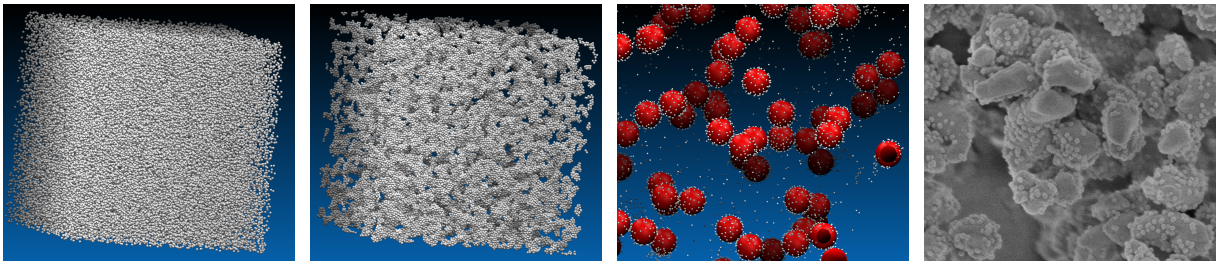
Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2015)
F. Jaillet, G. Zachmann, and F. Zara (Editors)

Brownian dynamics simulation on the GPU: virtual colloidal suspensions

Công Tâm Tran[1,2], Benoît Crespin[1], Manuella Cerbelaud[2] and Arnaud Videcoq[2]

[1]Univ. Limoges, CNRS, XLIM/DMI, UMR 7252, F-87000 Limoges, France
[2]Univ. Limoges, CNRS, ENSCI, SPCTS, UMR 7315, F-87000 Limoges, France



(a) (b) (c) (d)

Figure 1: *(a) Initial state of a 60K virtual colloidal suspension (b) Brownian Dynamics simulation of aggregation (occurs within approximately 1 second) (c) Virtual colloidal suspension with a bimodal size distribution (d) Microscopic image of a real colloidal suspension where alumina and silica particles aggregate*

Abstract

Brownian Dynamics simulations are frequently used to describe and study the motion and aggregation of colloidal particles, in the field of soft matter and material science. In this paper, we focus on the problem of neighbourhood search to accelerate computations on a single GPU. Our approach for one kind of particle outperforms existing implementations by introducing a novel dynamic test. For bimodal size distributions we also introduce a new algorithm that separates computations for large and small particles, in order to avoid additional friction that is known to restrict diffusive displacements.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.3.8 [Computer Graphics]: Applications—I.6.8 [Simulation and modeling]: Types of Simulation—Parallel J.2 [Computer Applications]: Physical Sciences and Engineering—Chemistry

1. Introduction

The Brownian motion is widely observed in nature. It describes the random motion of small particles suspended in a fluid, such as smoke particles in the air or colloids in liquids. Colloidal suspensions, *i.e.* small solid particles suspended in a liquid, are employed in very different fields such as the food industry, biological analyses or industrial processes. For example, they are widely used in ceramic shaping processes (casting, ink-jet printing, stereolithography, etc.) to produce parts of complex geometry and good quality. In this area, it is important to control the suspension behaviour. Indeed, each process needs specific rheological properties and the particle arrangement in suspension greatly influences the final properties (electrical, thermal, mechanical and optical) of the ceramic parts. Since a lot of parameters such as pH, ionic strength or external fields influence this behaviour, numerical simulations constitute a very useful tool to investigate and discriminate the role of each parameter. The

DOI: 10.2312/vriphys.20151332

simplest simulation method to study colloidal suspensions is Brownian Dynamics (BD) [AT87]. In this technique, the fluid is represented as a continuum medium and its effect upon the colloids is represented by a combination of frictional ($\Xi_i(t)$) and random forces (Γ_i). The motion of a colloid i is described by the Langevin equation, where v_i is its velocity, m_i its mass, t the time, r_{ij} the distance between colloids i and j, and F_{ij} the interaction force between i and j:

$$m_i \frac{dv_i(t)}{dt} = \sum_j F_{ij}(r_{ij}(t)) + \Xi_i(t) + \Gamma_i(t) \qquad (1)$$

By integrating this equation, BD simulations provide the temporal evolution of the colloid positions in the suspension. This colloidal motion virtually reproduced is very useful to understand the aggregation process represented in Figure 1a and 1b, especially when experimental characterizations are difficult or impossible to perform. BD simulations have been successfully used to study different suspensions [KHB*03, CVA*08, PVR*10]. One of the main drawbacks of the BD technique is its crude description of the hydrodynamics. As a consequence, it is not appropriate to study concentrated or sheared suspensions. An other consequence is that BD fails to properly simulate colloidal suspensions with a bimodal size distribution of colloids characterized by two very different sizes, as shown on Figures 1c and 1d. The diffusivity of a large particle surrounded by tenths of small ones is underestimated in an unphysical manner [CVAF09]. This is because the BD technique adds all the particle frictions, which lead to a caging effect. While it is possible to better describe the hydrodynamics in the BD simulations, using for instance the Rotne-Prager diffusion tensor [JBTK99], this dramatically increases the computation time.

The present paper deals with a GPU implementation of BD and a novel algorithm capable of efficiently eliminating this unphysical caging effect. Most of the problems encountered are related to more general issues found in particle-based dynamic simulation and physical interaction animation. Particle systems have been made popular in computer graphics by the early works of Reeves [Ree83], and particle-based representations are now largely used in many different fields ranging from autonomous agents to neural networks [KE95]. A good example is fluid simulations, where particles can be used to represent independent volumes of fluid with constant mass that interact with each other and generate realistic motion [IOS*14]. Such simulations have to cope with problems similar to those encountered with BD, where the main bottleneck is usually neighborhood search. This step of the simulation allows to determine the set of neighbouring particles that are close to a given particle, in order to compute their interactions and hence its motion.

