Computer Physics Communications 182 (2011) 2307–2313

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Rigid body constraints realized in massively-parallel molecular dynamics on graphics processing units

Trung Dac Nguyen [a], Carolyn L. Phillips [b], Joshua A. Anderson [a], Sharon C. Glotzer [a,c,*]

[a] Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109, USA
[b] Applied Physics Program, University of Michigan, Ann Arbor, MI 48109, USA
[c] Department of Materials Science and Engineering, University of Michigan, Ann Arbor, MI 48109, USA

ARTICLE INFO

Article history:
Received 25 February 2011
Accepted 11 June 2011
Available online 14 June 2011

Keywords:
GPU
GPGPU
CUDA
Molecular dynamics
Rigid body

ABSTRACT

Molecular dynamics (MD) methods compute the trajectory of a system of point particles in response to a potential function by numerically integrating Newton's equations of motion. Extending these basic methods with rigid body constraints enables composite particles with complex shapes such as anisotropic nanoparticles, grains, molecules, and rigid proteins to be modeled. Rigid body constraints are added to the GPU-accelerated MD package, HOOMD-blue, version 0.10.0. The software can now simulate systems of particles, rigid bodies, or mixed systems in microcanonical (NVE), canonical (NVT), and isothermal-isobaric (NPT) ensembles. It can also apply the FIRE energy minimization technique to these systems. In this paper, we detail the massively parallel scheme that implements these algorithms and discuss how our design is tuned for the maximum possible performance. Two different case studies are included to demonstrate the performance attained, patchy spheres and tethered nanorods. In typical cases, HOOMD-blue on a single GTX 480 executes 2.5–3.6 times faster than LAMMPS executing the same simulation on any number of CPU cores in parallel. Simulations with rigid bodies may now be run with larger systems and for longer time scales on a single workstation than was previously even possible on large clusters.

1. Introduction

Molecular dynamics (MD) simulations and related methods are powerful tools for modeling systems of particles [1]. The basic MD technique computes the trajectory of n particles under the influence of a potential $V(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n)$, the negative gradient of which gives a conservative force $\vec{F} = -\vec{\nabla} V$, by integrating Newton's equations of motion over discrete time steps that each advance the state of the system from $[\vec{r}_i(t), \vec{p}_i(t)]$ to $[\vec{r}_i(t + \Delta t), \vec{p}_i(t + \Delta t)]$. The quantities $\vec{r}_i$ and $\vec{p}_i$ are the position and momentum of the i-th particle, respectively, t is the current simulation time, and Δt is the step size. Many applications of MD, such as soft matter self-assembly [2–4] and protein folding [5–7], necessitate running hundreds of millions of time steps per run and thousands of individual runs. Accelerating the rate at which time steps are performed reduces the time to discovery and enables better predictions through the use of higher fidelity models.

Classical MD breaks the potential into pair-wise and bond terms $V = \sum_{\text{pairs } i,j} V_p(r_{ij}) + \sum_{\text{bonds } i,j} V_b(r_{ij})$. Smooth, "soft" potentials $V_p(r)$ and $V_b(r)$ can be used in conjunction with a large step size. On the other hand, steep, "hard" potentials, such as bonds with stiff spring constants, require using a prohibitively small step size to maintain accuracy and stability. Potentials with infinitely steep interaction terms can only be achieved with extensions to the basic MD framework.

One such extension is SHAKE [8]. The SHAKE algorithm enforces fixed bond distances between two particles. Via an iterative method, any number of bonds in the system can be constrained. A set of particles may be combined into a single rigid body with an appropriate choice of bond constraints while taking special care not to over-constrain the system. However, certain rigid shapes, such as planar and linear molecules, cannot be created in three dimensions by setting bond distances alone because the constraint matrix is singular. Although the SHAKE algorithm has been extended to handle arbitrary shapes, for example, via angle and dihedral angle constraints, [9,10] the computational cost of these algorithms often becomes prohibitive for parallel simulation codes as the number of constraints per cluster increases.

Modeling large or generic rigid arrangements of particles can also be achieved by treating each defined set of particles as a single rigid body with only three translational and three orientational degrees of freedom (or two and one, respectively, for 2D simulations) [11]. Such a method can be added to an MD package with minimal modifications by taking advantage of the existing code that computes particle–particle interactions. Rigid body constraints are available in MD software packages such as DLPOLY [12] and

* Corresponding author at: Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109, USA.
E-mail address: sglotzer@umich.edu (S.C. Glotzer).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2011.06.005

LAMMPS [13], and have been used to model cubes, rods, bent rods, jacks, plates, bumpy spheres, water molecules and ions, and buckyballs [2,14–20].

One problem that arises for both rigid body and SHAKE constraints is that they often cannot be used with complementary methods that would violate the constraints, such as energy minimization. Local energy minimization methods find a local minimum of the potential energy landscape given an initial configuration, in effect *quenching* the system. Such methods are used to relax initial configurations, find zero temperature equilibrium configurations of crystalline solids, native configurations of biomolecules, and low energy atom clusters [21], and can be an important step in protein folding algorithms [22].

