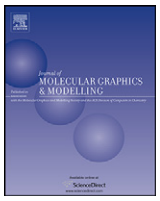
Journal of Molecular Graphics and Modelling 41 (2013) 1–11

Contents lists available at SciVerse ScienceDirect

Journal of Molecular Graphics and Modelling

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



Algorithms of GPU-enabled reactive force field (ReaxFF) molecular dynamics

Mo Zheng[a,b], Xiaoxia Li[a,*], Li Guo[a,**]

[a] State Key Laboratory of Multiphase Complex Systems, Institute of Process Engineering, Chinese Academy of Sciences, Beijing 100190, PR China
[b] Graduate University of Chinese Academy of Sciences, Beijing 100049, PR China

ARTICLE INFO

Article history:
Received 5 November 2012
Received in revised form 1 February 2013
Accepted 2 February 2013
Available online 10 February 2013

Keywords:
ReaxFF
GPU computing
GMD-Reax
Reactive molecular dynamics
Lammps

ABSTRACT

Reactive force field (ReaxFF), a recent and novel bond order potential, allows for reactive molecular dynamics (ReaxFF MD) simulations for modeling larger and more complex molecular systems involving chemical reactions when compared with computation intensive quantum mechanical methods. However, ReaxFF MD can be approximately 10–50 times slower than classical MD due to its explicit modeling of bond forming and breaking, the dynamic charge equilibration at each time-step, and its one order smaller time-step than the classical MD, all of which pose significant computational challenges in simulation capability to reach spatio-temporal scales of nanometers and nanoseconds. The very recent advances of graphics processing unit (GPU) provide not only highly favorable performance for GPU enabled MD programs compared with CPU implementations but also an opportunity to manage with the computing power and memory demanding nature imposed on computer hardware by ReaxFF MD. In this paper, we present the algorithms of GMD-Reax, the first GPU enabled ReaxFF MD program with significantly improved performance surpassing CPU implementations on desktop workstations. The performance of GMD-Reax has been benchmarked on a PC equipped with a NVIDIA C2050 GPU for coal pyrolysis simulation systems with atoms ranging from 1378 to 27,283. GMD-Reax achieved speedups as high as 12 times faster than Duin et al.'s FORTRAN codes in Lammps on 8 CPU cores and 6 times faster than the Lammps' C codes based on PuReMD in terms of the simulation time per time-step averaged over 100 steps. GMD-Reax could be used as a new and efficient computational tool for exploiting very complex molecular reactions via ReaxFF MD simulation on desktop workstations.

1. Introduction

Molecular dynamics based on classical mechanics (classical MD or MD) is a basic and popular molecular modeling method that is widely applied in biochemistry, biophysics and material science. But classical MD only describes physical elastic collision between atoms with static bonds and fixed partial charges, which are not feasible for chemical reactive systems. Density functional theory (DFT) is a quantum mechanical (QM) modeling method used in physics and chemistry to investigate the electronic structure of many body systems with high accuracy when involved in chemical bonding [1]. But DFT is extremely computationally intensive and is usually applied to systems containing tens or hundreds of atoms and picoseconds simulation timeframes [2].

Motivated by restrictions of the two approaches mentioned above for simulation of molecular systems involving chemical bonding, a number of recent efforts have been in progress for bridging the gap between quantum chemistry and classical molecular dynamics. Among these efforts, reactive force field (ReaxFF) developed by van Duin et al. [3] opened up a new pathway for studying molecular systems with chemical reactions. ReaxFF molecular dynamics (ReaxFF MD) presents the movement rule of chemical systems based on the bond order (BO) concept [4,5], which has been proven to be a smooth transition from non-bonded to single-, double- and triple-bonded system with all connectivity-dependent interactions to be the bond order dependent to ensure that the energy contribution disappears upon bond dissociation [6]. Since the bond orders in ReaxFF are updated in each time-step iteration, the connectivity of the system can continuously change with time [7]. ReaxFF MD allows the simulation of larger molecular reactive systems with acceptable accuracy when validated with DFT [3,6] and demonstrates its potential in simulating very complex systems on large scale supercomputers for up to millions of atoms with nanosecond time scale to predict reasonable reaction mechanism [5].

