distributed particles and representative velocity vectors.

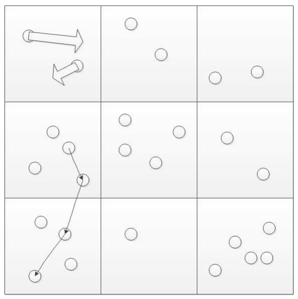

Figure 1. Example SRD grid showing the particles, and an embedded polymer. Particle velocities are shown with arrows.

The dynamics consists of two steps: (i) streaming, and (ii) collision. In the streaming step, the particles move according to

$$\begin{aligned} x(t+\Delta t) = &\quad x(t) + v_x(t)\Delta t \\ y(t+\Delta t) = &\quad y(t) + v_y(t)\Delta t \end{aligned} \tag{1}$$

where Δt is the time step.

In the collision step, we calculate the average velocity of each box, given by

$$\vec{U} = \frac{1}{n} \sum \vec{v}(t) \ . \tag{2}$$

A rotation matrix for each box is then generated, i.e.,

$$R = \begin{pmatrix} W_{xx} & W_{xy} \\ W_{yx} & W_{yy} \end{pmatrix} = \begin{pmatrix} \cos\alpha & \pm\sin\alpha \\ \mp\sin\alpha & \cos\alpha \end{pmatrix} \tag{3}$$

where α is the rotation angle, and the direction of the rotation is chosen stochastically. The velocities of the particles are then updated using

$$\begin{pmatrix} v_x(t+\Delta t) \\ v_y(t+\Delta t) \end{pmatrix} = \begin{pmatrix} U_x \\ U_y \end{pmatrix} + \begin{pmatrix} W_{xx} & W_{xy} \\ W_{yx} & W_{yy} \end{pmatrix} \cdot \begin{pmatrix} v_x(t) - U_x \\ v_y(t) - U_y \end{pmatrix} \tag{4}$$

And the algorithm repeats with the streaming step.

The algorithm conserves mass, energy and momentum [6]. In order to insure Galilean invariance, a grid shift operation is implemented before the collision step [25]. This is done by randomly offsetting particle coordinates in the range $[-l/2, l/2]$ for each collision step. In the simulations presented in this paper, periodic boundary conditions are used.

B. Embedding of Polymers

We model polymers using a semi-flexible bead-spring model, using standard fluid particles as the polymer beads. The total energy of the polymer is given by

$$U = U_{spring} + U_{bend} \tag{5}$$

where U_{spring} is the potential that controls stretching/compression of the polymer backbone given by

$$U_{spring} = \frac{k}{2} \sum \left(|\vec{R}_{i-1} - \vec{R}_i| - \ell_0 \right)^2 \tag{6}$$

and U_{bend} is the bending energy that controls the degree of bending. The bending potential is defined as

$$U_{bend} = -\frac{\kappa}{\ell_0} \sum \frac{\left(\vec{R}_{i-1} - \vec{R}_i\right) \cdot \left(\vec{R}_i - \vec{R}_{i+1}\right)}{|\vec{R}_{i-1} - \vec{R}_i||\vec{R}_i - \vec{R}_{i+1}|} \ . \tag{7}$$

Here $\vec{R}_i$ is the position of the polymer bead i, ℓ_0 is the rest length of the springs, k is the bond spring constant, and κ is the bending rigidity. Fig. 1 gives an example of how a polymer is embedded in the SRD simulation grid. In the simulations the spring potential is kept strong enough to ensure the polymer length does not change more than one percent, mimicking rigid bonds. The equations of motion for the polymer are solved using a Velocity-Verlet scheme [26] for several intermediate time steps in between consecutive solvent streaming steps given by Equation 1.

For certain applications, it is also desirable to have non-crossing polymers. This is implemented with a short-range repulsive truncated-shifted Lennard-Jones potential [27] between every point on every polymer. Fig. 2 illustrates the forces between polymers.

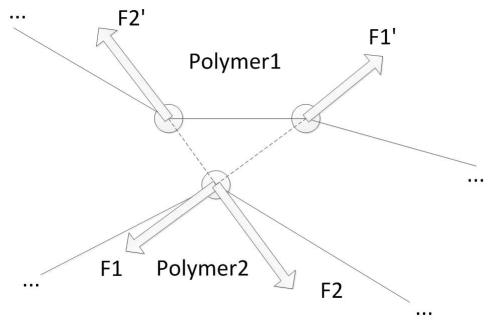

Figure 2. Illustration of forces between node particles between different polymers.