One minimization technique commonly employed for unconstrained particles is the conjugate gradient (CG) method [23]. Applying this method to a system of rigid bodies may be possible, however we are unaware of any existing adaptation and do not attempt to derive one. Instead, we find that a straightforward application of the Fast Inertial Relaxation Engine (FIRE) [21] algorithm works for both unconstrained particles and rigid bodies while using the basic framework of the rigid body NVE dynamics method.

1.1. GPU overview

Modern graphics processing units (GPUs) deliver tremendous computational performance at a fraction of the cost and power consumption of a cluster of traditional CPUs. On a single silicon chip with three billion transistors, current GPU models provide a theoretical peak 1.3 teraflops of compute throughput and 177 gigabytes per second of memory bandwidth from off-chip, or global, device memory [24]. In our experience, it is easier to write a small application that achieves a higher fraction of the theoretical peak performance on a GPU than a CPU.

The GPU hardware is unlike a CPU in many ways. First and foremost, while a CPU core executes a single instruction at a time, a GPU executes hundreds. The processor chip on the NVIDIA® GeForce® GTX 480 (GF100), for example, contains 480 individual CUDA cores. Each core is capable of processing one single precision floating point or integer operation per clock tick. The CUDA cores on the GTX 480 are grouped into 15 multiprocessors (MPs), which perform instruction scheduling and are each capable of maintaining up to 1536 independent computation streams or *threads* in flight at any one time. Thus the GPU is only fully occupied when more than 23 thousand threads are executing on the device.

When launched, threads are grouped into *blocks*. Each thread can locally access its index within the block, `threadIdx`, the index of its block, `blockIdx`, and a small (up to 48 kb) pool of shared memory also available to the other threads in the same block. For global memory transactions, the GF100 includes a full cache hierarchy with up to 48 kilobytes of hardware managed L1 cache in each streaming multiprocessor, 768 kilobytes of shared L2 cache, and up to six gigabytes of device memory.

The performance of functions executed on the GPU, or *kernels*, can be limited by either the memory bandwidth between the processor and device memory, or the rate at which arithmetic instructions are retired. In most molecular dynamics applications, the bottleneck is the device memory bandwidth. Optimal performance is obtained in these cases by carefully minimizing the amount of memory accessed and by tuning the access pattern to maximize cache hits.

While the device memory bandwidth is fast, transfers between host memory (accessible by the CPU) and device memory are typically between two and six gigabytes per second, depending on the hardware configuration. Thus, in order to maximize overall application speed, transfers between the host and device must be avoided whenever possible.

1.2. MD on GPUs

The CUDA C programming environment, which was the first to enable truly general purpose computing on massively parallel GPUs, was released in 2007. GPU-accelerated MD methods were developed shortly thereafter [25–27]. HOOMD-blue [25,28] differs from most other GPU-accelerated MD methods in that it implements every step of the computation on the GPU and avoids all host/device transfers, except when needed for disk I/O. By avoiding both serial code bottlenecks and slow memory transfers between the host and device, HOOMD-blue reaches maximum performance on a single GPU. As of this publication, the most recent release version 0.9.1 deployed on a single GTX 480 performs at a speed equivalent to LAMMPS [13] parallelized over 50–90 CPU cores on a Xeon E5540 cluster with Infiniband for a wide range of molecular dynamics simulations.

HOOMD-blue, available under an open source license [28], implements the standard algorithms employed by classical MD frameworks. In each time step, the state of system is updated in $\Theta(N)$ time in a number of phases. First, the particles are (1) binned into a cell list. From this cell list (2) a neighbor list is constructed for each particle that contains the indices of all the particles within the specified interaction range. The neighbor list is consulted when (3) computing the pair forces between all interacting pairs of particles. Finally, (4) the computed forces are used to update the particles forward to the next time step. Each phase (1–4) consists of one or more kernels that are executed on the GPU, and all necessary data structures are stored in device memory [25,28]. Different versions of each phase can be interchanged to implement numerous force fields and ensembles, thereby enabling diverse simulation possibilities in a single code package.

In this paper, rigid body constraints are implemented in HOOMD-blue with the inclusion of new data structures and an additional version of phase 4 that updates the position, velocity, and orientation of defined rigid bodies. Rigid bodies are built out of particles so that the existing modules that compute particle–particle interactions (phases 1–3) may be used without modification. We present the algorithm in Section 2 and its implementation in HOOMD-blue in Section 3. We present validation and performance metrics in Section 4 and concluding remarks in Section 5.

2. Algorithm

First, the terminology of a system of rigid and non-rigid bodies is introduced. A system contains n particles, each of which may belong to one rigid body or none at all, for a total of N_{bodies} rigid bodies such that $N_{\text{bodies}} \leqslant n$. The center of mass and velocity in the *space frame* shall be indicated by lowercase $\vec{r}$ and $\vec{v}$ for a particle and uppercase $\vec{R}$ and $\vec{V}$ for a rigid body with appropriate subscript indices.

Consequently, each rigid body b is composed of N_b particles indexed by $B_{bk} = [B_{b1}, B_{b2}, \ldots, B_{bN_b}]$. The center of mass of body b is located at position $\vec{R}_b$, moving at a velocity $\vec{V}_b$. Body b has a mass M_b and moment of inertia $\mathbf{I}_b$. The orientational degrees of freedom include its angular momentum $\vec{L}_b$ and a normalized quaternion q_b representing its orientation. In the *body frame*, a body's center of mass is at the origin and $\mathbf{I}_b$ is diagonal.