* Corresponding author. Present address: State Key Laboratory of Multiphase Complex Systems, Institute of Process Engineering, Chinese Academy of Sciences, No. 1 Zhongguancun North Second Street, Beijing 100190, PR China.
Tel.: +86 10 82544944; fax: +86 10 82544945.
** Corresponding author. Present address: State Key Laboratory of Multiphase Complex Systems, Institute of Process Engineering, Chinese Academy of Sciences, No. 1 Zhongguancun North Second Street, Beijing 100190, PR China.
Tel.: +86 10 82544945; fax: +86 10 82544945.
E-mail addresses: xxia@home.ipe.ac.cn (X. Li), lguo@home.ipe.ac.cn (L. Guo).

1093-3263/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jmgm.2013.02.001

Supercomputers are often beyond the reach of research groups and high performance computational tools on a single desktop workstation would be more useful for individual researcher in exploiting simulation applications. However, according to the benchmark of Lammps [7] from Sandia National Laboratory, ReaxFF is the most CPU time and memory demanding potential among the 20 force fields when compared with LJ liquid potential. ReaxFF could take about 8.9 s for a single time-step (0.1 fs) for a system of PETN ([3-nitrooxy-2,2-bis(nitrooxymethyl)propyl] nitrate) crystal with 32,840 atoms in a simulation on a desktop computer using ReaxFF C code in Lammps, i.e., such a calculation for 1 ns would cost more than 1030 days, that is hardly acceptable for simulation practitioners. The hardware limitation could be still a barrier for applying ReaxFF MD simulation on desktop computers, particularly for simulation of large molecular systems.

With the computing peak performance and memory bandwidth far exceeding CPU, the graphics processing unit (GPU) has shown its great potential more recently in accelerating MD simulation by bringing teraflops computational capability to desktop workstations, which may bring transformation of many computations from clusters to desktops [8,9]. Although there may be feature limitations, the benchmarks using state-of-the-art GPU enabled MD programs such as Amber [10], NAMD [11], Gromacs [12] have shown speedups of several folds, even one to two orders of magnitude, over the well optimized CPU implementations on a single PC. The continuing advances of GPU could provide not only an alternative for parallel but also a new opportunity to manage the computing power and memory demanding nature imposed on computer hardware by ReaxFF, especially for effective simulation on desktop workstations.

In this paper, we present the novel design and algorithms of our GMD-Reax, the first GPU enabled ReaxFF MD implementation to our best knowledge, both for increasing simulation system size and acceleration of ReaxFF, aiming at a computational tool on desktop workstations. It works on a PC equipped with a single C2050 GPU from NVIDIA with Fermi architecture [13]. GMD-Reax has been tuned to take advantage of GPU as much as possible. In addition, every effort was made to develop a general architecture in the code so that it can be applicable with the upgrade of GPU.

In this paper, the ReaxFF force field and its paralleled algorithm implementation on CPU clusters are overviewed in Section 2, which are closely related in dealing with the computing challenge of ReaxFF by paralleled strategy. Recent developments in GPU technology and Compute Unified Device Architecture (CUDA) used for GPU programming will be briefly reviewed in Section 3. The original algorithms of ReaxFF and its GPU based algorithms in our GMD-Reax are outlined in Section 4, and then performance benchmarks of GMD-Reax are analyzed accordingly in Section 5. We conclude the paper with brief discussion to further research topics in the last section.