As this step is repeated for all particles and for all time steps, different strategies and data structures were investigated over the years, preferably with efficient implementations on parallel architectures such as modern GPUs. Most particle-based fluid simulations rely on uniform grids (also called cell lists), where particles are spatially distributed in regular cells subdividing the simulation domain. With an appropriate size of cells, querying neighbours of a given particle only requires to check in the cell of this particle and in the neighbouring cells. On the other hand, BD implementations prefer another approach called neighbour lists (or *Verlet* lists), which store the neighbours of each particle [AT87]. These lists are updated every n_i-th time step. This value needs to be set carefully by the user: the larger the value of n_i, the better the efficiency of this approach over uniform grids which are updated at each time step. However, if n_i is too large a lot of unnecessary neighbours will be considered for distance computations. Recent GPU implementations allow simulations to run on relatively cheap hardware when compared to massively parallel supercomputers with high-level computational capacity. Neighborhood search is usually performed through a combination of neighbour lists which avoids GPU/CPU transfers and improves memory coherence, with uniform grids for updates.

This paper describes first a hybrid neighbour list / uniform grid approach for BD simulations with one type of particles. The main idea is to replace the conservative strategy for neighbour list updates by an on-the-fly approach where the need for updates is tested at each time step. We show that our implementation results in about 20% speedup as compared to previous works. We then extend our approach to two types of particles with very different sizes, and address the aforementioned caging problem.

2. Related works

2.1. Brownian dynamics

Brownian dynamics simulations have been developed in the 70's. One of the most famous algorithms was developed by Ermark [EM78]. By expressing the frictional term by the Stokes law ($\Xi_i(t) = -\zeta_i v_i$ with $\zeta_i = 6\pi\eta a_i$, where η is the solvent viscosity and a_i the radius of colloid) and by considering a time step larger than the velocity relaxation time of the colloid ($\tau_v = m_i/\zeta_i$), Equation 1 can be simplified as follows:

$$\frac{dr_i(t)}{dt} = \frac{1}{\zeta_i}\sum_j F_{ij}(r_{ij}(t)) + \frac{1}{\zeta_i}\Gamma_i(t), \qquad (2)$$

where $r_i(t)$ is the position of a particle over time. By integrating this equation with a white noise algorithm [MP89], the following iterative equation is obtained:

$$r_i(t+\delta t) = r_i(t) + \sqrt{\frac{2k_BT}{\zeta_i}}(\delta t)^{1/2}Y_i + \frac{1}{\zeta_i}\sum_j F_{ij}(r_{ij}(t))\delta t, \qquad (3)$$

with δt the time step, k_B the Boltzmann constant, T the temperature and Y_i uncorrelated Gaussian-distributed random numbers with an average of zero and a standard deviation of

1. When the time step is lower than the velocity relaxation time of the colloid the full Equation 1 is integrated:

$$v_i(t+\delta t) = v_i(t) + \frac{\sqrt{2k_BT\zeta_i}}{m_i}(\delta t)^{1/2}Y_i + \frac{1}{m_i}\left(-\zeta_i v_i(t) + \sum_j F_{ij}(r_{ij}(t))\right)\delta t, \quad (4)$$

$$r_i(t+\delta t) = r_i(t) + v_i(t)\delta t. \quad (5)$$

These iterative equations are used in the so-called *Langevin Dynamics* simulations.

When the colloids aggregate in the simulation, the aggregation kinetics can be followed by counting the number of aggregates as a function of time [TVC*13]. If we define an aggregate as an assembly of at least two colloids and if we start from a dispersed system, the number of aggregates first increases and then decreases. The first stage corresponds to a phase where dimer formation is predominant (two colloids encounter and aggregate) and the second stage corresponds to a phase where aggregates coalescence is predominant. The final stage of the aggregation process is reached when only one aggregate remains in the simulation.

2.2. Neighborhood search

Brownian and Langevin Dynamics are a particular case of *molecular dynamics simulations* (MD). Plimpton [Pli95] describes different parallel algorithms implemented on supercomputers. The best results were achieved when each processor is assigned a fixed spatial region, compared to other techniques (one thread per particle, one thread per interaction force). This pioneer work was implemented into the LAMMPS software [Pli95, Laa] and prefigured recent GPGPU techniques for MD simulations.

Several works appear in the literature following the emergence of GPUs with large computational capacities. In 2008, different implementations of molecular dynamics simulations using CUDA were presented [ALT08, LSVMW08]. These techniques both rely on Verlet lists and outperform parallel versions of the LAMMPS software running on a distributed memory cluster. Rovigatti *et al* [RvRR15] described two variants of one-thread-per-particle and one-thread-per-interaction-force approaches called "vertex-based" and "edge-based", respectively. Authors show that the "edge-based" technique becomes the most efficient choice for complicated interactions or a limited number of them between particles, contradicting the results presented in [Pli95]. There seems to be a consensus in recent approaches to combine Verlet lists with a uniform grid to improve shared memory parallelism and efficiency, as presented in [Gon12] and [PSC13]. Finally, a hybrid CUDA-MPI implementation distributed on 1024 GPUs is presented by Tang and Karniadakis [TK13] for large-scale simulations, which achieves speedups by a factor of 10-30 for a 128-million-particle system. Again, a grid is used to facilitate the construction of neighbor lists. These techniques are implemented in several versatile parallelized software developed for MD simulations with hundreds to millions of particles, such as Gromacs [BvdSvD95, VDSLH*05], HOOMD [SHUS10] and LAMMPS.

Fluid simulations also rely on particles to animate smoke, liquids or more complex fluids. One of the most well known models, Smoothed Particle Hydrodynamics (or SPH), uses interpolation kernels presented by Monaghan [Mon92] then Müller *et al* [MCG03]. As with MD, a critical problem in such simulations concerns neighborhood search. Hierarchical tree structures such as KD- or BSP-trees are commonly used for CPU-based implementations, but also collision detection and many other applications [EW82]. Multi-resolution simulations with variable kernel support (hence variable interaction radii) also rely on hierarchical spatial structures to improve efficiency [APKG07].