Thus, the position and velocity of a particle in the space frame can be calculated as follows:

$$\vec{r}_{B_{bk}} = \vec{R}_b + \mathbf{R}(q_b) \cdot \vec{D}_{bk}, \tag{1}$$

$$\vec{v}_{B_{bk}} = \vec{V}_b + \vec{\omega}_b \times \left(\mathbf{R}(q_b) \cdot \vec{D}_{bk}\right), \tag{2}$$

where $\vec{D}_{bk}$ is a displacement vector that defines the position of the particle relative to the center of mass (COM) in the body frame and $\vec{\omega}_b = \mathbf{R}(q_b)\mathbf{I}_b^{-1}\mathbf{R}^T(q_b)\vec{L}_b$ is the body's angular velocity about its

COM. $\mathbf{R}(q)$ is a 3×3 matrix that rotates vectors from the body frame to the space frame [29].

The net force $\vec{F}$ and torque $\vec{\tau}$ acting on body b in the space frame are the sums of the individual forces and torques resulting from the particle–particle forces $\vec{f}_i$ computed by existing algorithms. The sums

$$\vec{F}_b = \sum_{k=1}^{N_b} \vec{f}_{B_{bk}} \tag{3}$$

and

$$\vec{\tau}_b = \sum_{k=1}^{N_b} \left[\mathbf{R}(q_b) \cdot \vec{D}_{bk}\right] \times \vec{f}_{B_{bk}} \tag{4}$$

are performed over all constituent particles.

2.1. NVE integration scheme

In the microcanonical NVE ensemble, Newtonian mechanics [29] governs the motion of rigid bodies with the following equations

$$\dot{\vec{R}}_b = \vec{V}_b, \tag{5}$$

$$\dot{\vec{V}}_b = \vec{F}_b / M_b, \tag{6}$$

$$\dot{\vec{L}}_b = \vec{\tau}_b, \tag{7}$$

$$\dot{q}_b = \frac{1}{2}\mathbf{A}(\vec{\omega}_b) \cdot q_b, \tag{8}$$

where $\mathbf{A}(\vec{\omega}_b)$ is a 4×4 matrix defined in Ref. [29].

These equations are numerically integrated in a way analogous to the velocity Verlet discretization scheme used for unconstrained particles [1]. The velocity and angular momentum of each rigid body are first updated to $t + \Delta t/2$, and the position and orientation are updated to $t + \Delta t$ by the equations

$$\vec{V}(t + \Delta t/2) = \vec{V}(t) + \frac{\Delta t}{2M} \cdot \vec{F}(t), \tag{9}$$

$$\vec{R}(t + \Delta t) = \vec{R}(t) + \Delta t \cdot \vec{V}(t + \Delta t/2), \tag{10}$$

$$\vec{L}(t + \Delta t/2) = \vec{L}(t) + \Delta t/2 \cdot \vec{\tau}(t), \tag{11}$$

$$q(t + \Delta t) = \mathrm{Q}\big(q(t), \Delta t, \vec{L}(t + \Delta t/2), I\big), \tag{12}$$

where the function Q is an application of the Richardson method [30].

Forces and torques are then calculated based on the updated positions and orientations, and the velocity and angular momentum are advanced fully to $t + \Delta t$,

$$\vec{V}(t + \Delta t) = \vec{V}(t + \Delta t/2) + \frac{\Delta t}{2M} \cdot \vec{F}(t + \Delta t), \tag{13}$$

$$\vec{L}(t + \Delta t) = \vec{L}(t + \Delta t/2) + \Delta t/2 \cdot \vec{\tau}(t + \Delta T). \tag{14}$$

2.2. NVT and NPT integration schemes

One method to model a system of rigid bodies in a canonical NVT ensemble is to combine a Langevin thermostat with an NVE integration scheme, also known as Brownian dynamics (BD) [1]. The thermostat is applied to each individual particle in the system, which effectively thermalizes the rigid bodies without momentum conservation.

Simulations in the NVT ensemble, as well as isothermal–isobaric NPT ensemble, can also be accomplished with the application of a Nosé–Hoover thermostat (and for NPT, a barostat) with an extended Hamiltonian. Miller and coauthors [29] derive a Hamiltonian formulation of the NVE rigid body equations of motion by introducing the conjugate quaternion momentum. Kameraj and coauthors [31] extend it with requisite thermostat and barostat and derive the resulting numerical integration steps similar to Eqs. (9)–(14).

