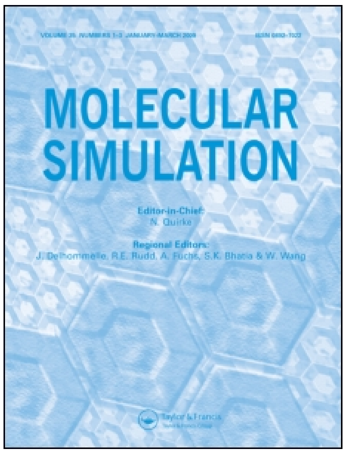

Molecular Simulation
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

# Molecular dynamics simulation of macromolecules using graphics processing unit

Ji Xu[ab]; Ying Ren[a]; Wei Ge[a]; Xiang Yu[bc]; Xiaozhen Yang[c]; Jinghai Li[a]
[a] State Key Laboratory of Multiphase Complex Systems, Institute of Process Engineering, Chinese Academy of Sciences, Beijing, P.R. China [b] Graduate University of the Chinese Academy of Sciences, Beijing, P.R. China [c] Beijing National Laboratory for Molecular Sciences, Joint Laboratory of Polymer Science and Materials, Institute of Chemistry, Chinese Academy of Sciences, Beijing, P.R. China

Online publication date: 10 December 2010

**To cite this Article** Xu, Ji , Ren, Ying , Ge, Wei , Yu, Xiang , Yang, Xiaozhen and Li, Jinghai(2010) 'Molecular dynamics simulation of macromolecules using graphics processing unit', Molecular Simulation, 36: 14, 1131 — 1140
**To link to this Article: DOI:** 10.1080/08927022.2010.506512
**URL:** http://dx.doi.org/10.1080/08927022.2010.506512

*Molecular Simulation*
Vol. 36, No. 14, December 2010, 1131–1140

# Molecular dynamics simulation of macromolecules using graphics processing unit

Ji Xu[ab], Ying Ren[a], Wei Ge[a]*, Xiang Yu[bc], Xiaozhen Yang[c] and Jinghai Li[a]

[a]*State Key Laboratory of Multiphase Complex Systems, Institute of Process Engineering, Chinese Academy of Sciences, P.O. Box 353, Beijing 100190, P.R. China;* [b]*Graduate University of the Chinese Academy of Sciences, Beijing 100039, P.R. China;* [c]*Beijing National Laboratory for Molecular Sciences, Joint Laboratory of Polymer Science and Materials, Institute of Chemistry, Chinese Academy of Sciences, Beijing 100190, P.R. China*

(*Received 10 March 2010; final version received 2 July 2010*)

Molecular dynamics (MD) simulation is a powerful computational tool to study the behaviour of macromolecular systems. However, many simulations in this field are limited in spatial or temporal scale by the available computational resource. In recent years, graphics processing units (GPUs) have provided unprecedented computational power for scientific applications. Many MD algorithms suit the multithread nature of GPU. In this paper, MD algorithms for macromolecular systems that run entirely on GPU are presented. For validation, we have performed MD simulations of polymer crystallisation with our GPU package, GPU_MD-1.0.5, and the results agree perfectly with computations on CPUs, meanwhile GPU_MD-1.0.5 achieves about 39 times speedup compared with GROMACS-4.0.5 on a single CPU core. Therefore, our single GPU code has already provided an inexpensive alternative for macromolecular simulations of traditional CPU clusters and will serve as a basis for developing parallel GPU programs to further speed up the computations.

**Keywords:** macromolecule; molecular dynamics; speedup; GPU; CUDA

## 1. Introduction

Macromolecular systems have been one of the most active areas of polymer science and molecular dynamics (MD) and simulation has proven to be a powerful computational tool to complement experimental results. However, the computationally intensive nature of MD algorithms and the limited computational horsepower available today make it difficult for current simulations to reach large spatio-temporal scales typical to macromolecular experiments, though several MD packages, such as GROMACS [1–4], NAMD [5], LAMMPS [6], can already run efficiently on distributed memory computer clusters. Fortunately, the graphics processing unit (GPU), originally designed for computationally intensive, highly parallel graphic operations, has become now programmable for general-purpose computations with the advent of convenient software development environments, such as the Compute Unified Device Architecture (CUDA) [7] from NVIDIA (Santa Clara, CA, USA). With CUDA, GPU can serve as an accelerator to CPU, executing a very large number of threads in parallel. Some MD algorithms, with atom as the smallest particle, are data-parallel and can be mapped to GPU conveniently with one thread dealing with one atom. Stone et al. [8] accelerated a non-bonded force calculation in NAMD codes with a spatial bin method which does not require the CPU to build neighbour lists. Anderson et al. [9] developed a general-purpose MD program named 'HOOMD', which is fully implemented on GPU except the part of bins updating. These techniques have shed some light on the simulation of macromolecular systems [10].

In this article, we implement MD simulations of macromolecular systems on a single GPU to reach a larger spatio-temporal scale so as to investigate the dynamics of polymer entanglement and crystallisation. We have found that the time consumed in memory coping process of bins updating increases with the size of the simulated systems. So, in our codes, all the MD processes including the bins-updating part are put into GPU, and angle potentials and dihedral potentials are also included to keep macromolecular conformations.

## 2. Model

In this paper, the macromolecular system considered constitutes of the polymer molecules of polyethylene (PE). The monomers are treated as two types of united atoms, representing $CH_2$ (C_32) and $CH_3$ (C_33), respectively. Each PE molecule consists of 150 monomers, resulting in 298 C_32 and 2 C_33 interaction sites. The Dreiding II force field [11] is adopted. The interactions include two parts, bonded interactions ($E_{\mathrm{bonded}}$) and non-bonded

*Corresponding author. Email: wge@home.ipe.ac.cn

ISSN 0892-7022 print/ISSN 1029-0435 online
© 2010 Taylor & Francis
DOI: 10.1080/08927022.2010.506512
http://www.informaworld.com

Table 1. Parameters of the force field.

