Computer Physics Communications 203 (2016) 45–52

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Efficient neighbor list calculation for molecular simulation of colloidal systems using graphics processing units

Michael P. Howard [a], Joshua A. Anderson [b], Arash Nikoubashman [a,1], Sharon C. Glotzer [b,c], Athanassios Z. Panagiotopoulos [a,*]

[a] Department of Chemical and Biological Engineering, Princeton University, Princeton, NJ 08544, USA
[b] Department of Chemical Engineering, 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 24 August 2015
Received in revised form
1 February 2016
Accepted 4 February 2016
Available online 3 March 2016

Keywords:
Molecular simulation
Colloid
Size disparity
Non-uniform
Neighbor list
Bounding volume hierarchy
GPU

ABSTRACT

We present an algorithm based on linear bounding volume hierarchies (LBVHs) for computing neighbor (Verlet) lists using graphics processing units (GPUs) for colloidal systems characterized by large size disparities. We compare this to a GPU implementation of the current state-of-the-art CPU algorithm based on stenciled cell lists. We report benchmarks for both neighbor list algorithms in a Lennard-Jones binary mixture with synthetic interaction range disparity and a realistic colloid solution. LBVHs outperformed the stenciled cell lists for systems with moderate or large size disparity and dilute or semidilute fractions of large particles, conditions typical of colloidal systems.

1. Introduction

There has been a steady growth of computational resources available to the scientific community [1]. The fastest supercomputers today offer petascale performance through hundreds of thousands of CPU cores with dedicated coprocessors or graphics processing units (GPUs) as accelerators. Whereas in the past most atomistic molecular simulations were restricted to no more than a few hundred particles over nanosecond time scales, modern computing architectures and simulation techniques have enabled simulations of millions of particles [2] and up to millisecond [3] time scales. In particular, molecular dynamics (MD) methods have emerged as powerful tools for large-scale molecular simulations for two important reasons: (1) the MD algorithm is highly parallel and so easily adapted for supercomputing, and (2) many highly optimized and flexible simulation packages are readily available to researchers. In the past two decades, significant time and resources have been devoted to the development of such MD packages, including GROMACS [4], LAMMPS [5], and NAMD [6], among many other commercial and open-source options. These packages provide robust MD implementations for massively parallel computers. A more recent addition is HOOMD-blue [7], which was developed and optimized for GPUs, and has a single GPU performance one order of magnitude faster than a single CPU [8].

Despite these advances in hardware and software, MD simulations of soft matter remain challenging because there is typically a large disparity in length and time scales between components. For example, colloidal particles (nanometers to micrometers in diameter) in solution are separated in size by several orders of magnitude from an atomistic description of the molecular solvent. MD simulations retaining full atomistic detail of the solvent can become intractable because many solvent atoms must be included to model only a few colloidal particles. Moreover, the time scales associated with the degrees of freedom of the solvent are generally much shorter than the relatively slow motion of the larger colloids. This means that these simulations require very short MD time steps to faithfully capture the dynamics of the solvent, and many such steps are required to observe any appreciable dynamics of the colloids.

The MD algorithm in its simplest form consists of two basic steps: (1) calculation of the forces on all particles and

* Corresponding author.
E-mail address: azp@princeton.edu (A.Z. Panagiotopoulos).
[1] Present address: Institute of Physics, Johannes Gutenberg University Mainz, Staudingerweg 7, 55128 Mainz, Germany.

http://dx.doi.org/10.1016/j.cpc.2016.02.003
0010-4655/

(2) integration of Newton's equations of motion. The force calculation is by far the most computationally expensive part of the MD algorithm. In particular, the calculation of nonbonded pair interactions between particles typically dominates the force calculation. In the simplest MD implementation, the forces between all possible pairs of the N total particles in the simulation are evaluated, leading to $O(N^2)$ scaling.

To reduce the number of force pairs evaluated, the interaction potential between particles of types i and j is typically truncated at a radial cutoff distance r_{ij} where the force has decayed sufficiently so that truncation does not significantly influence the properties of interest. A neighbor (Verlet) list storing a list of particles that are within the cutoff is then created for each particle [9]. The pair forces only need to be computed for the particles in the neighbor list, which is a small subset of N for each particle. The neighbor list can be rebuilt less frequently than every MD step if a small buffer width is added to r_{ij}, trading wasted pair force distance checks with the frequency of rebuilding the neighbor list, which accelerates the calculation compared to evaluating all possible force pairs at every step. However, the force calculation is ultimately still $O(N^2)$ if the neighbor list is built by simply checking the distances between all particle pairs.

Acceleration structures reduce the computational cost of building the neighbor list by restricting the neighbor search for each particle to a subset of the particles in the system. The most commonly employed acceleration structure in general-purpose MD codes is the cell list. A typical cell list spatially bins particles into uniformly sized cells in $O(N)$ [9]. Distance checks must only be performed for particles that are in neighboring cells, effectively reducing the cost of computing the neighbor list to $O(Nm)$, where m is the average number of particles in a cell (usually $m \ll N$). The cell width is typically determined by the largest cutoff radius between all pairs so that 27 cells must be checked for each particle in three-dimensional simulations.

