Advances in Engineering Software xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Advances in Engineering Software

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

Research paper

50 million atoms scale molecular dynamics modelling on a single consumer graphics card

Gaobo Xiao, Mingjun Ren*, Haibo Hong

State Key Laboratory of Mechanical System and Vibration, School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai, People's Republic of China

ARTICLE INFO

Keywords:
Molecular dynamics
parallel computing
Graphics processing unit (GPU)
Compute unified device architecture (CUDA)
dynamic cell list

ABSTRACT

This paper presents a dynamic cell-list method for realizing large scale molecular dynamics (MD) simulations with more than 50 million atoms on a single consumer graphics card. It adapts the cell-list algorithm by introducing an efficient two-step atom location scheme and a dynamic memory allocation scheme such that only those cells containing atoms consume device memory. In addition, a large amount of memory is saved since it does not use the neighbour list. The computational efficiency is improved by reducing the memory loading times and maximizing coalesced memory access as compared to methods utilizing neighbour lists, since memory bandwidth is becoming the bottle-neck of the latest GPUs. As a result, MD simulations with more than 50 million atoms utilizing advanced three-body interaction potential are made possible on a consumer graphics card with just 11 GB of graphics memory. The proposed framework is designed to run totally on the graphics card, with all the data stored in the graphics memory to avoid the time-consuming data transfer between host and device. It achieves 2.5 times the speed and 20 times the atom number of the latest Lammps GPU package on the NVIDIA GTX 1080Ti GPU. The proposed framework is expected to help adapt existing MD packages for supporting large scale MD simulations on personal desktops and thereby extend MD to a wider range of researchers and engineers.

1. Introduction

Molecular dynamics (MD) has been a powerful tool for studying the atomic scale phenomena in many fields like theoretical physics [1], material science [2,3], biochemistry [4] and biophysics [5]. The major problem with this method is its huge demand in computations [6], limiting the time and spatial scales that can be handled effectively.

A significant advance in scientific computing in recent years is the development of general purpose computing on graphics processing unit (GPU) [7,8], especially the introducing of Compute Unified Device Architecture (CUDA) [9,10], a parallel computing platform and application programming interface (API) model created by NVIDIA corporation. CUDA allows researchers and engineers to take advantage of the massive computing power of the thousands of computing cores on NVIDIA GPUs, bringing tens to hundreds of times of speedup for scientific computing as compared to implementations on CPUs [11]. OpenCL (open computing language) is another framework that supports parallel programming for GPU and enables general purpose computing on GPUs from other vendors [12].

Many MD codes, e.g. LAMMPS [12], AMBER [13], and Gromacs [14], have added support for GPU acceleration by providing options of transferring the computational intensive parts down to the GPU in each time step. Since the back and forth transferring of data between host and the device is inefficient, a number of MD codes have been developed from scratch to harness the full computational power of GPU by implementing the computing process entirely on GPU, including RUMD [15], OpenMM [16,17], crystal MD [18], ACEMD [4], HAL's MD [19], and HOOMD-Blue [20].

A common point of these GPU implementations of MD is that they focused on the boost of computational speed for small to medium sized systems on single GPU, i.e. from thousands to a few millions of atoms, and resorted to multiple GPU systems, e.g. computer clusters [21,22], for handling systems with more atoms. What is less studied is the MD simulation of large scale systems on a single GPU, e.g. systems with tens of millions of atoms. This makes it difficult for many researchers and engineers, for whom access to supercomputers is inconvenient, to investigate those problems that require a modelling scale from several tens to a few hundreds of nanometers by MD simulation [23]. Examples include ultra-precision machining [24], atomic force microscopy (AFM) [23], and nano electro-mechanical systems (NEMS) [25], in which intensive deformation happens on the scale of several tens to around a hundred nanometers. Phenomena on this scale is difficult to be observed by experimental techniques, or to be modelled by continuum methods like finite element method (FEM) [26]. MD is the right method

* Corresponding author:
E-mail address: renmj@sjtu.edu.cn (M. Ren).

https://doi.org/10.1016/j.advengsoft.2018.08.004
Received 26 June 2018; Received in revised form 29 July 2018; Accepted 12 August 2018

for modelling phenomena on this scale, but its application is very limited due to the high costs of supercomputers [6].

This study presents a framework for implementing large scale MD simulations with tens of millions of atoms on a single graphics card. This framework adapts the cell-list method by introducing an efficient two-step atom location scheme and a dynamic memory allocation scheme to save the device memory usage. The computational efficiency is also improved due to the reduced times of memory loading and coalesced memory access as compared to methods utilizing neighbour lists. As a result, MD simulation with more than 50 million atoms with advanced three-body interaction potential is realized on a single GPU with only 11 GB of graphics memory. Benchmark tests show that the proposed framework achieves 2.5 times the speed and 20 times the particle number as compared to the latest Lammps GPU package [27] on the same GPU hardware. As a demonstration of potential application, the codes are used to simulate the nano-grinding of SiC under practical cutting depth to visualize the crack formation during the material removal process. It is expected that the proposed framework can help adapt existing MD packages to support large scale MD simulations on desktop computers and thereby extend MD modelling to a wider range of applications and a wider group of researchers and engineers.

2. Review of existing methods

Though both CUDA and OpenCL support general purpose computing on GPU and they are similar to each other in principles, currently CUDA is more friendly to developers in terms of development tools and device drivers [12]. In this study the proposed MD framework was implemented using the CUDA API, and the following context would use the CUDA terminologies for describing the technical details.