| Atom | Type | $\sigma$ (nm) | $\varepsilon$ (kJ mol$^{-1}$) | |
|---|---|---|---|---|
| | C_32 | 14.027 | 0.3624 | |
| | C_33 | 15.035 | 0.3699 | |
| Bond | Type | $k_{ij}$ (kJ mol$^{-1}$ nm$^{-2}$) | $b_{ij}$ (nm) | |
| | C_32–C_32 | 292,880 | 0.153 | |
| | C_32–C_33 | 292,880 | 0.153 | |
| Angle | Type | $k_{ij}$ (kJ mol$^{-1}$ rad$^{-2}$) | $\theta_{ijk}$ (°) | |
| | C_32–C_32–C_32 | 418.40 | 109.471 | |
| | C_32–C_32–C_33 | 418.40 | 109.471 | |
| | C_33–C_32–C_32 | 418.40 | 109.471 | |
| Dihedral | Type | $k_\Phi$ (kJ mol$^{-1}$) | $\Phi_s$ (°) | $n$ |
| | C_32–C_32 | 4.184 | 0 | 3 |

interactions ($E_{\text{nb}}$). The former depends on the molecular structure and the latter depends on the distance between the atoms, that is

$$E = E_{\text{nb}} + E_{\text{bonded}}, \tag{1}$$

where $E_{\text{nb}}$ is calculated with the Lennard-Jones potential:

$$E_{\text{nb}}(r_{ij}) = 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right], \tag{2}$$

where $\varepsilon_{ij}$ is the depth of the potential well and $\sigma_{ij}$ is the finite distance at which the interparticle potential is zero.

Three types of interactions are considered in $E_{\text{bonded}}$, namely, bond stretch ($E_{\text{b}}$, two-body), bond-angle bend ($E_{\text{a}}$, three-body) and dihedral angle torsion ($E_{\text{d}}$, four-body), which are called bond (b), angle (a) and dihedral (d), respectively, for simplicity, that is

$$E_{\text{bonded}} = E_{\text{b}} + E_{\text{a}} + E_{\text{d}}, \tag{3}$$

where $E_{\text{b}}$ is expressed as a harmonic potential between atoms $i$ and $j$:

$$E_{\text{b}}(r_{ij}) = \frac{1}{2}k_{ij}^{b}(r_{ij} - b_{ij})^2, \tag{4}$$

where $k_{ij}^{b}$ is the force constant and $b_{ij}$ is the equilibrium distance between $i$ and $j$.

$E_{\text{a}}$ is represented by a harmonic potential on angle $\theta_{ijk}$:

$$E_{\text{a}}(\theta_{ijk}) = \frac{1}{2}k_{ij}^{a}\left(\theta_{ijk} - \theta_{ijk}^{0}\right)^2. \tag{5}$$

$k_{ij}^{a}$ is the force constant and $\theta_{ijk}^{0}$ is the equilibrium angle of atoms $i$–$j$–$k$.

$E_{\text{d}}$ is calculated as

$$E_{\text{d}} = k_{\phi}(1 + \cos(n\phi - \phi_s)), \tag{6}$$

where $\phi$, $n$ and $\phi_s$ represent the angle between the $i$–$j$–$k$ and $j$–$k$–$l$ planes, the periodicity and the equilibrium angle, respectively. Parameters in the equations are listed in Table 1.

## 3. GPU-based algorithms

The GPU-based algorithm for the model described above is developed by the Institute of Process Engineering (IPE) based on their previous work [12]. The general simulation procedure is illustrated in Figure 1. Leap-frog scheme [13] is adopted to integrate the equations of motion. The polymer system is simulated using an NVT ensemble and the extended ensemble Nosé–Hoover method [14–16] is used to control the temperature. For several reasons, such as force truncation and integration errors, the translation and rotation around the centre of mass would be inevitably generated and should be removed during the simulation.

To illustrate the algorithm, we define five variables, as shown in Table 2. The atom handled by the global thread is defined as home atom, and atomic information can be loaded from the global memory of GPU by GTIDX.

### *3.1 Binning the atoms and neighbour list generation*

To speed up the binning of atoms and the generating of neighbour list, we use the concept of 'charge group' in GROMACS [4]. In PE systems, one ethylene molecule is treated as one charge group. As our simulations usually involve a huge number of charge groups, grid search is much faster than simple search when generating the neighbour list [17], and is hence adopted. In this approach, the simulated domain is first divided into grids with the size of each dimension equal to the cut-off distance of the non-bonded force. All charge groups are then put into the

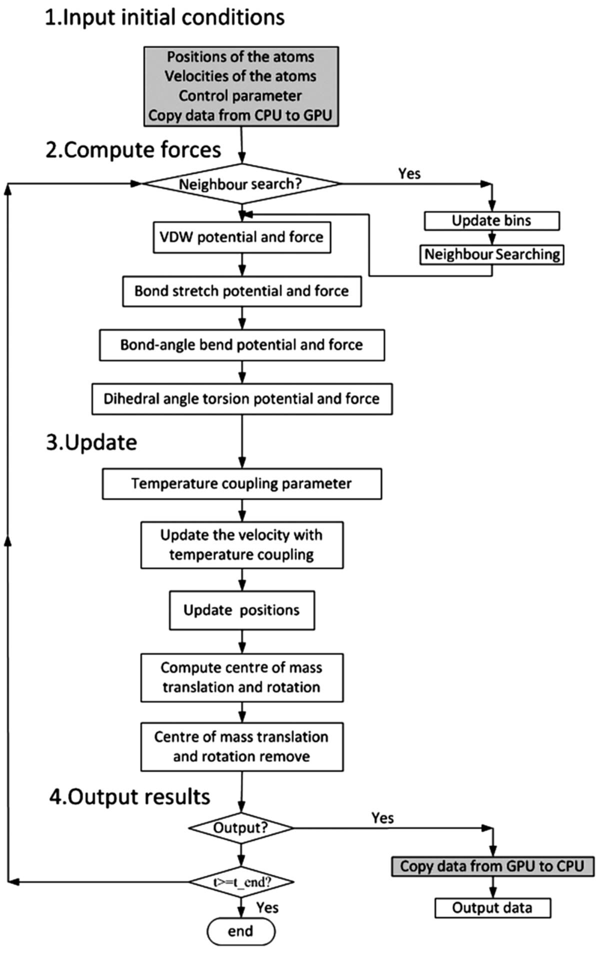
1.Input initial conditions
Positions of the atoms
Velocities of the atoms
Control parameter
Copy data from CPU to GPU
2.Compute forces
Neighbour search?
Yes
Update bins
Neighbour Searching
VDW potential and force
Bond stretch potential and force
Bond-angle bend potential and force
Dihedral angle torsion potential and force
3.Update
Temperature coupling parameter
Update the velocity with temperature coupling
Update positions
Compute centre of mass translation and rotation
Centre of mass translation and rotation remove
4.Output results
Output?
Yes
Copy data from GPU to CPU
Output data
t>=t_end?
Yes
end

Figure 1. Diagram of the whole MD procedure. Note: Shaded boxes involve data transfer between GPU and CPU.

corresponding grids according to their positions. In this way, a charge group only needs to search the charge groups in its own grid and 26 neighbouring grids, totally 27 grids.