Neighbour lists, frequently used in MD, could be another option for fluid simulations but they would suffer from particles neighborhoods changing rapidly at each iteration. A uniform grid is usually preferred in parallel approaches because it fits better to the streaming architecture of modern GPUs or multi-core CPUs [IABT11]. If the size of grid cells is chosen to be equal to the maximal size of interpolation kernels, neighborhood search for a given particle means that only the cell of this particle and its adjacent cells have to be queried. Different strategies can be used for the parallel construction of uniform grids, for example index or Z-index sort [IOS*14]. These approaches align neighbouring particles into memory, thus improving memory coherence between concurrent threads. Hash functions can be used to compute index values for sorting [THM*03]. A typical hash function will generate a unique hash key for all particles located in the same cell:

$$hashkey = (p_1\left\lfloor\frac{x}{c}\right\rfloor \oplus p_2\left\lfloor\frac{y}{c}\right\rfloor \oplus p_3\left\lfloor\frac{z}{c}\right\rfloor) \mod n_{grid} \quad (6)$$

where (x,y,z) is the position of a particle, c is the size of a cell, p_1, p_2 and p_3 are large prime numbers, and n_{grid} is the number of cells in the grid. This rather inexpensive calculation allows to recompute the grid at each time step without degrading performances [IOS*14].

3. Brownian dynamics simulations with one kind of particle

In this section we focus on BD simulations with only one kind of particle, which means that all particles share the same characteristics (mass, radius, etc). Our approach implements a combination of neighbour lists with a uniform grid, with a noticeable difference with previous approaches: instead of using a fixed number of iterations to update neighbour lists, we rely on a test performed at each iteration. Our

results show that this approach outperforms existing methods and makes our implementation competitive compared to existing BD software.

Our simulations are developed on the GPU for a simple aqueous suspension only composed of alumina particles of diameter $d_A = 600\,\text{nm}$. Interactions between particles are modeled by an attractive short-range generalized Lennard Jones potential $V_{ij}(r)$ (see the details in Annex). The simulation is carried out using Equation 3 with a time step of $\delta t = 3.685 \times 10^{-7}$ s, which is equivalent to more than 2.5M iterations per second. The *density* ϕ (also called "volume fraction") of the BD simulation is the ratio of the total volume of particles over the volume of the simulation box.

The *cut-off radius* R_C represents the maximal interaction radius. If a uniform grid is implemented to accelerate neighbourhood search, the cell size is usually chosen equal to R_C.

If neighbourhood search relies on neighbors lists, additional parameters are needed as shown in Figure 2. The *list sphere radius* R_L ($R_L > R_C$) represents the maximal distance below which particles must be considered as potential neighbours. This parameter can be set freely by the user, but its value is of great importance when considering how many neighbour list updates will be necessary [AT87]. If $R_L \simeq R_C$, then potential neighbours change very rapidly and neighbour lists must be recomputed at almost each iteration. Inversely, if $R_L >> R_C$ then potential neighbours do not change too often but their number is very high and induces high memory costs and unnecessary computations, because many potential neighbours will have to be tested although their distance is much larger than R_C. One way to efficiently compute R_L is derived from Equation 3:

$$R_L = R_C + 2 v_{avg} n_i \delta t \ , \ v_{avg} = \sqrt{\frac{3 k_B T}{m_i}} \qquad (7)$$

where n_i is a fixed number of iterations and v_{avg} is the average velocity of a particle during one iteration of the simulation. If we let $R_{shell} = R_L - R_C$, then it is safe to assume that neighbour lists must be recomputed after n_i iterations, because in the meantime particles can travel a distance up to $R_{shell}/2$. This scenario is illustrated in Figure 2 with particle 2. The same idea is used by most existing GPU-based MD implementations described in the previous section. Again we must choose a value for n_i that will not induce too frequent updates nor unnecessary distance computations.

The main problem is illustrated in Figure 2: since particles have a Brownian motion, worst-case scenarios where a particle moves straightforward during several iterations almost never happen. In average, particles will spend much more than n_i iterations before the covered distance equals $R_{shell}/2$. In non-parallel implementations, updating neighbour lists only occurs if the sum of the two maximal displacements during an iteration is above R_{shell} [AT87]. We verified this idea on a 1M particles simulation by computing the number of iterations needed to travel the distance $R_{shell}/2$ depending on different values of n_i. The results are summarized on Table 1: we can conclude from this experiment that the "real" average number of iterations is far higher, especially for large values of n_i.

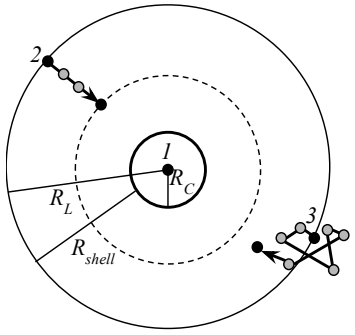


Figure 2: *A particle* (1) *with an initially empty neighbour list within radius* R_L*. If particles* 1 *and* 2 *travel the distance* $R_{shell}/2$ *towards each other, then their neighbour lists must be recomputed since they would lie within interaction range* R_C*. However, BD particles will likely spend many iterations moving around their initial position before covering distance* $R_{shell}/2$*, as shown with particle* 3*.*

	Iterations needed to reach $R_{shell}/2$			
n_i	Min	Max	Avg	Avg comp. time
5	0	9	7.22	22.23 ms
8	2	25	17.8	14.9 ms
11	16	52	41.5	12.7 ms
12	18	66	51.0	12.5 ms
13	37	77	61.3	12.5 ms
20	107	209	161	12.8 ms
30	187	384	323.9	13.3 ms

Table 1: *Summary of our experiments with a variable number of iterations* n_i *and the "real" number of iterations needed to reach the corresponding value* $R_{shell}/2$ *(1M particles,* ϕ *=15%). The final column shows the average computation time for an iteration which helps to select the most appropriate value for* n_i *(12 in this case).*

3.1. Hybrid Neighbor / Cell approach for neighbourhood search

Our implementation relies on a hybrid approach that combines neighbour lists and uniform grids. Following the experiments presented in Table 1, the main difference with previous approaches is that we check if neighbour lists need to be recomputed. This test is performed at each step because the displacement of a particle is not bounded. Such a strategy may seem time consuming at first glance: instead of an automatic update every n_i iterations, we must compute the

distance between the current and stored locations of each particle at each time step. However, our approach is not only rather inexpensive on the GPU, but it also reduces memory consumption by limiting the number of potential neighbors.