III. GPU Implementation

A. Initializing the simulation

We use a function on the CPU host for initialization, because the initialization functions are executed only once and their execution time is short compared to the full simulation time. Vector variables in Euclidean space, such as positions and velocities, are represented as double-precision floating point coordinates along the x and y axes for the two dimensional simulations, and with an additional z component for the three dimensional simulations. For the purpose of simplicity, we mainly discuss the two dimensional case here, as it is relatively easy to extend the approach to three dimensional simulations.

The field is divided into boxes. The box structure has the following properties: a momentum vector, number of particles inside the box, and the direction of random rotation for that box for that time step. Each box has multiple particles. The particle structure has the following properties: a coordinate vector, a velocity vector, mass, and an integer vector to record how many times the particle has crossed the periodic boundaries. We use one array to store the particles and another array to store the polymers. Although each polymer is an array of particles, we still use a one-dimensional array to store all the particles for all polymers, for the convenience of the GPU implementation, i.e.,

$$i = i_{poly} \times n + i_{part} \tag{8}$$

where i is the index of the particle in the one-dimensional array, n is the number of node particles in a polymer, i_{poly} is the index of the polymer and i_{part} is its index in that polymer.

Upon initialization, the arrays of boxes, particles and polymers are stored in the host memory. We then copy these arrays to global memory using the `cudaMemcpy( )` function. The data stays in the global memory until the simulation is complete, avoiding having to copy between host and GPU memory at every time step, which is time consuming.

B. SRD kernels

The particle (solvent) simulation loop is outside the kernel because each iteration depends on the results of previous iteration, requiring each step to be sequential. For each iteration of the particle simulation, the following 4 kernel functions are launched. The first one is `kernel_ResetBoxes( )`. It runs with one thread per box. For each box, it resets the mean momentum and particle count to zero. The second one is `kernel_SumMomenta( )`. It runs with one thread per particle. For each particle, it locates the box in which the particle belongs, based on the particle's position and the current grid-shift. It then adds that particle's momentum to the total momentum of the box, and increments the number of particles in that box. Since the particle to box mapping is unpredictable, this addition is done using double precision atomic operations [28]. The following function is `kernel_DivideBoxes( )`, again running with one thread per box. For each box, it calculates the mean momentum of the ith box

$$\vec{u_i} = \vec{p_i}/c_i \tag{9}$$

where $\vec{p_i}$ is the total momenta and c_i is the counted number of particles of the ith box. The function also generates the random direction $r = \pm 1$ for the rotation matrix.

Finally, we have the `kernel_Collide_Move( )` function. In this step, we combine the streaming and collision steps into a single kernel function to save on synchronization and GPU kernel initialization time. Particles move with their given velocities during time step Δt, after which they collide, as described in Eq (4). Fig. 3 shows the procedure of the GPU code for the SRD simulations.

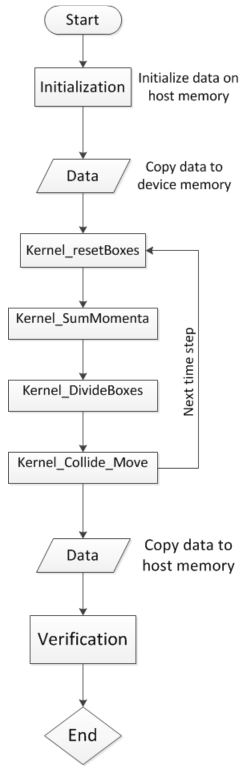

Figure 3. Flowchart of the GPU implementation of the SRD algorithm.

C. Polymer kernels

The nature of the polymer's simulation style complicates the GPU implementation. The separate, iterated Velocity-Verlet steps require a separate GPU kernel. Therefore, `kernel_Collide_Move( )` is split into `kernel_Collide( )` and `kernel_Move( )`. `kernel_Collide( )` works on both particles and polymers. `kernel_Move( )` is only used for solvent particles. A new kernel function called `kernel_MolecularDynamics( )` is created for polymers. To achieve effective synchronization, we assign one block to each polymer. This allows us to synchronize the block between each iteration without terminating the kernel. In addition, faster shared memory is used for communication among threads within the same block.