To bin the charge groups, it is natural and preferred to assign one charge group to one thread. Before putting the charge groups into the bins, the positions of the charge groups should be computed. Then, the bin index of the charge group is computed. After that, the charge groups are put into the bins. The atomic function is used to retrieve the size of charge groups which are already in the bins. In this way, the problem of writing the same GPU global memory can be avoided.

Table 2. Abbreviations of variables.

| Variable | Description |
|---|---|
| GAIDX | Global atom index of the system |
| GTIDX | Global thread index of a GPU kernel, GTIDX = GAIDX-1 |
| MAIDX | Atom index in one molecule, ranging from 1 to NAOM |
| MIDX | Molecule index |
| NAOM | Number of atoms in one molecule |

A neighbour list of each charge group should be constructed after binning charge groups. The cut-off distance is set to a value large enough so that non-bonded potentials can be fully expressed. A two-dimensional array ($N \times M$) in the global memory of GPU is allocated to store the neighbour list. Here, $N$ is the number of charge groups plus the padded number given by the cudaMallocPitch function, and $M$ is the maximum number of neighbours for each charge group. The number of neighbours of every charge group is different, especially for heterogeneous systems, which makes it a luxury way to use memory, but the global memory of GPU will read in a coalesced way [7].

In the neighbour list searching kernel, one thread block corresponds to one gird, and one thread corresponds to one charge group in the bin. One thread block deals with its own bin and the neighbouring 26 bins. The positions and charge group indices are loaded from global memory to shared memory in advance, and then each thread uses the data in shared memory to search the neighbouring charge groups. After that, the atom's neighbour list is generated. Each thread is set to deal with one charge group called i-cg and its neighbour charge groups called j-cg. Every atom in i-cg is compared with the atoms in each j-cg to determine whether the latter atoms are exclusions of the former atom.

In Dreiding II force field [11], only the first and second bonded neighbours are excluded from the non-bonded interactions, so only one int4 is needed to store the exclusions of one atom. Although reading int4 data from the global memory of GPU can increase the speed, not all the force fields only exclude the first and second bonded neighbours. For the general purpose, two arrays are used to store the exclusions. One is used to store the range of exclusions of one atom, which is called index array. The other stores the exclusions themselves, which is called content array. One exclusion data structure contains all the exclusions of one type of molecule. Each thread computes MAIDX and MIDX according to GAIDX. Then, MAIDX is used to get the start and end positions of the exclusions of the atom from the index array which are used in the content array to load the atom's exclusions into registers from global memory, and then these data are added by NAOM*MIDX to get the global exclusions of the atom. Details of these two arrays are given in Section 3.3.

### 3.2 *Non-bonded interactions*

After generating the neighbour list, computations of non-bonded interactions are carried out in GPU with one thread dealing with one atom. GTIDX is computed first according to the executing environment of the GPU kernel. It is also GAIDX-1. Data (position, type, number of neighbours) of the home atom are then loaded into registers according to GTIDX, followed by the inner loop of the non-bonded calculation. The parameters of Equation (2) are stored in the GPU constant memory and are loaded according to the types of the interacting atoms.

### *3.3 Bond, angle and dihedral interactions*

The bond, angle and dihedral interactions of a certain molecule are all fixed throughout the simulation and are thus computed based on fixed lists. Usually, interaction lists are generated according to the types of the molecules at the beginning of the simulation. For the PE system, it involves only one type of molecule. Bond interactions are treated pair-wisely like exclusions. Two arrays, index and content, are used too, plus one more variable – the number of atoms in the molecule. A simple molecule in Figure 2(a) is taken to describe the details of constructing the ‘index’ and ‘content’ arrays in Figure 2(b).

The bond interactions are computed according to the constructed bond information. It is convenient to map this work into GPU kernel with one thread dealing with one atom. But, there is a significant difference between the algorithms of GPU and CPU. In CPU, all bond interactions are computed in one for-loop and the force can be given to both atoms of the bond. But in GPU, to make use of the multithread nature of GPU, every thread should compute all the bonds of the home atom, so actually the computational cost is doubled.

As to angle and dihedral interactions, the inherent multi-body feature makes them much more complicated than bond interactions. Take the angle computation for example: it is different from the CPU algorithm, which only contains indices of the three atoms in an angle. For computation of one angle interaction, four variables, the angle index which is used to choose the parameters in Equation (5), the position of the home atom in the angle and the indices of the two neighbouring atoms are needed. It should be noted that the other two atoms must be stored from the first to the third for each angle. So, one element in the angle content array has four variables, which are different from the bond content array. The index array and the number of atoms in the molecule have the same usage. According to the definitions above, the angle information for the molecule in Figure 2(a) can be depicted as shown in Figure 2(c). After generating the angle information in CPU, it is copied to the global memory of GPU. Then, the angle interactions can be calculated in the same way as the bonded interactions. It should also be noted that, in the CPU algorithm, usually one loop deals with one angle and then forces are added to the atoms composing the angle. But in the GPU algorithm, similar to the calculation of bond interaction, the number of floating point operations is tripled.

Dihedral interactions can be treated in the same way as angle interactions, as it involves four atoms, the floating point operations are quadrupled compared to the CPU algorithm.