Algorithm 1 Update bodies, step 1

Require: $\lceil N_{\text{bodies}}/\text{blockDim} \rceil$ blocks are run on the device

$b \leftarrow \text{blockIdx} \cdot \text{blockDim} + \text{threadIdx}$
if $b \leqslant N_b$ **then**
  $M_b \Rightarrow M$
  $\mathbf{I}_b \Rightarrow \mathbf{I}$
  $\vec{R}_b \Rightarrow \vec{R}_{\text{old}}$
  $\vec{V}_b \Rightarrow \vec{V}_{\text{old}}$
  $\vec{L}_b \Rightarrow \vec{L}_{\text{old}}$
  $q_b \Rightarrow q_{\text{old}}$
  $\vec{F}_b \Rightarrow \vec{F}$
  $\vec{\tau}_b \Rightarrow \vec{\tau}$
  $\vec{V} \leftarrow \vec{V}_{\text{old}} + \frac{\Delta t}{2M} \cdot \vec{F}$
  $\vec{V}_b \leftarrow \vec{V}$
  $\vec{R}_b \Leftarrow \vec{R}_{\text{old}} + \Delta t \cdot \vec{V}$
  $\vec{L} \leftarrow \vec{L}_{\text{old}} + \Delta t/2 \cdot \vec{\tau}$
  $\vec{L}_b \Leftarrow \vec{L}$
  $q_b \Leftarrow \mathrm{Q}(q_{\text{old}}, \Delta t, \vec{L}, \mathbf{I})$
end if

3. Implementation

Augmenting HOOMD-blue to include rigid body constraints is accomplished in two parts. First, the following data structures are added to hold the dynamic, static, and computed properties for each body: $\vec{R}_b$, $\vec{V}_b$, q_b, $\vec{L}_b$, M_b, $\mathbf{I}_b$, N_b, B_{bk}, $\vec{D}_{bk}$, $\vec{F}_b$, and $\vec{\tau}_b$. Each quantity with a single subscript is stored in a simple array. Those with two subscripts are stored in rectangular matrices where the second index is the fastest varying index. Dimensions are sized to the largest body and the leftover space padded with zeroes. Second, new routines are written that integrate the equations of motion of the rigid bodies in the system, with separate versions for the NVE, NVT, and NPT ensembles.

To optimize performance, all data structures are stored in device memory and all integration steps are carried out on the GPU. No communication is required between the host and the device to advance the system. Although padded matrices waste some memory in systems where different bodies contain different numbers of particles, they enable contiguous memory accesses in the integration kernels.

3.1. NVE integration kernels

In HOOMD-blue, the integration of Newton's equations of motion for rigid bodies, Eqs. (9)–(14), is distributed over five kernels. The first two kernels update the state of the body and its constituent particles. Next, one kernel sums the force and torque on each body from the forces applied to its particles. Finally, two kernels apply the second half of the update to the state of the body and its particles.

Pseudocode describing the basic structure of these kernels is provided in Algorithms 1 and 2. Within the pseudocode, device memory reads/writes are indicated by a double arrow $\Rightarrow/\Leftarrow$ and local memory writes by a single arrow $\leftarrow$. The performance of each of these kernels is bound by device memory bandwidth. Memory accesses are ordered to be contiguous so as to best utilize the cache hierarchy on the GF100 and maximize their performance.

The first kernel, detailed in Algorithm 1, updates the state of the rigid body at the beginning of the time step. Each thread loads

Algorithm 2 Update particles

Require: N_{body} blocks are run on the device
Require: N I, $\vec{R}$, q, $\vec{V}$, and $\vec{\omega}$ are stored in shared memory.
1: $b \leftarrow$ blockIdx
2: **if** threadIdx $== 0$ **then**
3: $N_b \Rightarrow N$
4: $\mathbf{I}_b \Rightarrow \mathbf{I}$
5: $\vec{R}_b \Rightarrow \vec{R}$
6: $q_b \Rightarrow q$
7: $\vec{V}_b \Rightarrow \vec{V}$
8: $\vec{L}_b \Rightarrow \vec{L}$
9: $\vec{\omega} \leftarrow \mathbf{R}^T(q)\mathbf{I}^{-1}\mathbf{R}(q)\vec{L}$
10: **end if**
11: syncthreads()
12: **for** $w = 1$ to $\lceil N/\text{blockDim} \rceil$ **do**
13: $k \leftarrow w * \text{blockDim} + \text{threadIdx}$
14: **if** $k \leqslant N$ **then**
15: $B_{bk} \Rightarrow i$
16: $\vec{D}_{bk} \Rightarrow \vec{D}$
17: $\vec{r}_i \Leftarrow \vec{R} + \mathbf{R}(q) \cdot \vec{D}$
18: $\vec{v}_i \Leftarrow \vec{V} + \vec{\omega} \times (\mathbf{R}(q) \cdot \vec{D})$
19: **end if**
20: **end for**

state data for its assigned body from global memory, updates the position, orientation, velocity, and angular momentum following Eqs. (9)–(12), and writes the updated state back to global memory. All memory transactions made by Algorithm 1 are contiguous.

The second kernel, detailed in Algorithm 2, sets the constrained position and velocity of each particle that belongs to a rigid body. One block of threads is assigned to each body. At the beginning of the kernel, one thread loads the state of the rigid body into shared memory and a barrier synchronization is performed. Then, all threads participate in computing $\vec{r}_i$ and $\vec{v}_i$. Each thread computes these quantities for several particles i, where $i = B_{bk}$ and $k = \text{threadIdx} + w \cdot \text{blockDim}$, in a loop over $w = 0, 1, 2, 3, \ldots$. This sliding window construction handles bodies of arbitrary size with a single fixed block size. The matrices B_{bk} and $\vec{D}_{bk}$ are stored with k as their fast index so that the reads on lines 15 and 16 of Algorithm 2 by neighboring threads are contiguous in memory. The writes on lines 17 and 18 may or may not be contiguous, depending on the order in which particle indices are stored in B_{bk}. To avoid this potential performance hit, all particles in body b are grouped together and listed in order in B_{bk}.