The major concern for implementing large scale MD algorithms is to reduce the computational complexity from $O(N^2)$ to $O(N)$ by limiting the range of force evaluation to a small number of particles for each particle [28,29], usually within a cut-off distance (r_c) beyond which the interactions are small enough to be neglected [30]. For implementations on CPU, this is usually done by adopting a linked-cell algorithm or a combination of neighbor list and linked-cell [10]. For GPU implementations, the linked-cell algorithm is not appropriate since it means a lot of random memory operations which would dramatically decrease the performance [10]. Therefore, most GPU implementations adopt a cell-list algorithm [4,31] or a combination of cell-list and neighbor list [1,6,7,10,12,15,28,30]. There are also some other algorithms for implementing MD on GPU, targeting MD simulations in specific fields. For example, OpenMM adopted a block list algorithm to speed up the MD simulation of biosystems [16], for which the time length other than the number of particles is the primary concern [4].

For the existing cell-list algorithms or the combination of cell-list and neighbor list on GPU, a limitation is that the memory utilization is not optimized and as a result the maximum number of particles is limited [31]. For the cell-list algorithms, the atoms are located to a grid of cells with edge length equal to or slightly larger than r_c at each time step, such that the search of interacting atoms for each atom can be constrained to the 27 cells around that atom [32]. Usually a fixed amount of device memory enough to accommodate the maximum possible number of atoms for a single cell is allocated for each cell, for the purpose of allowing coalesced memory access [10]. It is evident that for systems with a lot of empty cells, a large amount of memory would be wasted [31]. Unfortunately, many systems contain highly irregular geometries and experience significant deformations during the simulations, which means that the number of cells to be allocated need to be far more than those really contain atoms. For the combination of cell-list and neighbor list algorithm, the neighbor lists further require a large amount of memory [12]. Assuming that the length of neighbor list for each atom is 250 and only the IDs of the neighbor atoms are stored in the neighbor list, this means 1000 bytes for the neighbor list of each atom and 1 GB for 1 million atoms. This makes it difficult to model large systems on single GPUs since the device memory is usually limited.

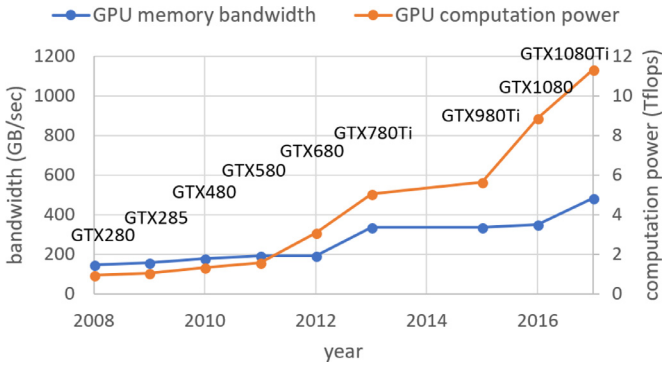

Fig. 1. Evolution of computation power and memory bandwidth of Nvidia GPUs

Though the adoption of neighbor list reduces the computational demand by decreasing the number of atoms that need to be checked for each atom, it poses heavy challenges to the memory bandwidth [12]. On one hand, it is difficult to reduce memory loading times by taking advantage of the shared memory, since every atom maintains its own neighbor list. Assuming the length of neighbor list is 250, each atom would need to be read from the device memory for 250 times during the force computation, while for the cell-list method one atom only needs to be read for 27 times. On the other hand, the loading of the positions of neighbor atoms is an uncoalesced memory access which would deteriorate the memory throughput. Therefore, it is hard to tell if the benefits from reduced computational demand can offset the loss from increased memory loading times and reduced throughput, especially when the memory bandwidth is becoming the bottle-neck in improving the computational capability of GPUs. Fig. 1 summarizes the evolution of computational power and memory bandwidth of NVIDIA's flagship GTX GPUs [33-39] in the past decade. It can be seen that the memory bandwidth in 2017 was only about 3 times of that in 2008, whereas the computational power has experienced a giant increase of over 11 times. Taking this factor into consideration, it is necessary to re-evaluate the effectiveness of the combination of cell-list and neighbor list on the latest GPUs.

3. Methodology

3.1. . Data structure

Though the proposed framework belongs to the cell-list type, its data structure is quite different with previous studies [31,40] for the purpose of enabling dynamic allocation. There are basically three types of data in the device memory as shown in Fig. 2, i.e. (i) the position, velocity and force lists, (ii) a list (“all cells list”) of structs storing the basic information of each cell in the simulation box, and (iii) a list (“atom cells list”) of structs that store the IDs and positions of atoms in each cell containing atoms, and a list (“atom cell ID list”) of the IDs of those cells containing atoms.

The position, velocity and force list just simply use the float4 datatype of CUDA. As the same with previous studies, the float4 is adopted instead of float3 to align the device memory to allow coalesced memory access, at the cost of wasting 4 bytes of memory space for each atom. In practice, the w component of the position is used to store a value representing the atom species when handling multi-element systems. The lengths of these lists are equal to the total number of atoms in the system, including ghost atoms when periodic boundary is applied. The sequence of the atoms in these lists is kept constant during the simulation, since the proposed method does not need to re-arrange the