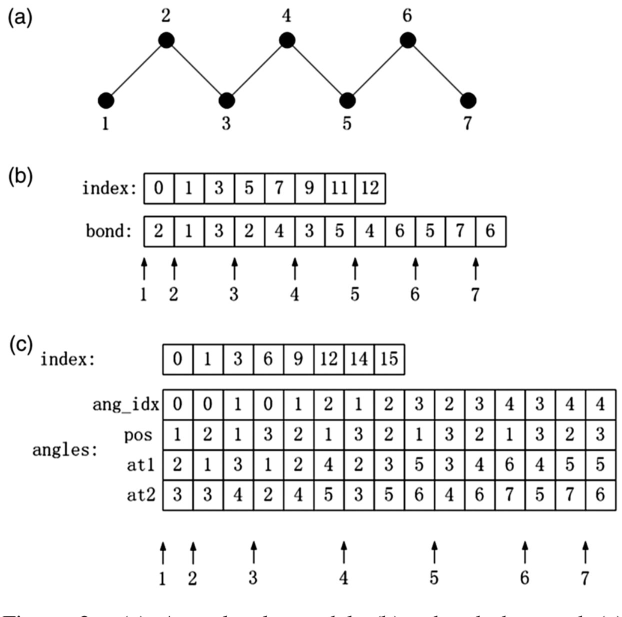
(a)
2 4 6
1 3 5 7
(b)
index: 0 1 3 5 7 9 11 12
bond: 2 1 3 2 4 3 5 4 6 5 7 6
1 2 3 4 5 6 7
(c)
index: 0 1 3 6 9 12 14 15
angles:
ang_idx 0 0 1 0 1 2 1 2 3 2 3 4 3 4 4
pos 1 2 1 3 2 1 3 2 1 3 2 1 3 2 3
at1 2 1 3 1 2 4 2 3 5 3 4 6 4 5 5
at2 3 3 4 2 4 5 3 5 6 4 6 7 5 7 6
1 2 3 4 5 6 7

Figure 2. (a) A molecule model, (b) t_bond data and (c) t_angles data.

### *3.4 Update configurations*

The leap-frog scheme [13] together with Nosé–Hoover temperature coupling algorithm is adopted to update the configurations. It is straightforward to map the leap-frog scheme into GPU kernel by using one global thread to update the information of one atom. The temperature coupling algorithm, however, needs the kinetic energy of the whole system, which makes it comparatively difficult to be realised in GPU. Here, we propose a method to store the kinetic energy of each atom in a temporary array, followed by a summation of the energies using the method in CUDA SDK [18] except the second stage. The function _threadfence() is used so as to put the parallel summation into one GPU kernel. The thread with identity number 0 is used to determine whether its thread block is used in the second stage. Only one thread block is actually used in the second summation stage.

Removal of centre-of-mass (COM) translation and rotation is also carried out. First, COM of translation and rotation are summed over all the atoms, respectively, like the operations of summing the kinetic energy, then both of them are removed over all the atoms.

## 4. Performance evaluations

The performance of the simulation method discussed above is evaluated on the Mole-8.7 system [19] at IPE. The

Table 3. Execution environment.

| Item | Description |
|---|---|
| Workstation | HP XW8600 |
| CPU | Two Intel® Xeon® E5430 2.66 GHz |
| Memory | Eight 667 MHz 2 GB JEDEC RAM |
| GPU | One Nvidia® Tesla C2050 |
| Operating system | CentOS5.2 |
| C/C++ compiler | GCC-4.1.2 |
| GPU driver | Devdriver_3.0_linux_64_195.36.15.run [20] |
| CUDA® SDK | gpucomputingsdk_3.0_linux.run [21] |
| CUDA® toolkit | cudatoolkit_3.0_linux_64_rhel5.3.run [22] |

execution environment is specified in Table 3. Our GPU code (GPU_MD-1.0.5) is compared with the CPU program GROMACS-4.0.5 [4] on all parts of the algorithm described in Section 3. It is also compared with another GPU program HOOMD-0.9.0. All the three programs are tested with single-precision floating point arithmetic.

To evaluate the performance of each algorithm, seven PE systems with different volume ($V$) and different numbers of PE molecules ($N$), as listed in Table 4, are simulated and different parts of the algorithms are evaluated, respectively. Every system was simulated for 100,000 steps. The times reported are all average execution times of each procedure.

### 4.1 Generating neighbour list and computation of non-bonded interactions

In GROMACS and GPU_MD-1.0.5, neighbour list generation has two parts, binning the charge groups and searching the neighbour charge groups. While in HOOMD, it consists of binning the atoms and searching for neighbour atoms. However, the two parts are integrated in GROMACS which are all included in the 'NB list' part in Table 5. Moreover, the times of HOOMD and GPU_MD-1.0.5 are given in the 'bin' and 'NB list' parts separately. The results show that using the charge group as the searching element can speed up the neighbour list updates.

Non-bonded interactions are computed after generating the neighbour list. The computation times of the seven systems by different programs are also given in Table 5. Apparently, both HOOMD and our codes are much faster than the CPU program.

GROMACS calculates all interactions with the buffer region, even though they are all zero. And neighbour lists are usually updated at fixed intervals [4], in our case, every 10 steps. When compared with GROMACS, the cut-off distance and the skin distance of neighbour list generation are kept the same, which are 0.9 and 0.0 nm, respectively. Moreover, all interactions within the buffer region are computed. While compared with HOOMD, the skin distance is set to 0.1 nm. Only the interactions in the cut-off region are calculated.

### 4.2 Computation of bonded interactions

The running times of three types of bonded interactions for GROMACS, HOOMD and our GPU code are given in Table 5. It can be found that HOOMD and our GPU kernels are much more efficient than corresponding GROMACS kernels. For GROMACS kernels, although the numbers of bond, angle and dihedral interactions decrease slightly, the running times increase significantly due to the increase in the interaction complexities. Meanwhile, the times increase with the size of the system. However, for HOOMD and our GPU kernels, the running times do not change much for the three different types of interactions, or for different sizes of the system, indicating that the intensities of computations are very close to each other.

### 4.3 Updating configurations

Leap-frog integrating consumes most time in updating the molecular configurations. The leap-frog scheme can be parallelised as easily as the bonded interaction algorithms. As shown in Table 5, the leap-frog GPU kernel also gets very good speedup as the bonded GPU kernels.

To remove the COM translation and rotation, parallel summation on GPU is performed. Table 5 suggests that the performance is not as good as the updating algorithm, which can be ascribed to the two steps of summation and the low parallelism of the second step with only one thread block.