As previously mentioned, the time step for calculating forces and simulating the movement of the polymer particles is much smaller than the regular SRD time step. For every small time step, the forces of every two node particles are calculated using device function `calcForce( )`. Since the force on a particle depends on its location relative to its neighbors, each thread needs a synchronized view of its neighbors. Each thread has access to the shared memory handled by other threads within the same block, and thus the particles in shared memory are shared. During time steps, we use `__syncthreads( )` to synchronize within the block, ensuring a consistent particle state.

At the beginning of the molecular dynamics time step, we call the device function `calcForce( )` to compute forces. This is followed by a loop consisting of the movement update portion of Velocity-Verlet, another force calculation, and then the velocity update portion. This is repeated as necessary for the Velocity-Verlet movement step to occupy as much time as the Eulerian movement step it replaces. Once complete, the polymers are written back to global memory.

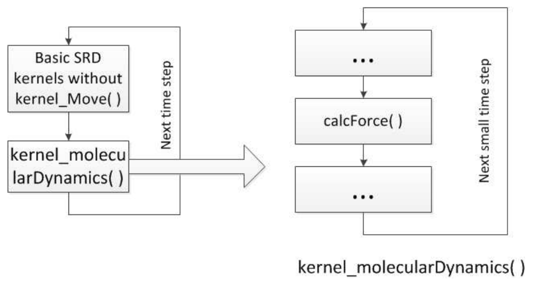
Basic SRD kernels without kernel_Move( )
kernel_molecularDynamics( )
Next time step
...
calcForce( )
...
Next small time step
kernel_molecularDynamics( )

Figure 4. Flowchart of polymer simulation

D. Lennard-Jones algorithm

The `kernel_molecularDynamics( )` function only concerns the interactions between the node particles in the same polymer. When we wish to use the Lennard-Jones interaction between polymers, the situation is further complicated, due to the large number of particles (more than can be simulated by a single block) that require consistent synchronization. As a result, we convert the `kernel_MolecularDynamics( )` function into several kernel functions and use a host-side loop to handle the small time step. Intra-polymer forces are calculated normally, and a separate `kernel_Lennard_Jones( )` function handles the external interactions. In this kernel function, each thread checks its pair of node particles, and if necessary applies a force to both. If so, we calculate the forces and use double-precision atomic operations to update the forces because they may also be updated by other threads. Fig. 5 illustrates the polymer kernels with Lennard-Jones algorithm included in the simulation.

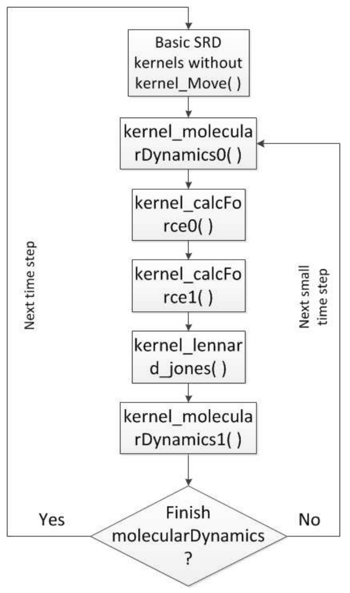
Basic SRD kernels without kernel_Move( )
kernel_molecularDynamics0( )
kernel_calcForce0( )
kernel_calcForce1( )
kernel_lennard_jones( )
kernel_molecularDynamics1( )
Finish molecularDynamics ?
Yes
No
Next time step
Next small time step

Figure 5. Flowchart of polymer simulation with Lennard-Jones algorithm included

E. Finalizing the simulation

When the simulation is completed, we need to verify the results. First, we copy the data from device memory back to host memory using `cudaMemcpy( )`. Then we run a function to compute the total momenta and energy of all the particles as well as polymers to verify energy and momentum conservations. Finally, we release the occupied memory on the GPU using `cudaFree( )` function.

IV. Performance Evaluation

During our experiment, we measure the computing performance on the server with two Intel Xeon X5650 CPUs (2.66GHZ, 12 cores, hyperthread) and two nVidia Tesla C2050 (1.15 GHz, 448 cores) GPUs with Compute Capability 2.0. Initializing and finalizing time is ignored in both CPU and GPU evaluations. There are no memory copy operations during the computation. The overhead of launching kernels is included in order to get the overall timing results. The CPU version used as a comparison is a high-performance single-threaded code, designed to be run in a parallel-experiment environment. Its performance is primarily memory-bound (in experiments small enough to fit entirely in CPU cache–less than 100,000 particles–a 3x speedup is observed), making a multi-threaded version perform worse on most parameter sets than this single-threaded version.