Algorithm 1 Brownian simulation with one kind of particle

```
1:  updateNeighbourLists = true
2:  for all iteration do
3:      if updateNeighbourLists = true then
4:          sort particles by hash key
5:          for all particle i do          // parallel kernel
6:              previousPos_i = position_i
7:              update NeighbourList_i
8:      updateNeighbourLists = false
9:      for all particle i do          // parallel kernel
10:         sumForces_i = 0
11:         for all particle j ∈ NeighbourList_i do
12:             if dist(position_i, position_j) < R_C then
13:                 update sumForces_i
14:         update position_i
15:         if dist(position_i, previousPos_i) > R_shell/2 then
16:             updateNeighbourLists = true
```

Algorithm 1 summarizes our method, where neighbour lists update is performed only if necessary. In this first step, a uniform grid is used along with Equation 6 to compute hash keys and sort particles. A parallel bitonic sort usually achieves the best result, as in most GPU-based fluid simulations [IOS*14]. Updating neighbour lists is also implemented in a parallel kernel, a thread group being launched for each cell in the grid. This results in high cache-hit rates due to neighboring particles being closer in GPU memory and hence faster computations. The final step updates the positions of each particle by checking all potential neighbours within range R_L stored in the particle's neighbour list. The kernel also checks whether neighbour lists should be updated before the next iteration. One could consider the use of a bitonic sort at each iteration in order to maintain the highest cache-hit rates in GPU memory. However sorting particles is one of the most expensive steps, as shown in Table 2. On this simulation, the sorting step appears very expensive but its global impact is limited to 5% as it is not executed at each time step. It should be noted that the Brownian, rather local motion of particles also limits the displacements between neighbouring cells in the uniform grid, and hence memory divergence.

The maximal size of each neighbour list NL_{max} is computed by considering the maximal number of particles that could fit into a sphere of radius R_L using a honeycomb distribution. Memory can then be allocated for neighbouring particles to all fit into one neighbour list. This maximal size may seem too large compared to the average number of neighbours during the simulation, resulting in wasted memory. On the other hand, a re-allocation scheme implemented on the GPU would also be computationally intensive if we were to apply a less conservative approach.

	Avg time	Iterations	Ratio
computeSumForce	7.15 ms	1	57.7%
updatePosition	3.6 ms	1	29.2%
updateTest	0.4 ms	1	3.2%
bitonicSort	29.5 ms	51	4.6%
updateNeighbours	33.5 ms	51	5.3%

Table 2: *Average time of computation for the different steps of Algorithm 1 and global computation ratio (1M particles, ϕ =15%). The third column shows the average number of iterations before the corresponding step is effectively computed.*

However, even with our approach there is actually no good way to automatically find the best value for R_L. The last column on Table 1 shows the impact of parameter n_i on the average computation time for an iteration. An optimum is obtained here for $n_i = 12$ or 13. In this case we choose $n_i = 12$ because this further reduces memory consumption, which also explains why updates occur each 51^{th} steps in Table 2. For lower values of n_i, computation times increase because updating neighbour lists occur too often. For higher values, too many distance computations are performed and more memory is requested but also wasted. In the general case the optimal value for n_i can be determined manually by running simulations with the same density but a lower number of particles, since the average number of neighbours remains approximately the same.

3.2. Implementation details

The global data structure in our implementation relies on several buffers stored in global GPU memory:

- positions, previous positions, force vectors and hash keys (3 `double4` and one `uint` per particle)
- neighbour lists (NL_{max} `uint` per particle)
- state vector for random Gaussian generation (100 `float` per thread)
- various buffers used for bitonic sort

The total size for a typical simulation with 1M particles is approximately 300 MB. This could be enhanced by considering single precision only, at least for force computations as in several MD softwares [VDSLH*05].

Our pseudo-random number generator (PRNG) requires a *state vector* of 100 floating-point values that is updated after each call to generate the next random number. The kernel implements a scheme called one-PRNG-per-thread which gives acceptable results for Brownian dynamics [PAG11] by limiting memory consumption and atomic calls. Still, random number generation accounts for almost all the compu-

tation time needed for the *updatePosition* step in Table 2 and could be further optimized.

3.3. Results

The curves shown in Figure 3 describe how our BD simulation behave with various densities [TVC*13]. As expected the number of aggregates grows rapidly until all particles have at least one close neighbour, then it falls down until only one main aggregate remains. Density plays a significant role in this process: the smaller the simulation box, the faster the convergence of the curve towards 1. The video showing the evolution of 60K particles with $\phi = 15\%$ from Figure 1a to 1b can be seen at: http://vimeo.com/133039225

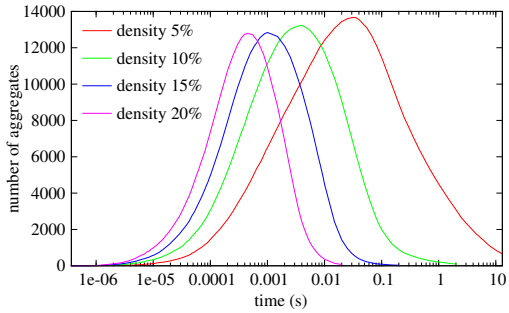


Figure 3: *Evolution of the number of aggregates during 10s of our BD simulation on a logarithmic scale with 60K particles with various densities. An aggregate is composed of two particles at least.*

Our method was implemented with OpenCL and tested on two NVIDIA GPUs. The Tesla K20m is a high-end professional graphics card (costs approx. 2,500$) with 13 physical cores and 5 GB of memory. The GTX690 is twice cheaper (approx. 1000$), with 8 cores and 4 GB of memory. Figure 4 demonstrates the scalability of our approach, depending on the density of the simulation. Variations in computation times can be related to the results shown in Figure 3 since higher densities induce a faster convergence towards a stationary state with limited particles displacements.