Next, particle–particle forces are computed via the standard MD force calculation kernels. Then the net force $\vec{F}_b$ and torque $\vec{\tau}_b$ on each body are calculated in the third kernel. As in Algorithm 2, one block of threads is assigned to each body. Each thread i loads B_{bk}, $\vec{D}_{bk}$, and the force $\vec{f}_k$ from global memory and the net force and torque are summed using a standard parallel reduction performed in shared memory. The resultant $\vec{F}_b$ and $\vec{\tau}_b$ are then written out to global memory.

In the fourth kernel, the velocity and angular momentum of each rigid body are updated again via Eqs. (13) and (14). One thread is assigned to each rigid body in a manner analogous to Algorithm 1.

Finally, in the fifth kernel, each body's particles are set to their updated constrained velocity. One block of threads is assigned per body. The kernel is nearly identical to Algorithm 2, except that only the particle velocity is calculated and written to global memory.

All particles that are not part of a rigid body are updated to the next step by the existing standard MD integration kernels. Validation and performance results for these rigid body integration algorithms are provided in Algorithm 4.

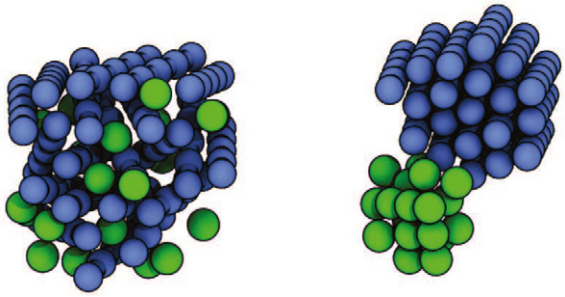

Fig. 1. (Left) Initial configuration of randomly placed rods (blue) intermixed with free particles (green). Rods are attracted to rods and free particles are attracted to free particles. (Right) Final configuration after the FIRE energy minimization converges. All system snapshots in this paper are composed in VMD [32] and raytraced with Tachyon [33]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.2. FIRE energy minimization

The FIRE algorithm [21] works in conjunction with any MD integrator to compute a trajectory to a local energy minimum. At each iteration step, the integrator is used to advance the positions and velocities for all the particles in the system, given the computed forces. FIRE modifies velocities and the step size by the following prescription. As long as the particles in the system are moving in directions that lower the energy of the system as a whole, and have been for a sufficient number of steps, particle velocities and the step size are increased, subject to limits. As soon as the particles are no longer moving so as to lower the energy of the whole system, all particles are brought to a halt, the step size is decreased, and new velocities are generated in the direction of the force on each particle. Convergence to a minimum energy is attained when the root mean square force and change in the energy of the system are below set tolerances. In Ref. [21], FIRE is demonstrated to be effective and surprisingly fast compared to competing schemes, even for systems with millions of degrees of freedom.

We extend FIRE to a system containing rigid bodies by adding the orientation of the rigid bodies to the degrees of freedom and use the rigid body NVE integrator to advance the positions, velocities, orientations, and angular velocities of the bodies. Both the center of mass velocities and the angular velocities of all the bodies are reset to zero if the energy of the system stops decreasing. Convergence is reached when the root mean square force, root mean square torque, and change in the energy of the system are below set tolerances. Ref. [21] points out that all degrees of freedom must be comparable for the algorithm to work. In practice, we find that the orientation is a sufficiently comparable degree of freedom and that it does not require special handling.

Fig. 1 demonstrates FIRE applied to an arrangement of rods and free particles. The rods are rigid bodies composed of five particles arranged linearly. Rod particles interact with other rod particles by the attractive Lennard-Jones (LJ) potential. Free particles also interact by the attractive LJ potential as well. Rod particles and free particles interact by a WCA volume excluding potential. An energy minimization is performed with a force tolerance of 1e−4, a torque tolerance of 0.1, and a change in energy tolerance of 1e−12. The FIRE energy minimization causes the rods to collapse into a hexagonally packed bundle and the free particles to collect into a droplet outside of the rod bundle after 60 684 iterations. Only the first five percent of the time steps are spent collapsing the rod bundle. The rest are needed to collect the dispersed LJ droplets into a single droplet.

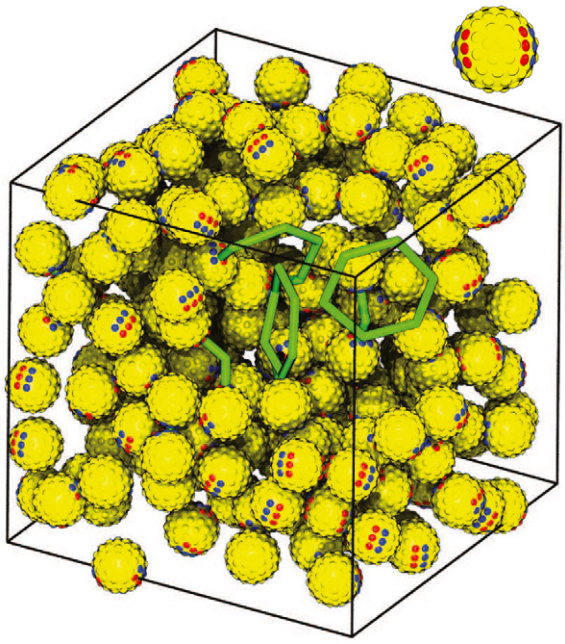

Fig. 2. A system of 225 patchy spheres, each composed of 90 particles. The red and blue particles are attractive patches on the surface of the body. A single patchy sphere is shown in the upper right for reference. As shown by Zhang [34], these bodies self-assemble into rings of six spheres. The spheres have been made invisible in the frontmost octant so that the ring structure formed by the invisible spheres can be shown in green. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