A. SRD performance

The speedup of our CUDA code varies for different parameters in the simulation. The number of boxes in our 2D simulation and the number of particles in each box are significant. All computations use double-precision floating-point. One thousand iterations are performed. Fig. 6 shows the execution time of CPU over GPU, which is essentially the speedup factor of GPU over CPU, for the simulations with different grid size and each grid initially contains 9 particles. So the largest case of 500-by-500 grid simulates the movement of a total of 2.25 million particles. Fig. 7 shows the same performance comparison for a fixed grid size of 128-by-128 while varying the number of particles in each grid. The largest case here is about 4.92 million particles in total. Since the GPU has a large, high-speed (GDDR5) memory, much larger simulation case can be executed on the GPU without significant performance degradation. Both figures show that a 10x speedup is readily achievable by using a single GPU when compared to the CPU. Further speedup is achievable through the use of multiple GPUs and also by further optimization of the GPU code.

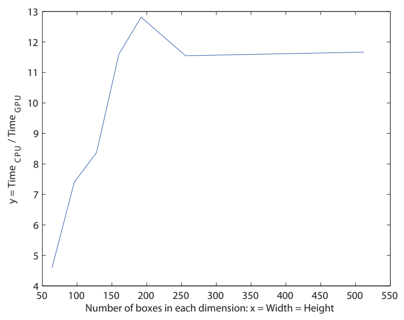

Figure 6. Speedup of GPU over CPU with different SRD grid size

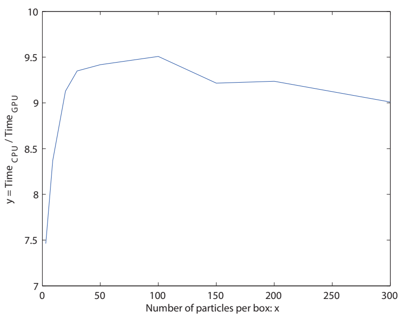

Figure 7. Speedup of GPU over CPU for SRD with different number of particles in each grid

We also tested our code in three dimensional cases. Table I shows the speedup in 3D grids. We can see that the performance is similar to that of 2D grids.

The total number of the particles increases rapidly with the grid size and thus we need to pay attention to the maximum number of blocks we can have in a kernel function. Since

Table I
SPEEDUP OF GPU OVER CPU IN 3D GRIDS

Particles per box \ Grid size	16^3	32^3	64^3
5	5.03	9.42	12.70
25	5.50	8.42	7.71

our Tesla C2050 has compute capability of 2.0, the maximum number of x-dimension of thread blocks is 65535, which is $2^{16}-1$. With 512 threads in one block, the maximum number of threads we can have is approximately 2^{25} in total. If the grid size is N^3 and we have n particles in one box, the total number of particles will be nN^3 which must be less than $2^9 \times (2^{16}-1)$. This is because the number of particles is much larger than that of boxes, and in some of the kernel functions we use one thread for each particle. In order to process larger grid with more particles in one box, we may let one thread to handle several particles with a loop inside the kernel. However, such loops themselves are not run in parallel, so there will be no contributions of these additional particles.

Some new GPUs have compute capability of 3.0 or higher, delivering a maximum number of $2^{32}-1$ thread blocks, which effectively eliminates the thread limitation.

B. Performance of polymer simulation

As before, the speedup depends on several parameters in the simulation. Besides the previously mentioned parameters, the number of polymers and number of particles per polymer are also important. We test this with the 2D version, with different number of polymers added and with different number of node particles per polymer. The execution time is much longer for both the CPU and GPU, but the speedup is much higher. Table II shows the speedup with 5 particles per box and grid size 128^2. Table III shows the speedup with 5 particles per box and grid size 256^2. We can see that the speedup increases with the number of polymers and number of node particles per polymer. This indicates that short-timestep polymer loop kernel has an even larger advantage over the CPU than the regular SRD kernels do.

Table II
SPEEDUP OF GPU OVER CPU WITH POLYMERS ADDED IN GRID WITH SIZE 128^2

Polymers \ Particles per polymer	32	64	128	256	512
1	4.95	5.20	5.38	6.43	7.29
10	7.47	9.14	13.90	23.57	35.92
20	7.07	10.39	16.37	29.87	44.82
30	6.61	10.83	18.18	32.40	49.17
40	8.06	13.19	24.11	41.39	62.45
50	7.47	12.74	23.70	42.02	63.25

Since the computation tasks of `molecularDynamics( )` are intensive and it is well parallelized on GPU, the speedup result is impressive.