Table 3 describes how our neighbourhood search approach outperforms existing strategies used in molecular or fluid dynamics. In a 1M particles simulation, the worst results appear when using a uniform grid because of the bitonic sort step computed at each iteration. The second approach uses a larger size R_L, but also a conservative test to check whether the grid should be reordered. In this case low performances are related to the excessive number of potential neighbours that must be considered. With the approach described at the beginning of Section 3, which combines static neighbour lists and a uniform grid, computation time is divided by 2. Our approach further reduces this time by almost 25% by introducing a dynamic conservative test.

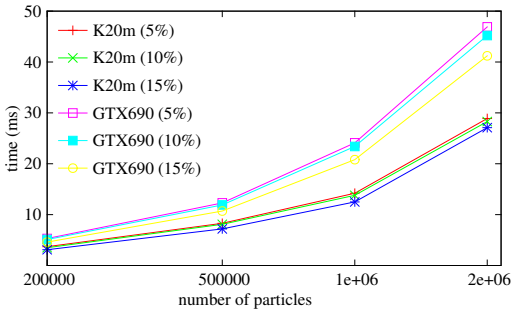


Figure 4: *Average computation time per iteration obtained with our approach for an increasing number of particles and various densities. Tests were conducted on two NVIDIA cards, K20m (2013) and GTX690 (2012)*

Neighbourhood search	Avg comp. time
Uniform grid with cell size = R_C	58.5 ms
Uniform grid with cell size = R_L	33.6 ms
Hybrid approach (static update)	16.5 ms
Our approach	12.5 ms

Table 3: *Average computation time for an iteration with various neighbourhood search strategies (1M particles, $\phi = 15\%$)*

We compared our results with an implementation of Gromacs [VDSLH*05] (v5.02) which is the only software available for such BD simulations. The tests ran on two Intel Xeon E5-2650-v2 processors with 8 cores each and 8 GB memory per core. Although it is not a GPU-based architecture, we consider this configuration to be a good equivalent to our K20m hardware with 3 more cores available and a lot more memory (but approximately the same price). The experiments were conducted with $n_i = 10$ in single precision, and a pre-computed Lennard Jones potential to accelerate computations. Even with such improvements, our approach is approximately 15% faster (*e.g.* 14.83ms per iteration for 1M particles were obtained with Gromacs).

4. Brownian dynamics simulations with two kinds of particles

In this section we seek to study more complex and realistic suspensions composed of two kinds of particles, which have a large size ratio (1/16) and where *hetero-aggregation* occurs. As a model, we use the suspensions described in [CVA*08, CVAF09] composed of large alumina particles with a radius $a_A = 200\,\text{nm}$ and small silica particles with a radius $a_S = 12.5\,\text{nm}$. The alumina particles are positively charged and silica particles negatively charged. The interactions between particles are described by a DLVO potential [EGJW95], noted $V^{DLVO}(r)$, which represents the interactions between oxide particles in suspension (see Annex).

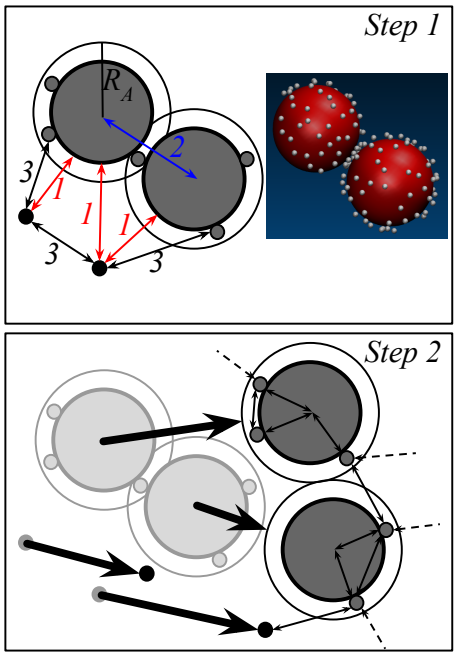


Figure 5: *Computation of interaction forces between large alumina and small silica particles. Top: computation of interaction forces between (1) isolated silica and alumina shown in red, (2) alumina-alumina in blue and (3) silica-silica in black. Bottom: after all isolated silica and Silica-Alumina Complexes (SACs) are displaced, interaction forces between adsorbed silica and all other particles are computed.*

Particles of the same kind tend to repel each other, but aggregation still occurs rapidly due to the attractive force between silica and alumina. Such simulations result in aggregates that appear visually close to microscopic observations as shown in Figures 1c and 1d with 82 silica for one alumina particle.

4.1. Naive approach

The velocity relaxation time of the silica particles is calculated as $\tau_v = 7.63 \times 10^{-11}$ s and for the alumina particles as $\tau_v = 3.53 \times 10^{-8}$ s. For BD simulations, the time step must be larger than both these values, *e.g.* $\delta t = 10^{-7}$s. The problem with such a large time step is that interaction forces change significantly for the silica particles, which does not allow to describe their motion properly. To avoid this effect, the time step is usually fixed between relaxation times at $\delta t = 5 \times 10^{-10}$ s. The motion of alumina is described by Langevin dynamics (Equations 4 and 5) and the motion of silica by Brownian dynamics (Equation 3). The values of cut-off and neighbour list radii (R_C and R_L respectively) now depend on the type of interaction: alumina-alumina, alumina-silica and silica-silica.

Simulations combining Brownian and Langevin dynamics are actually known to show that first, silica particles *adsorb* at the surface of alumina particles, and then silica-covered alumina particles aggregate via the adsorbed silica which lie between them as shown in Figure 5 [CVA*08]. The *adsorption* radius R_A is usually set to $1.05(a_A + a_S)$.