2. ReaxFF MD overview and related work

ReaxFF is an empirical reactive force field that was first reported for hydrocarbons [3] and then expanded in recent years by Duin and co-workers, for simulation of reactive molecular systems involving a range of elements such as carbon, hydrogen, oxygen, nitrogen and so on. ReaxFF MD has been successfully applied to a number of various complicated system simulations with reactions for the process of combustion and catalysis. ReaxFF has been employed in the study of several oxides such as Si/SiO_2 [14], V_xO_y [15], ZnO [16] and Al/Al_2O_3 [17] oxide interfaces for investigating catalytic mechanism or initial reaction path. ReaxFF has also been utilized to explore properties of new materials, namely single-walled carbon nanotubes (SWNT) [18] and Lithium Battery Electrolytes [19]. In addition, Mayernick [20] combined ReaxFF and Monte Carlo (MC) to perform simulated annealing to sample structural configurations of flat yttria-stabilized zirconia (YSZ) and obtained considerable results. More recently, ReaxFF was used in the combustion simulation of Illinois No. 6 coal char for a system containing over 30,000 atoms to explore the structural evolution of char structural and chemical mechanics by Castro-Marcano et al. [21] that demonstrated the ability of ReaxFF in dealing with large reactive molecular systems. These applications and the fact that the ReaxFF program has been distributed to over 150 research groups worldwide [22] as well have shown the capability and great potential of ReaxFF to handle complex chemistry and chemical diversity of larger molecular systems with chemical reaction to a wide range of materials and processes.

The ability of ReaxFF to describe bond formation and charge transfer for molecular systems in condensed phases comes from its special interatomic potential, in which the bonded interactions are based on distance dependent bond order functions that are adjusted to compensate for atomic over/under-coordination, and the atom charges are computed using electronegativity equalization. As stated in Eq. (1), the total potential energy in ReaxFF is the sum of the following energy terms [8]:

$$\begin{aligned} E_{\text{ReaxFF}}(r_{ij}, r_{ijk}, r_{ijkl}, q_i, \text{BO}_{ij}) = {} & E_{\text{bond}} + E_{\text{lp}} + E_{\text{over}} + E_{\text{under}} + E_{\text{val}} \\ & + E_{\text{pen}} + E_{\text{coa}} + E_{\text{tors}} + E_{\text{conj}} + E_{\text{Hbond}} \\ & + E_{\text{vdWaals}} + E_{\text{Coulomb}} \end{aligned} \tag{1}$$

where E_{vdWaals} and E_{Coulomb} are the energy of non-bonded interactions, namely the van der Waals and Coulomb interactions. In addition to E_{Hbond}, the energy of hydrogen bond, the rest of the energy terms are the bond order dependent and account for the bonded interactions that consists of E_{bond}, the energy of bond, E_{lp} for energy of lone-pairs, E_{over} of over-coordination, E_{under} of under-coordination, E_{val} of valence angle, E_{pen} of energy penalty for handling atoms with two double bonds, E_{coa} of coalition (three-body conjugation), E_{conj} of conjugated bonds (four-body conjugation), E_{tors} of torsion angles.

To avoid discontinuities on the potential energy surface, much more complex mathematical formulations for energy terms in Eq. (1) other than that in most of the classical MD methods are being used in ReaxFF [16] which significantly increase the computational complexity. Moreover, the time-step magnitude of ReaxFF MD is one order smaller than classical molecular dynamics (0.1 fs vs. 1 fs), resulting in its one order longer in computation time than that for classical MD. Unlike the fixed partial charges on atoms in classical MD, atomic partial charges in ReaxFF MD are dynamically approximated in a charge equilibration procedure by minimizing the Coulomb energy using Electron Equilibration Method (EEM) when the atomic coordinates are updated at each time-step [23]. Charge equilibration is quite computational expensive, although the procedure can be transformed into solving large sparse equations and the rank of the matrix is the total number of atoms. Therefore, ReaxFF MD can be 10–50 times slower than classical MD [22] and it becomes critical to improve its computing performance, especially when ReaxFF MD is used in simulation of complex chemical reactive systems.