Table III
SPEEDUP OF GPU OVER CPU WITH POLYMERS ADDED IN GRID WITH SIZE 256^2

Polymers \ Particles per polymer	32	64	128	256	512
1	9.55	9.61	9.84	10.09	10.10
10	10.61	11.77	13.98	17.63	24.88
20	9.68	11.81	15.63	22.63	32.66
30	9.43	11.69	16.61	25.68	37.76
40	10.17	13.31	19.92	31.92	47.89
50	9.55	13.37	20.19	32.77	49.36

C. Performance with Lennard-Jones algorithm included

While splitting the kernel function solves the synchronization issue, it does degrade performance. It is easy to see this from Table IV. There is only one polymer in the grid, which means that there is no need to use Lennard-Jones algorithm. The grid size is 128^2 and there are 5 particles in one box.

Table IV
SPEEDUP OF GPU OVER CPU WITH OR WITHOUT LENNARD-JONES ALGORITHM

Splitting kernels \ Particles per polymer	32	64	128
No	9.55	9.61	9.84
Yes	0.64	0.64	0.70

On the other hand, when there are many polymers, this puts a large load on `kernel_Lennard_Jones( )`, due to the necessity of comparing every pair of polymer particles. While this is a problem for the parallel GPU implementation, it still handles it better than the CPU implementation, and thus we are still able to achieve a significant speedup.

Table V shows the speedup with 5 particles per box and grid size 128^2. We can see that the performance is increasing with the number of polymers and number of node particles per polymer, again indicating that the GPU favors the polymer loops.

Table V
SPEEDUP OF GPU OVER CPU WITH LENNARD-JONES ALGORITHM INCLUDED IN GRID WITH SIZE 128^2

Polymers \ Particles per polymer	32	64	128
1	0.64	0.64	0.70
10	5.11	15.51	36.78
20	15.94	34.86	59.00
30	25.69	43.09	55.00
40	32.80	47.86	48.31
50	36.54	43.75	43.75

Compared with Table II which has the same grid size and number of particles per box, we can see that the speedup is minorly affected, but still achieves satisfying results. With the speedup is from 5 up to 43, the simulation is highly accelerated.

V. Conclusions

In this paper, we describe the particle-based SRD method along with modeling of polymers. We implement the algorithm in CUDA to improve its performance with a GPU, and compare it directly to the production-ready CPU implementation. We observe how different parameters, such as grid size and dimension, particle density, polymer length, and polymer density affect the comparative performance of both version. While we see a significant improvement, we could see further improvements by optimizing our kernel functions, data structures and even the algorithm itself. For example, in the Lennard-Jones algorithm, most node particles are outside of the interaction distance, and could be culled without a full test, giving a higher density of GPU threads processing real interactions.

The GPU approach achieves a speedup of 10 times over the CPU alone for basic SRD simulations, while once polymers are added, it increases up to 60 times, and 40 times with Lennard-Jones algorithm included. In conclusion, GPU acceleration is a cost-effective, high-performance computing method that is well-fitted for simulations in scientific research, especially for large-scale simulations taking weeks or months on conventional CPU-based software.

References

[1] A. Malevanets and R. Kapral, "Continuous-velocity lattice-gas model for fluid flow," *EPL (Europhysics Letters)*, vol. 44, no. 5, p. 552, 1998.

[2] ——, "Mesoscopic model for solvent dynamics," *J. Chem. Phys.*, vol. 110, pp. 8605–8613, 1999.

[3] E. Tüzel, G. Pan, T. Ihle, and D. M. Kroll, "Mesoscopic model for the fluctuating hydrodynamics of binary and ternary mixtures," *EPL (Europhysics Letters)*, vol. 80, no. 4, p. 40010, 2007.

[4] R. Winkler, K. Mussawisade, M. Ripoll, and G. Gompper, "Rod-like colloids and polymers in shear flow: a multi-particle-collision dynamics study," *Journal of Physics: Condensed Matter*, vol. 16, no. 38, p. S3941, 2004.

[5] S. H. Lee and R. Kapral, "Mesoscopic description of solvent effects on polymer dynamics," *The Journal of chemical physics*, vol. 124, no. 21, pp. 214 901–214 901, 2006.