However, with a naive approach which does not take hydrodynamics into account the motion of an alumina particle is reduced by the silica covering it [CVAF09]. The diffusion coefficient of an isolated alumina particle is $D_0 = 1.07 \times 10^{-12}$ m^{2}s^{-1}, whereas it drops to $D = 1.66 \times 10^{-13}$ m^{2}s^{-1} when 82 silica particles are adsorbed at the surface. This phenomenon, also called caging effect, is explained by the fact that BD techniques add all the particle frictions together. To remove this artifact, hydrodynamic effects can be inserted in the simulation by using for example the Yamakawa-Rotne-Prager tensor [JBTK99], but at the cost of extra computations. This would also make most existing BD software impossible to use, unless they run on high-performance supercomputers [TK13].

Our method described in the next section addresses this issue by introducing a novel approach.

4.2. New scheme of simulation

The core of our method is to modify the scheme of simulation by considering each silica-covered alumina particle as a single entity called Silica-Alumina Complex (SAC). A silica and an alumina belong to the same SAC if their distance is below the adsorption radius R_A. SACs can move as a whole in a first step using Langevin dynamics, without considering any interaction inside R_A. The own motion of adsorbed silica around an alumina inside each SAC has to be computed in a second step.

A summary of our approach is presented in Figure 5. In the first step (top draw), interactions forces are computed between: (1) alumina and isolated silica, (2) alumina and alumina, and (3) silica and silica if at least one of them is isolated. Then all isolated silica and SACs are displaced. The second step (bottom draw) focuses on the computation of the remaining interaction forces between adsorbed silica and all other particles. Displacements of adsorbed silica are then applied. This motion decomposition prevents additional friction from restricting the displacements of alumina particles, without computing hydrodynamic effects.

Our method uses three types of neighbour lists and three uniform grids for neighbourhood search, one for each interaction force. The maximal number of neighbours and the size of grid cells is given by the corresponding R_L value. The "silica-silica" grid (resp. "alumina-alumina") stores silica particles (resp. alumina). Accelerating "alumina-silica"

interactions is only interesting when a large alumina particle looks for small neighbouring silica. Hence only silica particles are stored in the "alumina-silica" grid and neighbour lists. This also allows an alumina to look for adsorbed silica within radius R_A efficiently in order to update the corresponding SAC. If an adsorbed silica lies within the R_A radius of multiple SACs, one of them is chosen arbitrarily. This is the case in Figure 5 for the silica particle located between the two alumina, which is arbitrarily associated to the left alumina in Step 1 and moves with the corresponding SAC.

Algorithm 2 Brownian/Langevin simulation with two kinds of particles

```
1:  for all iteration do
2:      // step 1
3:      for all interaction force do
4:          update neighbour lists if necessary
5:      update SACs composition if necessary
6:      for all interaction force do
7:          update sumForces for isolated silica
8:          update sumForces for SACs
9:      update SACs velocities and positions
10:     update isolated silica positions
11:     // step 2
12:     for all interaction force do
13:         update neighbour lists if necessary
14:     for all adsorbed silica do
15:         update sumForces
16:     update adsorbed silica positions
```

Algorithm 2 extends the BD case presented in Section 3 by separating computations. The parallel kernels used for updating neighbour lists, forces and positions are all derived from Algorithm 1. They are executed on all particles, except during step 2 where only adsorbed silica are considered. The use of Langevin dynamics for the motion of SACs also implies extra computations for SACs velocities using Equation 5. The main difference is that some calls must be repeated three times, one for each interaction force. Only the corresponding data structures (grids and neighbour lists) need to be cached into GPU memory during kernels execution, but the global number of kernel calls during an iteration is far more higher than with BD simulations.

4.3. Results and discussion

With a time step of $\delta t = 5 \times 10^{-10}$ s, 2 billions iterations must be computed for one second of simulation. In these conditions, an average time of 12.5ms per iteration (obtained with BD simulations) for 1M particles would mean more than 289 days of computation. Fortunately computation times can be reduced because the aggregation process is faster with smaller particles. The characteristic time when the number of aggregates starts to decrease occurs within 0.1s whereas it was 1s with the first system, but simulating 0.1s still means 200M iterations.

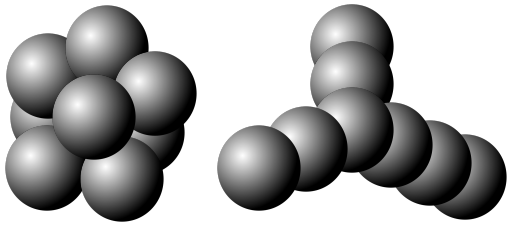


Figure 6: *Left: compact aggregate with 8 alumina particles and an average number of neighbours of* 7. *Right: linear aggregate with an average number of neighbours of* 1.75

Alumina	Silica	Time/iteration (ms)	Total (days)
200	2969	0.85	1.96
200	29699	2.92	6.76
300	44549	3.87	8.95

Table 4: *Average iteration and total time of computation for Brownian-Langevin simulations of 0.1s*

These observations led us to drastically reduce the total number of particles in order to achieve acceptable results. Table 4 shows the computation times obtained on a K20m NVIDIA GPU for a concentration of silica $R = 0.2$ (see Annex). Our method scales correctly with up to approximately 48K total particles, but the computation cost becomes prohibitive for larger simulations. Nevertheless, significant structures can be extracted by material science researchers who are more interested in statistical analysis than interactive visualization with a lower number of particles.

In order to compare the aggregation kinetics and the aggregate shapes with the first system, here only alumina particles are considered. A first analysis showed that the diffusion coefficient of isolated alumina particles was not affected by adsorbed silica, thus preventing any caging effect. This is confirmed by the evolution of the number of aggregates, which is similar to the results obtained for homo-aggregation in Figure 3. The shape of the aggregates themselves is significantly different. For unimodal size distributions, the average number of neighbours per aggregate at the end of the simulation is approximately 5, which corresponds to a rather compact shape. For bimodal size distributions, as more linear aggregates or "chains" are obtained with our simulations, this number drops towards 2, as illustrated in Figure 6. This effect can be observed on videos rendered using the VMD software [VMD] for bimodal size distributions at: `http://vimeo.com/133997636` and `http://vimeo.com/133997637`

Our approach still suffer from several limitations that should be addressed for other types of colloidal suspensions,

related to the magnitude of interaction forces between small and large particles. Small silica particles stay close to large alumina in our case, but with higher repulsion forces the radius of our SAC model would increase, along with computation times. The fact that silica particles can belong to only one SAC at a time could also be a problem in this case.