As shown in Fig. 1, ReaxFF molecular dynamics treats atom interactions and their evolving with time in the same way as in classical MD. There are few dependencies and coupling between interacting atoms. The loop over atoms is basically data parallel that fits well with the nature of GPU and could take advantage of the great computing power of it. Meanwhile the memory demanding introduced by the bond order dependency and other complex calculations in ReaxFF can also be tackled with GPU's high memory bandwidth.

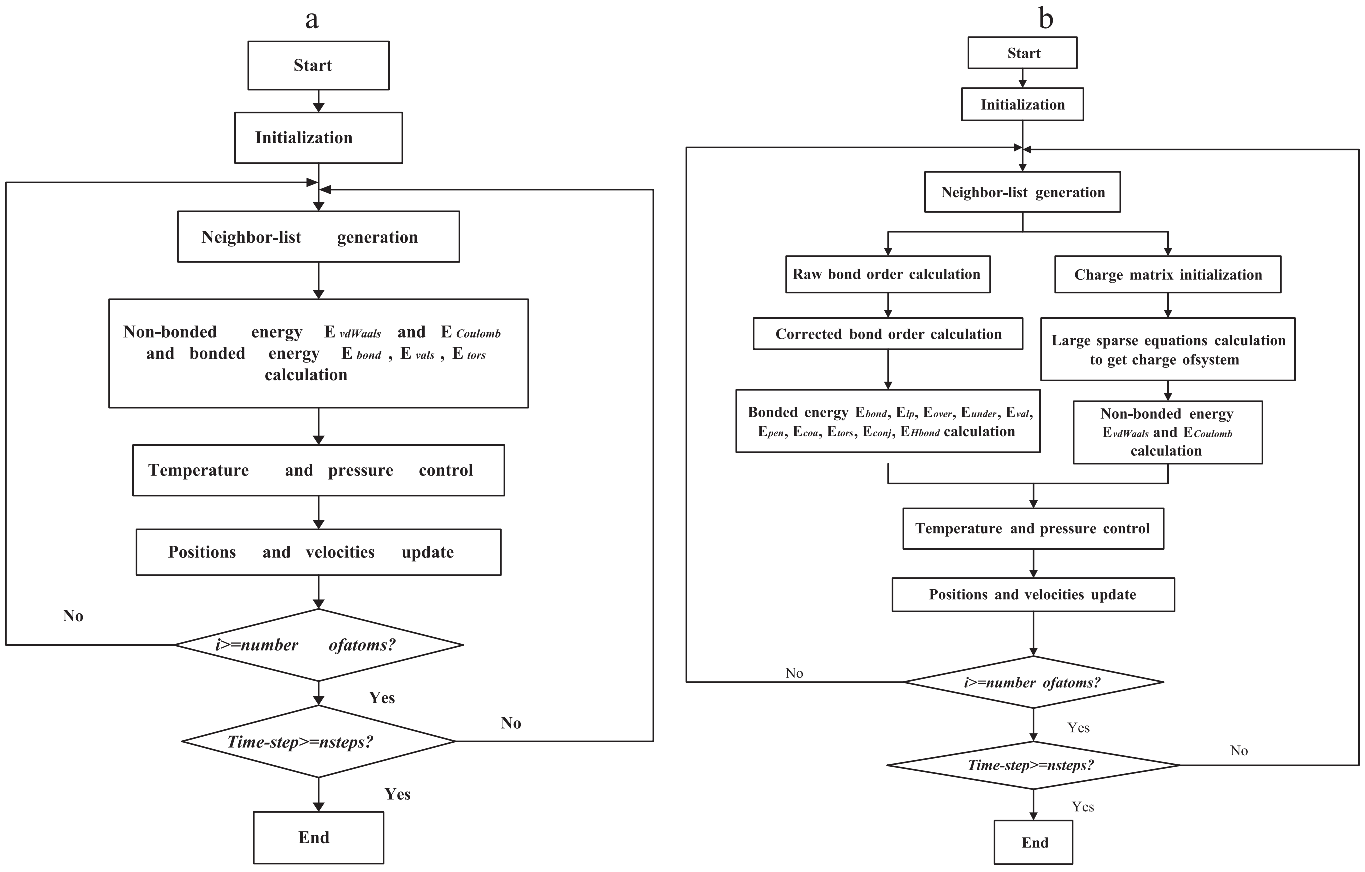

a
Start
Initialization
Neighbor-list generation
Non-bonded energy E vdWaals and E Coulomb and bonded energy E bond , E vals , E tors calculation
Temperature and pressure control
Positions and velocities update
i>=number ofatoms?
No
Yes
Time-step>=nsteps?
No
Yes
End
b
Start
Initialization
Neighbor-list generation
Raw bond order calculation
Charge matrix initialization
Corrected bond order calculation
Large sparse equations calculation to get charge ofsystem
Bonded energy Ebond, Elp, Eover, Eunder, Eval, Epen, Ecoa, Etors, Econj, EHbond calculation
Non-bonded energy EvdWaals and ECoulomb calculation
Temperature and pressure control
Positions and velocities update
i>=number ofatoms?
No
Yes
Time-step>=nsteps?
No
Yes
End

Fig. 1. The difference of algorithms and calculation scheme between classical MD and ReaxFF MD: (a) schematic flowchart of classical MD and (b) schematic flowchart of ReaxFF MD [3].

At the time writing, no GPU enabled acceleration of ReaxFF has been reported and there are only two high performance computing implementations of ReaxFF on high-end CPU cluster. The earlier one reported in 2007 is F-ReaxFF [24] that is embedded in a divide-and-conquer/cellular-decomposition framework for million-to-billion atoms simulations of chemical reactions of 1,3,5-trinitrohexahydro-striazine (RDX) crystal on high-end supercomputers. F-ReaxFF has been benchmarked on 1920 Itanium2 processors of the NASA Colombia supercomputer and demonstrated up to a half billion simulations of chemical reactions with unprecedented scales and accuracy on emerging petaflops-scale computer architectures. The other implementation is PuReMD (Purdue Reactive Molecular Dynamics code) that was recently reported by Aktulga et al. [25] in 2011. PuReMD has demonstrated good scalability to simulate up to 10 million water atoms and has higher processor performance in terms of per-time-step-per-atom when validated with up to 3375 cores on a commodity cluster (Hera at LLNL-OCF).

However, the two implementations, F-ReaxFF and PuReMD demonstrate applications based on computer clusters with thousands of cores, which usually can be accommodated in supercomputing centers and are normally too costly for most research labs. The paralleled ReaxFF kernel functions of PuReMD have been integrated into Lammps as a package in C language released in 2011 that is denoted as reax/c style [26] in Lammps, a result of close collaboration between Aktulga, etc. and Aidan Thompson at the Sandia National Laboratory. In Lammps, there is another ReaxFF package in FORTRAN language denoted as reax style, which is based on the original code of van Duin et al. [3], the major author of ReaxFF, by taking advantage of GRASP's parallel framework and the associated charge equilibration routines from Thompson [27]. The reax style was linked to Lammps as a FORTRAN library. The reax/c style (the C code) has been approved to be more efficient than reax style (the FORTRAN code) that matches exactly with Duin's original FORTRAN code. The ReaxFF codes in Lammps can run on desktop computers or clusters, benchmarks of Lammps [7] mentioned in Section 1 indicates that further improvement of the computing performance of will be highly needed when ReaxFF MD is applied in more complicated and larger molecular systems, for example, a coal molecular system could have over 30,000 atoms for combustion simulation by utilizing ReaxFF [21]. Our benchmarks on sample systems for coal pyrolysis simulation containing atoms ranging from 1378 to 27,283 revealed that the GPU enabled GMD-Reax is capable of achieving up to 6 folds speedups over the C codes in Lammps running on a single processor with 8 CPU cores, and can be over 12 times faster than that of Duin et al.'s FORTRAN code in Lammps. Detailed performance benchmarks of GMD-Reax are presented in Section 5.