### 4.4 Overall performance of the whole program

After testing the performance of every step, the overall performance of the program is evaluated. The GPU/CPU speedup ratios as a function of the atom numbers are depicted in Figure 3(a). In both CPU and GPU programs, the computation of interactions takes most of the time,

Table 4. Configurations of seven test systems.

| System | I | II | III | IV | V | VI | VII |
|---|---|---|---|---|---|---|---|
| $V$ (nm$^3$) | $10 \times 10 \times 16$ | $16 \times 16 \times 24$ | $20 \times 20 \times 32$ | $24 \times 24 \times 40$ | $30 \times 30 \times 40$ | $34 \times 34 \times 40$ | $40 \times 40 \times 40$ |
| $N$/1000 | 19.2 | 64.8 | 117.6 | 243.0 | 363.0 | 507.0 | 675.0 |

Table 5. Execution times (ms) of MD procedures[a].

| System | I | II | III | IV | V | VI | VII |
|---|---|---|---|---|---|---|---|
| GROMACS-4.0.5 (1 CPU core) | | | | | | | |
| NB list | 20.870 | 70.445 | 124.246 | 257.868 | 385.575 | 545.796 | 723.741 |
| Non-bonded | 7.637 | 24.544 | 44.715 | 92.989 | 138.969 | 193.915 | 258.405 |
| Bond | 0.855 | 2.909 | 5.258 | 11.051 | 16.518 | 23.012 | 30.899 |
| Angle | 2.942 | 9.951 | 18.096 | 37.288 | 56.063 | 78.259 | 103.933 |
| Dihedral | 6.378 | 21.574 | 39.078 | 80.902 | 120.823 | 168.509 | 224.627 |
| Updating | 1.993 | 6.771 | 12.637 | 27.623 | 41.467 | 57.670 | 76.868 |
| COM | 0.586 | 1.972 | 3.578 | 7.732 | 12.384 | 18.310 | 24.928 |
| HOOMD-0.9.0 | | | | | | | |
| Bin | 0.945 | 2.202 | 3.883 | 7.071 | 8.239 | 10.083 | 14.937 |
| NB list | 2.846 | 9.139 | 17.329 | 35.285 | 51.619 | 75.956 | 113.556 |
| Non-bonded | 0.451 | 1.006 | 1.682 | 3.207 | 4.741 | 6.490 | 8.214 |
| Bond | 0.091 | 0.127 | 0.162 | 0.254 | 0.337 | 0.440 | 0.562 |
| Angle | 0.144 | 0.266 | 0.400 | 0.722 | 1.023 | 1.394 | 1.818 |
| Dihedral | 0.150 | 0.297 | 0.458 | 0.841 | 1.202 | 1.649 | 2.158 |
| Updating[b] | 0.279 | 0.425 | 0.593 | 0.991 | 1.361 | 1.824 | 2.352 |
| GPU_MD-1.0.5 | | | | | | | |
| Bin[c] | 0.156 | 0.298 | 0.432 | 0.640 | 0.881 | 1.213 | 1.610 |
| NB list[c] | 2.077 | 6.819 | 12.540 | 25.839 | 37.994 | 52.291 | 70.851 |
| Non-bonded[c] | 0.266 | 0.720 | 1.147 | 2.293 | 3.373 | 4.715 | 6.204 |
| Bin[d] | 0.156 | 0.316 | 0.430 | 0.644 | 0.877 | 1.181 | 1.565 |
| NB list[d] | 2.234 | 7.320 | 13.399 | 27.634 | 40.746 | 55.976 | 75.727 |
| Non-bonded[d] | 0.340 | 0.967 | 1.546 | 3.148 | 4.643 | 6.497 | 8.555 |
| Bond | 0.083 | 0.120 | 0.165 | 0.274 | 0.379 | 0.500 | 0.649 |
| Angle | 0.101 | 0.187 | 0.281 | 0.497 | 0.713 | 0.954 | 1.250 |
| Dihedral | 0.101 | 0.187 | 0.281 | 0.497 | 0.713 | 0.954 | 1.250 |
| Updating | 0.081 | 0.131 | 0.191 | 0.330 | 0.460 | 0.617 | 0.802 |
| COM | 0.139 | 0.238 | 0.359 | 0.632 | 0.905 | 1.215 | 1.595 |

[a] Bin and neighbour list updates are executed every 10 steps, so the times given here are all average execution times over 100,000 steps. [b] The time of updating in HOOMD-0.9.0 includes summation of the force which is scattered in other parts of GPU_MD-1.0.5. [c] These data are used for comparison with GROMACS. [d] These data are used to compare with HOOMD-0.9.0.

usually more than 50%. Although the speedup of neighbour list generation is only about 10 times of GPU over CPU, the speedup of the interaction procedure is very high. So, the overall speedup of GPU_MD-1.0.5 depicted in Figure 3(a) is higher than the GPU neighbour list generation procedure.

The GPU_MD-1.0.5 program is also compared against the GROMACS-4.0.5 parallel program running on one node with eight processing cores. Domain decomposition cannot be used with angular type of COM removal in this version of GROMACS, so particle decomposition is adopted instead. Also, the GPU speedup ratio against parallel CPUs is depicted in Figure 3(a). It can be seen that the speedup ratios are still going up with the system size. The results of GPU_MD-1.0.5 compared with HOOMD are given in Figure 3(b).

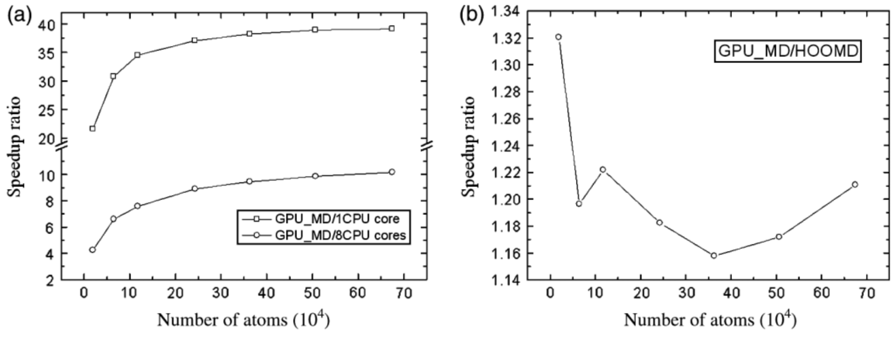

Figure 3. Overall speedup ratios of GPU_MD/1 CPU core, (a) GPU_MD/8 CPU cores and (b) GPU_MD/HOOMD. Note: the whole times of one step of GROMACS on eight CPU cores are computed from the times given by GROMACS log files, and the time of HOOMD does not include removing of COM.