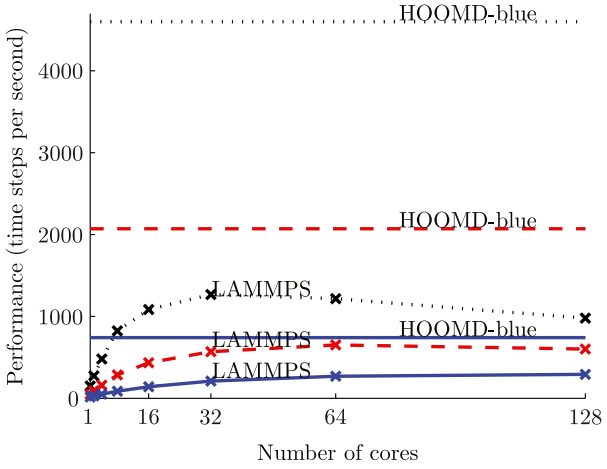

Fig. 3. Performance in time steps per second obtained while running a simulation of 225 (dotted lines), 667 (dashed lines), and 2000 (solid lines) rigid bodies consisting of 20 250, 60 030, and 180 000 particles respectively. LAMMPS performance on 1, 2, 4, 8, 16, 32, 64, and 128 CPU cores is compared to HOOMD-blue performance on a single NVIDIA GTX 480 (indicated by the horizontal lines).

4. Validation and performance

The rigid body constraint algorithm is well established in serial and parallel CPU codes [12,30] and is mathematically no different when implemented on the GPU. However, to verify the correct function of our code, various quantities are checked for validity including energy and momentum conservation in the NVE ensemble, as well as temperature and pressure stability and the correct distribution of energy over the degrees of freedom in the NVT and NPT ensembles. Numerous rigid body systems are also simulated side-by-side on both the CPU and GPU to compare the results and evaluate their relative performance.

The performance scaling of the GPU-accelerated algorithm is tested with simulations of a system of "patchy particles" studied by Zhang et al. [34]. These rigid bodies shall be subsequently referred to as "patchy spheres" to avoid confusion with our usage of the word "particle" to refer to the smallest simulation unit. Each patchy sphere is a rigid body composed of 90 particles distributed on the surface of a sphere. Two attractive patches, each constructed from two linear arrangements of particles that interact with Lennard-Jones pair potentials, are placed at an angle $\theta = 2\pi/5$ with respect to the center of the body. Per Zhang et al. [34], this system self-assembles into rings containing six patchy spheres. The chosen benchmark systems consist of 225, 667, and 2000 patchy spheres resulting in 20 250, 60 030, and 180 000 individual particles, respectively. Each system was annealed to an equilibrium structure at $k_BT = 1.0\varepsilon$. Fig. 2 shows the system of 225 patchy spheres.

Each simulation is executed using both the LAMMPS and HOOMD-blue code packages. LAMMPS simulations are deployed in parallel over 1, 2, 4, 8, 16, 32, 64, and 128 cores on the Nyx cluster at the University of Michigan. The nodes used are HP ProLiant DL1000 models with Intel® Xeon® e5540 processors operating at 2.53 GHz and connected via 20 GB/s Infiniband. All nodes have identical software configurations, running an x86_64 install of RHEL 5.5, CUDA 3.0, and NVIDIA drivers 195.36.24. The HOOMD-blue simulations were performed on a custom built workstation with a single NVIDIA GTX 480. It also contains an AMD Athlon™ II X4 630 processor operating at 2.8 GHz and runs CentOS 5.5, CUDA 3.0, and NVIDIA drivers 260.19.21.

Performance results are measured by the number time steps that are executed per second and are shown in Fig. 3. For the 20 K and 60 K particle systems, LAMMPS achieves peak performance at 32 and 64 cores, respectively. For the 180 K particle system LAMMPS no longer scales well at 128 cores; the performance is only 11% faster than it is at 64 cores. The reason for the poor scaling is the inter-node communication of the rigid body data structures during the time step. LAMMPS uses spatial decomposition to parallelizes an MD simulation over many cores. In simulations of rigid bodies on a CPU cluster, the particles of a given body can be distributed over an arbitrary number of cores. The force and torque summation is performed in LAMMPS by an all-reduce operation that returns results from all nodes to each node [30]. In comparison, the GPU-accelerated implementation is deployed on a single GPU and requires no inter-node or even host-device communication. The equivalent operation to the all-reduce operation is performed within a block on a single streaming multiprocessor. Consequently, HOOMD-blue attains a level of performance for rigid body simulations that cannot be reached with a parallel CPU-only code. For these patchy sphere benchmarks in particular, over a wide range of system sizes HOOMD-blue is 2.5–3.6× faster than LAMMPS at its peak performance for any number of cores.