3. GPU and CUDA overview

Due to its graphics lineage, graphics processing unit (GPU) is best suited for fine-grained data parallel computations that are arithmetic intensive which molecular dynamics simulation significantly contains [8]. CUDA, released in 2007, is a general purpose parallel computing architecture derived from C language that makes data parallel computing more straightforward on GPU, which results in the compelling applications in acceleration of classical molecular dynamics simulations [10–12]. To create GPU enabled ReaxFF MD on desktop computers could be of particular interest because GPU's high parallelism and memory bandwidth may offer a more efficient solution to ReaxFF's demanding nature to the computing power and memory of computer hardware. However, the

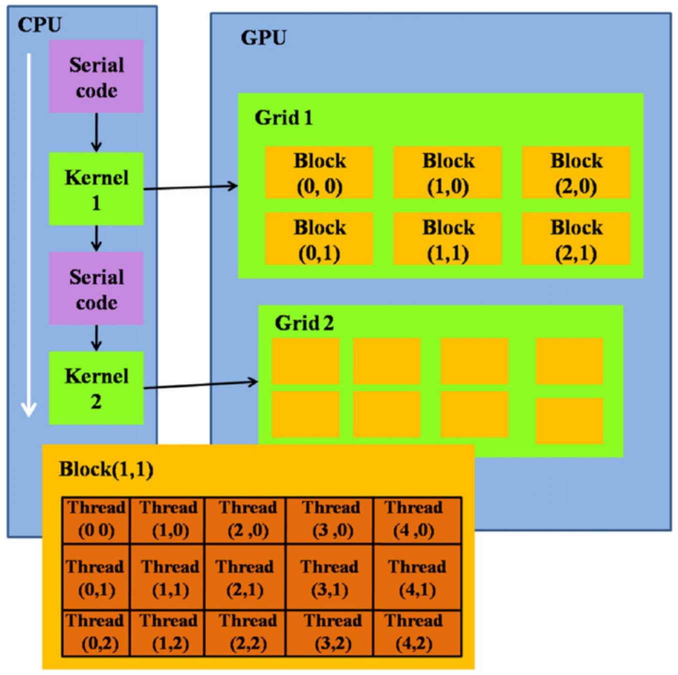


Fig. 2. CUDA execution model: the host (CPU) program launches a sequence of kernels executed on GPU [12].

tremendous computing power of GPU surpass CPU comes at the cost of less flexibility and increased programming complexity on GPU when compared to CPU. The creation of ReaxFF algorithms on GPU is not trivial and actually it is even more challenging to have them well optimized.

Basically, implementing an algorithm on GPU with CUDA is quite different from conventional CPU programming simply because GPU is data parallel. High performance implementation of an algorithm on GPU could be expected when data or task parallelism can be identified so that the corresponding computing tasks can be mapped into thousands of light threads launched on GPU to execute same codes with different data. A thorough understanding of GPU hardware architecture and CUDA programming model are necessary to exploit the computing power of GPU to develop highly efficient GPU enabled computational tool on desktop workstations.

Fig. 2 illustrates a typical data parallel programming model on GPU with CUDA. The host (CPU) launches a sequence of kernels organized as a hierarchy of threads running on the device (GPU) where the threads are grouped into blocks, and blocks into a grid. Each GPU has hierarchical memories to achieve high execution speed in their kernels. Registers and local memory are accessible for each thread; shared memory is for threads in a block; global memory, constant memory and texture memory are for grid level access. Efficient algorithm implementation on GPU requires not only proper hierarchical organization of multithreads but also efficient access of hierarchical device memories.

Although the fast upgrading of GPU hardware offers looser bounds in programming on GPU, special care should be taken both in algorithm design and performance tuning for optimized architecture features on GPU. In CUDA, the continuous threads in a block are executed in groups of 32 called a warp. Conditional branches carried out by the threads in the same warp will lead to branch divergence that CUDA will serialize both paths instead of parallel [28,29]. Therefore, branches should be avoided as much as possible in GPU thread and branch rich computing tasks could be more feasible on CPU than on GPU.