The cell list is extremely efficient in simulations that have nearly equal pair force cutoffs and a uniform particle distribution between the cells. However, performance degrades significantly in colloidal systems due to the large disparity in interaction lengths. Many unnecessary distance checks are performed for particles with short interaction ranges when the cell width is based on the largest cutoff, schematically illustrated in Fig. 1. The cell width is based on the largest cutoff r_{BB}. The solvent particles with cutoff r_{AA} must check the same number of cells (particles) as the colloids with the larger cutoff r_{BB}. However, unlike the colloids, the solvent particles reject many of these particles from their neighbor lists.

A general solution to this problem has been successfully deployed in LAMMPS [10]. The standard cell list is extended so that the cell width is based on the shortest cutoff and each particle type searches a different "stencil" of adjacent cells based on the largest cutoff radius. Distances to each cell in the stencil are precomputed so that a particle distance check can be skipped for many of the searched particles. This stenciled cell list method was reported to give speedups of nearly 100× for a colloidal solution with a 20:1 ratio in diameter compared to a standard cell list in LAMMPS. However, simulations of colloidal systems with such large size disparity are still extremely computationally intensive, requiring hundreds of CPU cores to obtain reasonable performance [10].

In this article, we explore two parallel algorithms for efficiently building neighbor lists in colloidal systems on the GPU: one based on stenciled cell lists and one based on a hierarchical tree acceleration structure. To our knowledge, the stenciled cell list algorithm [10] has not been previously implemented and tested on the GPU. In graphics processing, hierarchical tree data structures are used for performance-critical neighbor searches [11]. One such tree structure, the bounding volume hierarchy (BVH), has previously been used to generate neighbor lists between large macromolecules on the CPU [12,13]. However, these prior studies did not extend their approach to general-purpose molecular simulations, did not address size disparity between different particle types, and did not discuss implementation of the algorithm on GPUs.

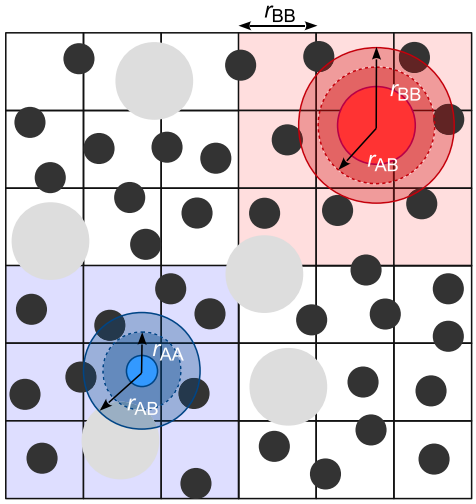

Fig. 1. Cell list needed to determine the neighbors of solvent (A) and colloid (B) particles when the bin size is based on the largest cutoff length r_{BB}. The pair interaction ranges are illustrated for each particle type. The shaded areas indicate the cells that each particle must search.

In Section 2, we present and compare parallel algorithms for the stenciled cell list and BVH. Technical details of the implementation of these algorithms within the HOOMD-blue simulation package are described in Section 3. Systematic performance benchmarks for the algorithms are reported in Section 4.

2. Algorithms

2.1. Stenciled cell list

The stenciled cell list [10] is a straightforward extension of the standard cell list, and is illustrated in Fig. 2. All particles are binned into a cell list of nominal cell width Δ_{bin}. The maximum cutoff radius is determined for each particle type, and a stencil is computed from the list of offsets to neighboring cells that have a nearest separation distance within that cutoff. For example, the solvent particle has a stencil radius corresponding to r_{AB}, and the list of offsets in 2D is $(0, 0)$, $(+1, +2)$, $(+2, -1)$, ... The colloid has a stencil radius r_{BB}. All the cells included in the stencils are shaded and outlined with a solid line.

Because the stencil size is set by the maximum cutoff radius per type, many particles that will not be included in the neighbor list must still be iterated over for shorter r_{ij}. In Fig. 2, both the solvent particle and colloid have the same effective stencil size for the given cutoffs. However, extra distance checks can be eliminated by precomputing the minimum distance to each cell in the stencil. For a particle of a given type, if the minimum distance to the nearest cell is greater than the pairwise cutoff, that particle can be skipped without distance checking or reading its position. The solvent particle only needs to distance check particles of type A inside the cells indicated by the dashed line corresponding to r_{AA}, and all particles of type A can be skipped in the other cells in the stencil.

The neighbor list is then built as described in Algorithm 1. A given particle looks up the appropriate stencil of cells based on its particle type (line 4). Each member of the stencil is iterated over (line 5), and each offset from the stencil is converted into a neighbor cell based on the current cell of the particle, including