Table 6. Simulation cases for interpenetration and crystallisation.

| Case | Program | Atom number | Temperature (K) | Simulation time (ns) | ICF of initial structure |
|---|---|---|---|---|---|
| Interpenetration | | | | | |
| IP_C | GROMACS-4.0.5 | 45,000 | 1000 | 30 | – |
| IP_G | GPU_MD-1.0.5 | 360,000 | | 30 | – |
| Crystallisation | | | | | |
| C_1 | GROMACS-4.0.5 | 45,000 | | 12 | Low |
| C_2 | | | 600 | | High |
| G_1 | GPU_MD-1.0.5 | 360,000 | | 50 | Low |
| G_2 | | | | | High |

Although GPU_MD-1.0.5 speeds up about 39 times of single process GROMACS-4.0.5, there are still some aspects to be optimised. First, in some algorithms of GPU_MD-1.0.5, the global memory reading is not in a coalesced way such as the exclusions data reading. In the inner loop of neighbour list generation, every thread must load the exclusions of the home atom, and some threads may read the same memory data of the GPU global memory. If the GPU shared memory is large enough, all the exclusions of the atoms processed by one thread block can be read into shared memory in advance. Second, branching operations should be minimised. In the calculations of angle and dihedral interactions, the interaction order must be defined according to the position of the angle or dihedral the atom locates, so several branching instructions have to be used in each thread, which affects the efficiencies of GPU kernels considerably. If writing into the same global memory of GPU without conflict can be realised in the future generation of GPU, and one angle or dihedral interaction will be managed by one thread, this problem will be solved.

## 5. Analysis of simulation results and validation of the program

In order to validate our GPU computation, the simulation results from two programs for the same case are analysed in detail. Yu et al. [10] have simulated the interpenetration and crystallisation processes of a PE system with 150 macromolecules composing 150 beads each using GROMACS 3.3 [3] on clusters with Xeon and Itanium CPUs. In this paper, the system is enlarged by eight times to 360,000 (1200 macromolecules composing 300 beads each) atoms. Two interpenetration cases and four crystallisation cases are simulated and the detailed information is listed in Table 6.

As in our previous work [10], the interpenetration process of the PE chains is performed first. The autocorrelation function (ACF) of end-to-end vectors of IP_G system are calculated and compared with the IP_C system in Ref. [10]. The result is plotted in Figure 4(a). It can be found that in both IP_G and IP_C systems, the ACF of end-to-end vectors declines to lower than 0.1 after about 5 ns, indicating the same relaxation time in both systems. After being equilibrated, the conformation distribution of the IP_G system is compared to the IP_C system and shown in Figure 4(b). One can find that they are almost overlapped, so that the bonded microstructure simulated by GPU_MD-1.0.5 is correct. Meanwhile, the normalised radial distribution functions (RDF) of both IP_G and IP_C systems are compared (Figure 5), and it can be found that the appearance position of peaks on the RDF curve in the IP_G system is just the same as the IP_C system, indicating the correctness of non-bonded microstructure obtained by GPU_MD-1.0.5. Furthermore, other properties of the interpenetration are calculated for both systems.

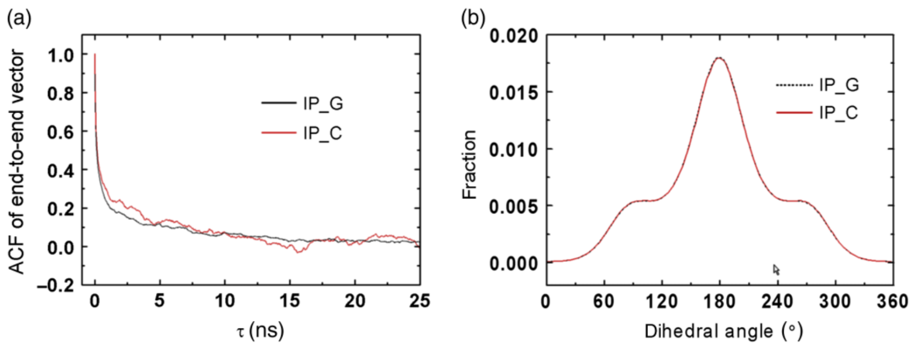

Figure 4. (a) ACF of end-to-end vectors for both IP_G and IP_C systems. (b) Conformation distribution after the interpenetration was equilibrated.

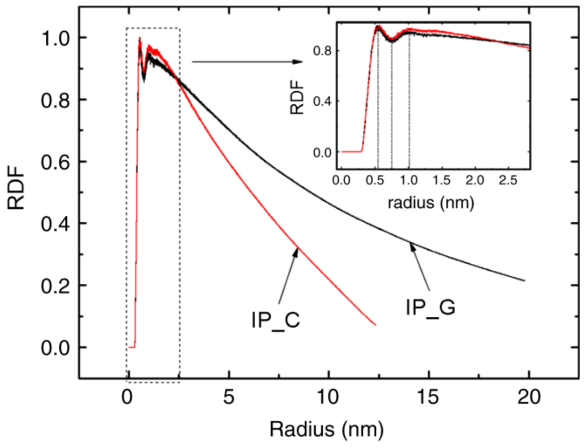

Figure 5. RDF of the amorphous coils in both IP_C and IP_G systems.

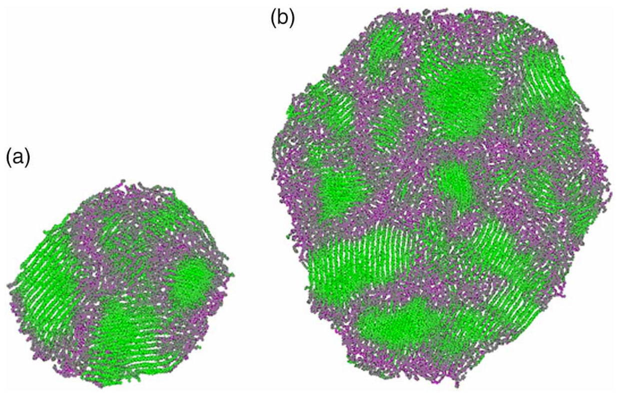

Figure 7. Distribution of crystalline domains and amorphous region by SOP images in the central slice at 2 ns of the crystallisation processes in both systems. The green region represents the crystalline domains while the purple region represents the amorphous region. (a) Sample C1 and (b) sample G1.