Another performance concern is global memory access latency. The large off-chip global memory has an extremely high access latency that is about 100 times over the arithmetic units [19]. To reduce it, coalesced global memory access is one of the most significant factors for optimized performance so that memory banks are continuous in physical space for a single access operation without conflicts. Moreover, proper use of fast on-chip memory in GPU will also be necessary for better performance [29]. For read-only data, constant memory could provide near arithmetic units rate performance, unfortunately its size is limited to 64 kB. Texture memory could be of help for random memory access or non-coalesced data access, while the shared memory is for inter-thread communication within a block and it has near register speed for data access without any latency in case of no bank conflict.

In very recent advances, implementations of GPU enabled molecular dynamics packages with CUDA such as Amber [10], ACEMD [30], NAMD [31], Gromacs [32], have been reported with significant benchmarked speed up on various GPU, but maybe with limited features when compared with their counterparts of CPU version. As an empirical reactive force field (ReaxFF) can be scheduled in the scheme of molecular dynamics simulations (see Fig. 1), in the sense that the motions of interacting atoms obey laws of classical mechanics and the interactions among them are modeled through suitable parameterizations. But accurate modeling of chemical reactions and keeping continuity on the potential energy surface require much more complex mathematical formulations of atomic interactions than those in most of classical MD. Consequently, ReaxFF is much more arithmetic intensive and memory demanding than the force fields in classical mechanics, which challenges both the hardware and algorithms strategy for efficient implementation. The hardware as well as tools and libraries available on GPU offered a chance to meet such challenges. In this paper, we focus on the core algorithms of ReaxFF molecular dynamics with CUDA and optimize them as much as possible on a single Fermi GPU.

4. GMD-Reax algorithms

At the time writing, GMD-Reax is the first implementation of ReaxFF algorithm within the scheme of molecular dynamics on GPU. Any possible care has been taken in the implementation to map the ReaxFF algorithms to the proper organization of GPU threads and data structures, to minimize sources of overhead and to optimize kernel parameters to achieve excellent per-time-step execution time. In this section, we discuss the details of algorithms and techniques used for GMD-Reax based on CUDA.

As summarized by the author of ReaxFF [3] and shown in Fig. 1, ReaxFF is embedded in the time-step iteration of MD. Unlike the calculation scheme of classical molecular dynamics, ReaxFF is divided into two parts after the neighbor list generated. The first part evaluates the bonded interactions that are all bond order dependent with very complicated corrections on bonded interactions, which significantly increase computational complexity. The second part performs the non-bonded interactions evaluation and dynamic charge equilibration by minimizing electrostatic energy that is beyond the scope of classical MD in which atomic charges are fixed during the simulation. Minimizing electrostatic energy through charge equilibration requires the solution of a large sparse linear system that must be performed at each time-step, which has to be a primary concern for parallel formulations of ReaxFF. Moreover, the time-step length for ReaxFF simulations is typically tenth of femtoseconds, an order of magnitude smaller than that in the classical MD. Consequently, these factors together pose significant computational challenges for the implementation of ReaxFF on GPU to reach spatio-temporal scales of nanometers and nanoseconds.

The scheme of ReaxFF MD based on CUDA is similar with the CPU version by C language in Lammps [21]. The basic difference is that the loops over atoms are mapped into the parallel multithreads on

GPU to perform the most of computing tasks that consist of complex computing instructions and few necessary branches, while CPU is responsible for logic rich operations and allocating tasks for GPU as well. It should be emphasized that it is necessary for GPU enabled program to copy data from the host to device before any computation to be performed and calculation results should also be transferred back from GPU to CPU. The data transformation between CPU and GPU via PCI-Express has a bandwidth up to 8 GB/s only, much lower than the memory bandwidth of CPU or GPU. Therefore it is essential to have such data transformation as less as possible for optimized performance. Our strategy to implement ReaxFF MD on a single GPU would make it more straightforward to minimize such data transformation, consequently and obviously, it is ideal to reduce the latency at most to meet such requirements to anticipate significant increase in the computational capability of ReaxFF simulation for larger reactive molecular system size and longer time scale over CPU on desktop workstations.