[6] E. Tüzel, *Particle-based mesoscale modeling of flow and transport in complex fluids*. Ph.D. Thesis, University of Minnesota, 2006.

[7] H. Noguchi and G. Gompper, "Vesicle dynamics in shear and capillary flows," *Journal of Physics: Condensed Matter*, vol. 17, no. 45, p. S3439, 2005.

[8] ——, "Dynamics of fluid vesicles in shear flow: Effect of membrane viscosity and thermal fluctuations," *Physical Review E*, vol. 72, no. 1, p. 011901, 2005.

[9] J. Elgeti, U. B. Kaupp, and G. Gompper, "Hydrodynamics of sperm cells near surfaces," *Biophysical journal*, vol. 99, no. 4, pp. 1018–1026, 2010.

[10] Y. Yang, J. Elgeti, and G. Gompper, "Cooperation of sperm in two dimensions: synchronization, attraction, and aggregation through hydrodynamic interactions," *Physical Review E*, vol. 78, no. 6, p. 061903, 2008.

[11] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.

[12] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "Gpgpu processing in cuda architecture," *CoRR*, vol. abs/1202.4347, 2012.

[13] W. B. Langdon, "Debugging cuda," in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, ser. GECCO '11. New York, NY, USA: ACM, 2011, pp. 415–422. [Online]. Available: http://doi.acm.org/10.1145/2001858.2002028

[14] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[15] W. B. Langdon, "Performing with cuda," in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, ser. GECCO '11. New York, NY, USA: ACM, 2011, pp. 423–430. [Online]. Available: http://doi.acm.org/10.1145/2001858.2002029

[16] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Accelerating fully homomorphic encryption using gpu," in *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*. IEEE, 2012, pp. 1–5.

[17] E. Tüzel, T. Ihle, and D. Kroll, "Dynamic correlations in stochastic rotation dynamics," *Physical Review E*, vol. 74, no. 5, 2006. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.74.056702

[18] E. Tüzel, M. Strauss, T. Ihle, and D. Kroll, "Transport coefficients for stochastic rotation dynamics in three dimensions," *Physical Review E*, vol. 68, no. 3, 2003. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.68.036701

[19] T. Ihle, E. Tüzel, and D. M. Kroll, "Equilibrium calculation of transport coefficients for a fluid-particle model," *Physical Review E*, vol. 72, no. 4, p. 046707, 2005.

[20] C. Pooley and J. Yeomans, "Kinetic theory derivation of the transport coefficients of stochastic rotation dynamics," *The Journal of Physical Chemistry B*, vol. 109, no. 14, pp. 6505–6513, 2005.

[21] E. Falck, J. Lahtinen, I. Vattulainen, and T. Ala-Nissila, "Influence of hydrodynamics on many-particle diffusion in 2d colloidal suspensions," *The European Physical Journal E*, vol. 13, no. 3, pp. 267–275, 2004.

[22] J. Padding and A. Louis, "Hydrodynamic interactions and brownian forces in colloidal suspensions: Coarse-graining over time and length scales," *Physical Review E*, vol. 74, no. 3, p. 031402, 2006.

[23] H. Noguchi, G. Gompper, L. Schmid, A. Wixforth, and T. Franke, "Dynamics of fluid vesicles in flow through structured microchannels," *EPL (Europhysics Letters)*, vol. 89, no. 2, p. 28002, 2010.

[24] D. A. Reid, H. Hildenbrandt, J. Padding, and C. Hemelrijk, "Flow around fishlike shapes studied using multiparticle collision dynamics," *Physical Review E*, vol. 79, no. 4, p. 046313, 2009.

[25] T. Ihle and D. Kroll, "Stochastic rotation dynamics: A galilean-invariant mesoscopic model for fluid flow," *Physical Review E*, vol. 63, no. 2, 2001. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.63.020201

[26] A. Malevanets and J. Yeomans, "Dynamics of short polymer chains in solution," *EPL (Europhysics Letters)*, vol. 52, no. 2, p. 231, 2000.

[27] J. E. Lennard-Jones, "On the Determination of Molecular Fields. II. From the Equation of State of a Gas," *Royal Society of London Proceedings Series A*, vol. 106, pp. 463–477, Oct. 1924.

[28] *CUDA C PROGRAMMING GUIDE*, 5th ed., nVIDIA, October 2012.