Figure 6(a) shows the interchain contact fraction (ICF) during the interpenetration process in both systems. The ICF [10,23] is thought to be the structural property of the entanglement in the polymer system. This figure shows that both systems have almost the same ICF plateau. Figure 6(b) shows the average radius of gyration ($R_g$) of all chains during the interpenetration in both IP_G and IP_C systems. With confidence, the max $R_g$ in the IP_G system is slightly larger than that in the IP_C system, indicating that the final coil of the IP_G system is even larger than the IP_C system and less surface confinement is presented. All the comparisons above show that GPU_MD-1.0.5 can achieve the same result as the GROMACS/CPU package for the high-temperature simulation.

When the temperature decreases, crystallisation will take place in PE. The crystallisation processes starting from initial structures with different ICF are also simulated in the present work as what have been performed previously [10]. Table 6 shows the two different samples in the crystallisation simulation by GPU_MD-1.0.5. The ICF of the initial structure is low in the G_1 system while high in the G_2 system. The simulation takes a much longer time (50 ns) than the C_1 (the L_3 in Ref. [10]) and C_2 (the H_1 in Ref. [10]) systems. Figure 7 shows the distribution of crystalline domains in the C_1 and G_1 systems by the SOP imagination method [10]. The green points in this figure represent the crystalline domains in the central slices at 2 ns of the crystallisation processes. It can be seen from these images that the nuclei or the crystallisation domains are similar in size in both systems. Further quantificational analysis is taken. The average stem length for C_1, C_2, G_1 and G_2 are plotted in Figure 8(a). For the G_1 and G_2 systems, the final stem length increases to about 13 bonds and the systems with initial lower ICF structures (G_1) would perform a slightly faster increase in the stem length at first but finally go to the similar value with the other one. The final values for G_1 and G_2 are nearly the same as the C_1 and C_2, and the same trend that the system with lower ICF initial structure would perform faster increase first can be found in C_1 and C_2. Moreover, Figure 8(b) shows the variation of fraction of *trans*-conformation during the crystallisation processes in these four systems, and Figure 8(c) shows the variation of crystallinity during the crystallisation processes. In each figure, the same final values for G_1, G_2 as C_1, C_2 can be clearly found. Also, for the variation of the crystallinity, the systems with

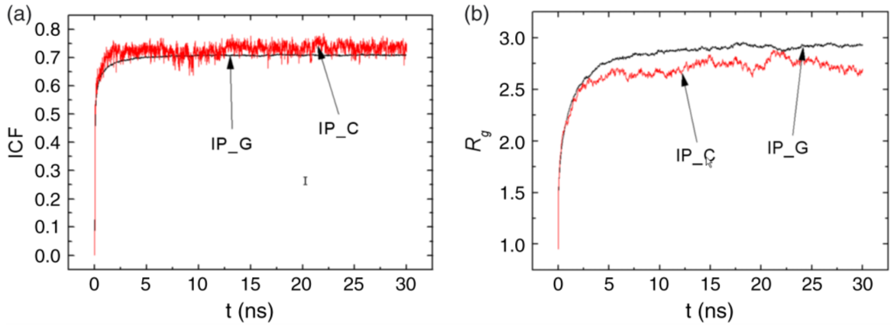

Figure 6. (a) ICF during the interpenetration process in both systems. (b) Variation of radius of gyration in both systems.

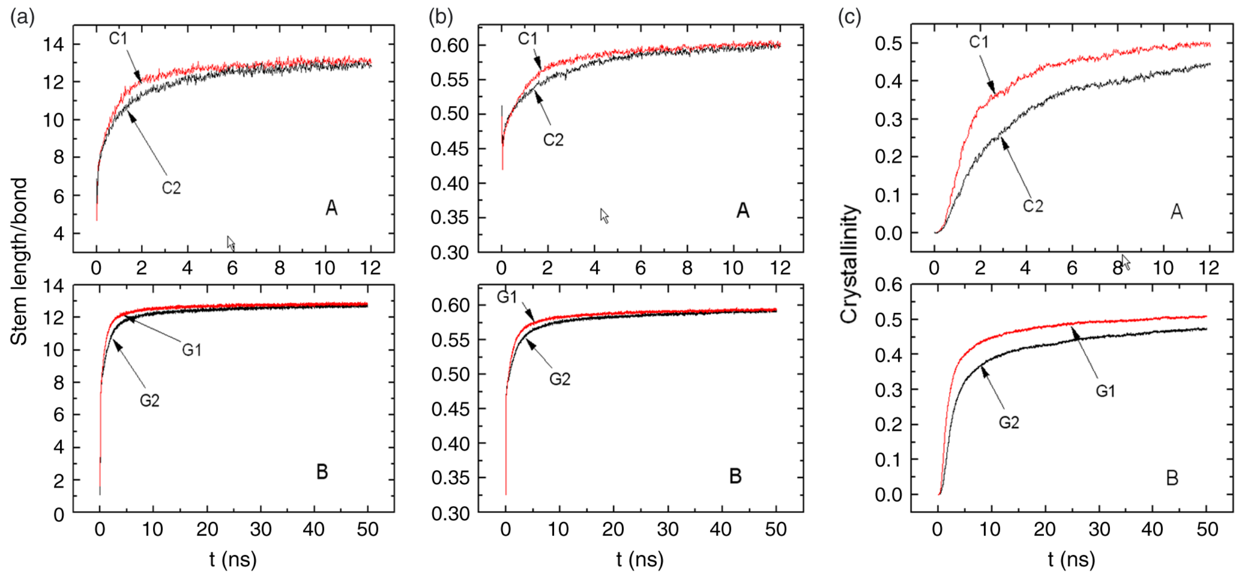

Figure 8. (a) Average stem length during the crystallisation processes. (b) Fraction of *trans*-conformation during the crystallisation processes. (c) Variation of crystallinity with time.

lower initial structures (C_1, G_1) can perform higher value all the time than the systems with higher initial structures (C_2, G_2). While for the variation of the *trans*-conformation, the systems with lower initial (C_1, G_1) structures only perform higher value at first and later became the same as the systems with higher initial structures (C_2, G_2). For the tests above, the thermodynamics properties obtained by the GPU program are the same as those obtained by the GROMACS-4.0.5 simulation. Thus, GPU_MD-1.0.5 is convinced to be valid for the simulation of crystallisation.