5. Conclusion

We have presented in this paper a novel approach to address the problem of neighbourhood search in the context of Molecular Dynamics simulations. Based on a combination of neighbourhood lists and a uniform grid, our method outperforms existing approaches for Brownian Dynamics by introducing a dynamic test at each iteration. We are able to run simulations with up to 2M particles on a single GPU, with computation times comparable to those obtained with well-known MD software like Gromacs.

For bimodal size distributions our implementation relies on the same neighbourhood search method and extends it to three types of interactions. We introduced a new way to separate computations for large and small particles, using the concept of Silica-Alumina Complexes. It provides a solution to the caging problem, *i.e.* additional friction limiting the displacements of large particles, occurring in Brownian-Langevin Dynamics. The time step defined by physical properties ($\delta t = 5 \times 10^{-10}$s) is the bottleneck of our computations and does not allow to run simulations with more than 50K particles on a single GPU. However, our results suggest that other types of colloidal suspensions found in food industry or biological analyses could benefit from our approach.

Future works will seek to improve the performances of our software for bimodal size distributions, which also correspond to important systems for material science researchers. The main goal is to find an implicit formulation for the motion of small silica particles. This would make it possible to use larger time steps and reduce computation times. We could also try to investigate how to extend the concept of Silica-Alumina Complexes to aggregates composed of several SACs. Their motion could be computed by considering them as single entities in a multi-phase algorithm extending Algorithm 2.

References

[ALT08] ANDERSON J. A., LORENZ C. D., TRAVESSET A.: General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics 227*, 10 (May 2008), 5342–5359. 3

[APKG07] ADAMS B., PAULY M., KEISER R., GUIBAS L. J.: Adaptively sampled particle fluids. *ACM Trans. Graph. 26*, 3 (July 2007). 3

[AT87] ALLEN M., TILDESLEY D.: *Computer simulation of Liquids*. Oxford University Press: Oxford, 1987. 2, 4

[BvdSvD95] BERENDSEN H., VAN DER SPOEL D., VAN DRUNEN R.: Gromacs: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications 91*, 1-3 (1995), 43 – 56. 3

[CVA*08] CERBELAUD M., VIDECOQ A., ABÉLARD P., PAGNOUX C., ROSSIGNOL F., FERRANDO R.: Heteroaggregation between Al_2O_3 submicrometer particles and SiO_2 nanoparticles: Experiment and simulation. *Langmuir 24*, 7 (2008), 3001–3008. 2, 6, 7

[CVAF09] CERBELAUD M., VIDECOQ A., ABÉLARD P., FERRANDO R.: Simulation of the heteroagglomeration between highly size-asymmetric ceramic particles. *Journal of Colloid and Interface Science 332*, 2 (2009), 360 – 365. 2, 6, 7

[EGJW95] ELIMELECH M., GREGORY J., JIA X., WILLIAMS R. (Eds.): *Particle Deposition and Aggregation*, oxford, england ed. Butterworth-Heinemann, 1995. 6

[EM78] ERMAK D. L., MCCAMMON J. A.: Brownian dynamics with hydrodynamic interactions. *The Journal of Chemical Physics 69*, 4 (1978), 1352–1360. 2

[EW82] EASTMAN C., WEISS S.: Tree structures for high dimensionality nearest neighbor searching. *Information Systems 7*, 2 (1982), 115 – 122. 3

[Gon12] GONNET P.: Pairwise verlet lists: Combining cell lists and verlet lists to improve memory locality and parallelism. *Journal of Computational Chemistry 33*, 1 (2012), 76–81. 3

[IABT11] IHMSEN M., AKINCI N., BECKER M., TESCHNER M.: A parallel sph implementation on multi-core cpus. *Computer Graphics Forum 30*, 1 (2011), 99–112. 3

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: Sph fluids in computer graphics. *Eurographics State-of-The-Art-Reports* (2014), 21–42. 2, 3, 5

[JBTK99] JARDAT M., BERNARD O., TURQ P., KNELLER G. R.: Transport coefficients of electrolyte solutions from smart brownian dynamics simulations. *The Journal of Chemical Physics 110*, 16 (1999), 7993–7999. 2, 7

[KE95] KENNEDY J., EBERHART R.: Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on* (Nov 1995), vol. 4, pp. 1942–1948 vol.4. 2

[KHB*03] KIM A. Y., HAUCH K. D., BERG J. C., MARTIN J. E., ANDERSON R. A.: Linear chains and chain-like fractals from electrostatic heteroaggregation. *Journal of Colloid and Interface Science 260*, 1 (2003), 149 – 159. 2

[Laa] Lammps molecular dynamics simulator. URL: `http://lammps.sandia.gov`. 3

[LSVMW08] LIU W., SCHMIDT B., VOSS G., MÜLLER-WITTIG W.: Accelerating molecular dynamics simulations using Graphics Processing Units with CUDA. *Computer Physics Communications 179*, 9 (Nov. 2008), 634–641. 3

[Lyk91] LYKLEMA J. (Ed.): *Fundamentals of interface and colloid science: Volume 1*, london ed. Academic Press, 1991. 10

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), SCA '03, pp. 154–159. 3

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30* (1992), 543–574. 3

[MP89] MANNELLA R., PALLESCHI V.: Fast and precise algorithm for computer simulation of stochastic differential equations. *Phys. Rev. A 40* (Sep 1989), 3381–3386. 2

[PAG11] PHILLIPS C. L., ANDERSON J. A., GLOTZER S. C.:

Pseudo-random number generation for Brownian Dynamics and Dissipative Particle Dynamics simulations on GPU devices. *Journal of Computational Physics* (June 2011). 5

[Pli95] PLIMPTON S.: Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics 117* (1995), 1–19. 3