We also tested systems that mix rigid bodies and unconstrained particles. One example, shown in Fig. 4, is a system of polymer-tethered nanorods originally studied in Ref. [2] using LAMMPS. In this simulation, each tethered rod is composed of a five particle rigid rod and a nine particle flexible tether. One thousand tethered rods, for a total of 14 000 particles, are placed in a box with packing fraction of 0.22. Rod particles are attracted to each other via a shifted Lennard-Jones pair potential with an interaction cutoff of 2.5 distance units. All other particle interactions are WCA volume excluding. The system is in an NVT ensemble with a kinetic temperature of $k_BT = 1.4\varepsilon$, where ε is the well depth of the Lennard-Jones potential. At these parameters the tethered nanorods self-assemble into lamellar bilayers [2].

Simulations are executed with HOOMD-blue on three modern NVIDIA GPUs, a GTX 480, a Tesla S1070, and a Tesla S2050. The Tesla S1070 and S2050 are installed in the Nyx cluster environ-

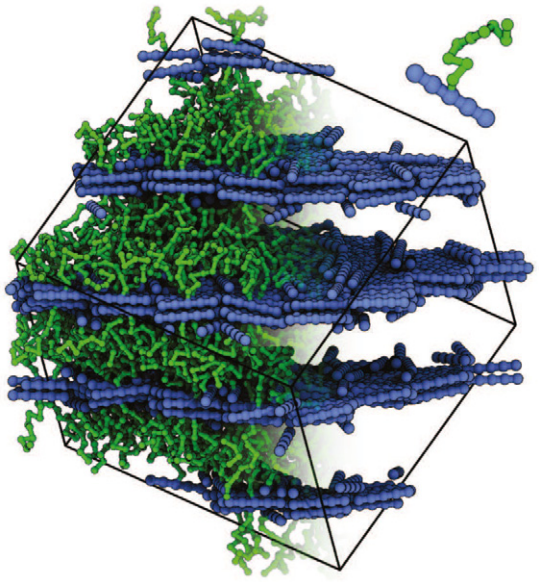

Fig. 4. A system of one thousand tethered nanorods that have self-assembled into a lamellar bilayer. The upper right inset depicts a single tethered nanorod for reference. Each tethered nanorod is modeled by five particles rigidly connected in a line, attached to a flexible tether of nine particles. Bonds, both rigidly constrained and unconstrained, are shown as cylinders. Tethers have been removed from view in the right half of the image.

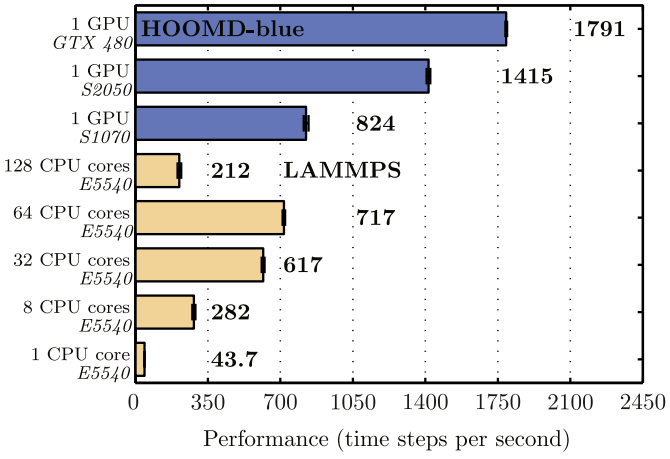

Fig. 5. Performance in time steps per second obtained while running a simulation of one thousand tethered nanorods (14 000 total particles) on various hardware configurations. Each benchmark is performed 50 times. Bars are plotted at the median value and error bars display one standard deviation of variability.

ment where they are hosted by IBM System x3455 nodes each with two AMD Opteron™ 2356 processors operating at 2.3 GHz. The LAMMPS simulations were deployed over 1, 8, 32, 64, and 128 cores of the Nyx cluster in the same configuration used for the patchy sphere runs.

The results of this side-to-side comparison is shown in Fig. 5. HOOMD-blue running on a GTX 480 executes the tethered nanorod simulation at 1791 time steps per second, which is 2.5× faster than LAMMPS running at peak performance in parallel on 64 CPU cores.

5. Conclusion

This paper discusses how a rigid body constraint algorithm is incorporated into HOOMD-blue, a massively parallel GPU-accelerated MD application. All data structures are stored on the GPU in order to attain the highest level of performance possible by avoiding costly transfers between the host and device. The performance of the kernels implementing the rigid body integration steps is limited only by the device memory bandwidth. This is achieved by carefully avoiding unnecessary device memory accesses and arranging the access patterns so as to make optimal use of the cache hierarchy on the GF100.