## 6. Conclusions and perspective

In this paper, we presented a macromolecular MD simulation fully implemented on GPU which reduces overheads of transferring data between GPU and CPU significantly. The performance of our GPU package, GPU_MD-1.0.5, is compared against GROMACS-4.0.5 and HOOMD. For simulations of polymer crystallisation within a wide range of number of atoms, GPU_MD-1.0.5, on a NVIDIA C2050 GPU runs about 39 times faster than GROMACS-4.0.5 on a single core of Intel Xeon(R) E5430 2.66 GHz CPU, up to about nine times faster than eight CPU cores in one node. The polymer crystallisation phenomenon is observed and the physical features studied are reasonable. For polar linear polymer (where neighbouring particles along the polymer chain have alternate charges and are governed by long-range electrostatic interactions) systems [24], the particle-mesh-Ewald method of our previous work [25] can be integrated. However, for MD simulations of even larger systems, one GPU core is incompetent. Parallel GPU computations which use the Message Passing Interface library is under investigation.

## Acknowledgements

This work is supported by the National Natural Science Foundation of China under the Grants Nos 20874107, 20821092 and 20821092 and Chinese Academy of Sciences under the Grants Nos KJCX2-SW-L08 and KGCX2-YW-124.

## References

[1] H.J.C. Berendsen, D. Van der Spoel, and R. Vandrunen, *Gromacs – A message-passing parallel molecular-dynamics implementation*, Comput. Phys. Commun. 91 (1995), pp. 43–56.
[2] E. Lindahl, B. Hess, and D. van der Spoel, *GROMACS 3.0: A package for molecular simulation and trajectory analysis*, J. Mol. Model. 7 (2001), pp. 306–317.
[3] D. Van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A.E. Mark, and H.J.C. Berendsen, *GROMACS: Fast, flexible, and free*, J. Comput. Chem. 26 (2005), pp. 1701–1718.
[4] B. Hess, D. van der Spoel, and E. Lindahl, *GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation*, J. Chem. Theory Comput. 4 (2008), pp. 435–447.
[5] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kale, and K. Schulten, *Scalable molecular dynamics with NAMD*, J. Comput. Chem. 26 (2005), pp. 1781–1802.
[6] S. Plimpton, *Fast parallel algorithms for short-range molecular-dynamics*, J. Comput. Phys. 117 (1995), pp. 1–19.
[7] NVIDIA, *CUDA Programming Guide*, NVIDIA, Santa Clara, CA, 2008.
[8] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, and K. Schulten, *Accelerating molecular modeling applications with graphics processors*, J. Comput. Chem. 28 (2007), pp. 2618–2640.
[9] J.A. Anderson, C.D. Lorenz, and A. Travesset, *General purpose molecular dynamics simulations fully implemented on graphics processing units*, J. Comput. Phys. 227 (2008), pp. 5342–5359.

[10] X. Yu, B. Kong, and X.Z. Yang, *Molecular dynamics study on the crystallization of a cluster of polymer chains depending on the initial entanglement structure*, Macromolecules 41 (2008), pp. 6733–6740.

[11] S.L. Mayo, B.D. Olafson, and W.A. Goddard, *Dreiding – A generic force-field for molecular simulations*, J. Phys. Chem. 94 (1990), pp. 8897–8909.

[12] F. Chen, W. Ge, C. Hou, B. Li, J. Li, X. Li, F. Meng, W. Wang, X. Wang, J. Xu, Q. Xiong, and Y. Zhang, *Multi-scale Discrete Simulation Parallel Computing Base on GPU*, Chinese Science Press, Beijing, 2009.

[13] L. Verlet, *Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules*, Phys. Rev. 159 (1967), pp. 98–112.

[14] S. Nosé, *A molecular-dynamics method for simulations in the canonical ensemble*, Mol. Phys. 52 (1984), pp. 255–268.

[15] S. Nosé, *A molecular dynamics method for simulations in the canonical ensemble (Reprinted from Molecular Physics, vol 52, pg 255, 1984)*, Mol. Phys. 100 (2002), pp. 191–198.

[16] W.G. Hoover, *Canonical dynamics – Equilibrium phase-space distributions*, Phys. Rev. A 31 (1985), pp. 1695–1697.

[17] D.C. Rapaport, *The Art of Molecular Dynamics Simulation*, Cambridge University Press, Cambridge, 2004.

[18] M. Harris, *NVIDIA Developer Technology of project reduction in cudatoolkit 2.1*, 2009: NVIDIA.

[19] F. Chen, W. Ge, L. Guo, X. He, B. Li, J. Li, X. Li, X. Wang, and X. Yuan, *Multi-scale HPC system for multi-scale discrete simulation – Development and application of a supercomputer with 1 Petaflops peak performance in single precision*, Particuology 7 (2009), pp. 332–335.

[20] NVIDIA, *CUDA Driver*, NVIDIA, Santa Clara, CA, 2010.

[21] *CUDA Software Development Kit*, NVIDIA, Santa Clara, CA, 2010.

[22] *CUDA Toolkit*, NVIDIA, Santa Clara, CA, 2010.

[23] Z.Q. Zhang and X.Z. Yang, *The effect of chain interpenetration on an ordering process in the early stage of polymer crystal nucleation*, Polymer 47 (2006), pp. 5213–5219.

[24] R.H. Gee, N. Lacevic, and L.E. Fried, *Atomistic simulations of spinodal phase separation preceding polymer crystallization*, Nat. Mater 5 (2006), pp. 39–43.

[25] J. Xu, W. Ge, Y. Ren, and J. Li, *Implementation of Particle-Mesh Ewald (PME) on Graphics Processing Units*, Chin. J. Comput. Phys. (2009).