[PSC13] PROCTOR A. J., STEVENS C. A., CHO S. S.: Gpu-optimized hybrid neighbor/cell list algorithm for coarse-grained md simulations of protein and rna folding and assembly. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics* (2013), BCB'13, pp. 633:633–633:640. 3

[PVR*10] PIECHOWIAK M. A., VIDECOQ A., ROSSIGNOL F., PAGNOUX C., CARRION C., CERBELAUD M., FERRANDO R.: Oppositely charged model ceramic colloids: Numerical predictions and experimental observations by confocal laser scanning microscopy. *Langmuir 26*, 15 (2010), 12540–12547. 2

[Ree83] REEVES W. T.: Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics* (1983). 2

[RvRR15] ROVIGATTI L., ŠULC P., REGULY I. Z., ROMANO F.: A comparison between parallelization approaches in molecular dynamics simulations on GPUs. *J. Comput. Chem. 36*, 1 (Jan. 2015), 1–8. 3

[SHUS10] STONE J. E., HARDY D. J., UFIMTSEV I. S., SCHULTEN K.: GPU-accelerated molecular modeling coming of age. *Journal of molecular graphics & modelling 29*, 2 (Sept. 2010), 116–125. 3

[THM*03] TESCHNER M., HEIDELBERGER B., MÃIJLLER M., POMERANTES D., GROSS M. H.: Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Vision Modeling and Visualization* (2003), pp. 47–54. 3

[TK13] TANG Y.-H., KARNIADAKIS G. E.: Accelerating Dissipative Particle Dynamics Simulations on GPUs: Algorithms, Numerics and Applications. *Computer Physics Communications 185*, 11 (Nov. 2013), 2809–2822. 3, 7

[TVC*13] TOMILOV A., VIDECOQ A., CERBELAUD M., PIECHOWIAK M. A., CHARTIER T., ALA-NISSILA T., BOCHICCHIO D., FERRANDO R.: Aggregation in colloidal suspensions: Evaluation of the role of hydrodynamic interactions by means of numerical simulations. *The Journal of Physical Chemistry B 117*, 46 (2013), 14509–14517. 3, 6

[VDSLH*05] VAN DER SPOEL D., LINDAHL E., HESS B., GROENHOF G., MARK A. E., BERENDSEN H. J. C.: Gromacs : fast, flexible, and free. *Journal of Computational Chemistry 26*, 16 (2005), 1701–1718. QC 20120301. 3, 5, 6

[VMD] Vmd: Visual molecular dynamics. URL: http://www.ks.uiuc.edu/Research/vmd/. 8

Annex 1: Brownian dynamics with one kind of particle

The generalized Lennard Jones potential $V_{ij}(r)$ is defined by:

$$V_{ij}(r) = 4\varepsilon_p k_B T \left[\left(\frac{d_A}{r_{ij}}\right)^{36} - \left(\frac{d_A}{r_{ij}}\right)^{18} \right] \text{ if } r_{ij} <= R_C,$$
$$V_{ij}(r) = 0 \text{ otherwise,} \tag{8}$$

with d_A the particle's diameter, R_C the *cut-off* radius and $\varepsilon_p = 14$. Simulations are performed in water at $T = 293\,\mathrm{K}$ ($\eta = 10^{-3}$ Pa.s). The velocity relaxation time of the alumina particles is calculated at $\tau = 8.5 \times 10^{-8}$ s.

Compositions	R=0.2%	R=1.1%
Number of silica for one alumina	15	82
Ψ_A (mV)	45	30
Ψ_S (mV)	-20	-26

Table 5: *Parameters used in simulations with alumina and silica particles.*

Particles lie inside a cubic simulation box with periodic boundary conditions, and its size is fixed by the density of particles (ϕ) and their number (n) as $L_{box} = \sqrt[3]{\frac{4\pi n d_A^3}{3\phi}}$

Annex 2: Brownian dynamics simulations with two kinds of particles

The DLVO potential (named after scientists Derjaguin, Landau, Verwey and Overbeek) is the sum of two contributions: a van der Waals potential V^{vdW} and an electrostatic potential V^{el} [Lyk91] due to the surface charges of colloids:

$$V^{DLVO}(r_{ij}) = V^{vdW}(r_{ij}) + V^{el}(r_{ij}), \tag{9}$$

where

$$V^{vdW}(r_{ij}) = -\frac{A_{ij}}{6}\left[\frac{2a_i a_j}{r_{ij}^2-(a_i+a_j)^2} + \frac{2a_i a_j}{r_{ij}^2-(a_i-a_j)^2} + \ln\left(\frac{r_{ij}^2-(a_i+a_j)^2}{r_{ij}^2-(a_i-a_j)^2}\right)\right] \tag{10}$$

and

$$V^{el}(r_{ij}) = \pi\varepsilon\frac{a_i a_j}{a_i+a_j}\left(\Psi_i^2+\Psi_j^2\right)\times \left[\frac{2\Psi_i\Psi_j}{\Psi_i^2+\Psi_j^2}\ln\left(\frac{1+\exp(-\kappa h_{ij})}{1-\exp(-\kappa h_{ij})}\right) + \ln(1-\exp(-2\kappa h_{ij}))\right] \tag{11}$$

with A_{ij} the constant of Hamaker ($A_{AA} = 4.76 \times 10^{-20}$ J, $A_{SS} = 4.6 \times 10^{-21}$ J and $A_{AS} = 1.48 \times 10^{-20}$ J for the alumina-alumina, silica-silica and alumina-silica interactions respectively), κ the inverse of Debye length ($\kappa = 10^8\,\mathrm{m}^{-1}$), ε the dielectric constant of water, Ψ_i the surface potential of i and h_{ij} the surface-to-surface separation distance. Two concentrations of silica quoted R are used, which give the parameters summarized in Table 5. Simulations are performed with a density $\phi = 0.03$.

Acknowledgements

The results presented in this paper were obtained with the CALI supercomputer at the University of Limoges, sponsored by the Limousin Region and institutes XLIM, IPAM and GEIST.