Methods for simulating NVE, NVT, and NPT ensembles of rigid bodies are implemented in HOOMD-blue version 0.10.0, which is available free and open source [28]. While two orders of magnitude increases in computational speed over a single CPU core have already been documented for this code package running basic MD simulations [25,28], the GPU is especially well-suited for rigid body constraints. Two case studies are presented in this paper where HOOMD-blue consistently executes a factor of 2.5–3.6 times faster than the peak performance of the LAMMPS code package parallelized over any number of cores.

This paper also introduces a modest adaptation to the FIRE energy minimization algorithm that makes it suitable for use with rigid bodies. To our knowledge, HOOMD-blue is the first MD code to allow energy minimization to be applied to systems with rigid body constraints.

With GPU acceleration, MD simulations of systems of rigid bodies can now be carried out on larger systems and for longer time scales on a single workstation than was previously possible even on large clusters. This advance will allow simulations of diverse systems, from molecules and proteins to nanoparticles and colloids, and explorations of previously inaccessible phase spaces.

Acknowledgements

We thank Aaron Keys for helpful discussions on the FIRE algorithm for rigid bodies and Anthony Sheh for helping with the CPU implementation of the quaternion scheme. This research was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Materials Sciences and Engineering, under award DE-FG02-02ER46000, the U.S. Department of Energy Computational Science Graduate Fellowship, and DOD/DDRE under the National Security Science & Engineering Faculty Fellowship award N00244-09-1-0062. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the DOD/DDRE. T.D.N. also acknowledges the Vietnam Education Foundation. The authors acknowledge support from the James S. McDonnell Foundation 21st Century Science Research Award/Studying Complex Systems.

References

[1] D. Frenkel, B. Smit, Understanding Molecular Simulations, 2nd edition, Academic Press, 2002.
[2] T.D. Nguyen, S.C. Glotzer, ACS Nano 4 (2010) 2585.
[3] J.A. Anderson, R. Sknepnek, A. Travesset, Phys. Rev. E 82 (2010).
[4] C.L. Phillips, C.R. Iacovella, S.C. Glotzer, Soft Matter 6 (2010) 1693.
[5] Y. Duan, P.A. Kollman, Science 282 (1998) 740.
[6] C.D. Snow, H. Nguyen, V.S. Pande, M. Gruebele, Nature 420 (2002) 102.
[7] P.L. Freddolino, C.B. Harrison, Y. Liu, K. Schulten, Nat. Phys. 6 (2010).
[8] J. Ryckaert, G. Ciccotti, H. Berendsen, J. Comput. Phys. 23 (1977) 327.
[9] D. Dubbeldam, G.A.E. Oxford, R. Krishna, L.J. Broadbelt, R.Q. Snurr, J. Chem. Phys. 133 (2010).
[10] P. Eastman, V.S. Pande, J. Chem. Theory Comput. 6 (2010) 434.
[11] D. Evans, S. Murad, Mol. Phys. 34 (1977) 327.
[12] DLPOLY, http://www.ccp5.ac.uk/DL_POLY/, 2010.
[13] S. Plimpton, J. Comput. Phys. 117 (1995) 1.
[14] D.R. Heine, M.K. Petersen, G.S. Grest, J. Chem. Phys. 132 (2010).
[15] M.A. Horsch, Z. Zhang, S.C. Glotzer, Soft Matter 6 (2010) 945.
[16] T.D. Nguyen, S.C. Glotzer, Small 5 (2009) 2092.
[17] T.D. Nguyen, Z. Zhang, S.C. Glotzer, J. Chem. Phys. 129 (2008).
[18] J. Elliott, A. Windle, J. Chem. Phys. 113 (2000) 10367.
[19] R. Mountain, Int. J. Thermophys. 28 (2007) 536.
[20] S.D. Johnson, R.D. Mountain, P.H.E. Meijer, J. Chem. Phys. 103 (1995) 1106.

[21] E. Bitzek, P. Koskinen, F. Gaehler, M. Moseler, P. Gumbsch, Phys. Rev. Lett. 97 (2006).
[22] J.M. Troyer, F.E. Cohen, Proteins 23 (1995) 97.
[23] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes in C, Cambridge University Press, 1997.
[24] NVIDIA GTX 480 product page, http://www.nvidia.com/object/product_geforce_gtx_480_us.html, 2010.
[25] J.A. Anderson, C.D. Lorenz, A. Travesset, J. Comput. Phys. 227 (2008) 5342.
[26] J.A. van Meel, A. Arnold, D. Frenkel, S.F.P. Zwart, R.G. Belleman, Mol. Simul. 34 (2008) 259.
[27] J.E. Stone, et al., J. Comput. Chem. 28 (2007) 2618.
[28] HOOMD-blue, http://codeblue.umich.edu/hoomd-blue/, 2010.
[29] T.F. Miller, et al., J. Chem. Phys. 116 (2002) 8649.
[30] LAMMPS source code, http://lammps.sandia.gov/, 2010.
[31] H. Kameraj, R.J. Low, M.P. Neal, J. Chem. Phys. 122 (2003) 224114.
[32] W. Humphrey, A. Dalke, K. Schulten, J. Mol. Graphics 14 (1996) 33.
[33] J. Stone, An efficient library for parallel ray tracing and animation, Master's thesis, Computer Science Department, University of Missouri-Rolla, 1998.
[34] Z. Zhang, S.C. Glotzer, Nano Lett. 4 (2004) 1407.