Linear-arrays are employed as the basic data storage structures for atom positions, forces and energies in GMD-Reax simply because CUDA currently does not support vector or queue data structure in C++. The array is a proper and reasonable data structure for it which can be read in a coalesced way on Fermi GPU. The positions and types of atoms are bounded to texture memory so as to facilitate the random access and increase the access speed. The fast on-chip constant memory broadcasts the parameters unchanged during the ReaxFF calculation to get near register speed access.

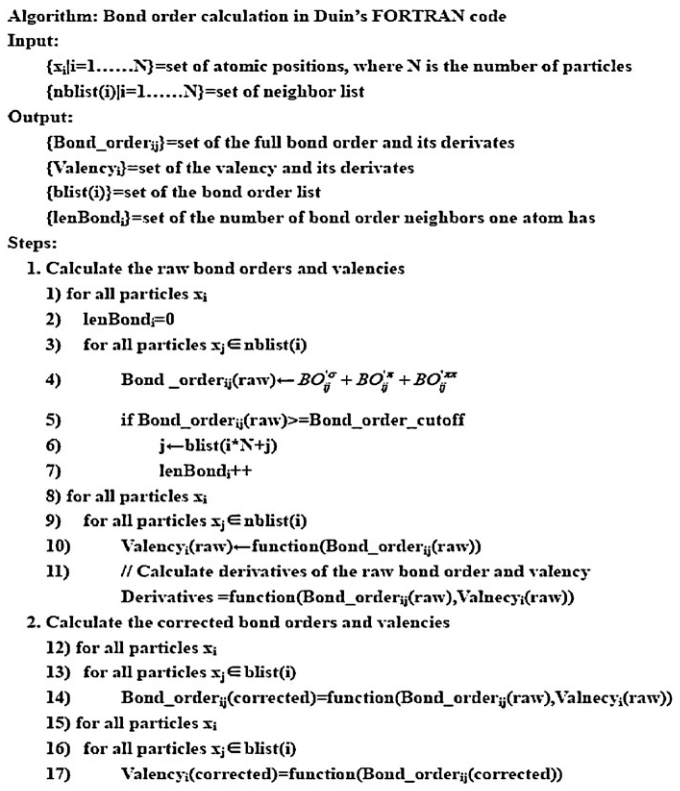
4.1. Bonded interactions

In ReaxFF, all bonded potentials including the hydrogen bond potential as well depend on the bond order (BO) between atoms. All forces arising from bonded interactions rely on the derivate values of the bond order. The concept of the bond order, originated from Pauli's exclusion principle, depends on the bond types formed by two atoms and the distance between them, the bond cut-off distance, as well. The bond order calculation contains two steps. The first step computes the raw BOs that are simple functions including three exponential functions of atom pair distance and bond cut-off as shown in Eq. (2) [5]. The second step corrects the bond order further using a function derived from the total bond order called valency and the raw bond orders. This correction makes the bond order accurate enough to describe bond breaking and forming in chemical reactions.

$$\mathrm{BO}'_{ij} = \mathrm{BO}'^{\sigma}_{ij} + \mathrm{BO}'^{\pi}_{ij} + \mathrm{BO}'^{\pi\pi}_{ij} = \exp\left[p_{bo1}\left(\frac{r_{ij}}{r_0^{\sigma}}\right)^{p_{bo2}}\right] + \exp\left[p_{bo1}\left(\frac{r_{ij}}{r_0^{\pi}}\right)^{p_{bo4}}\right] + \exp\left[p_{bo1}\left(\frac{r_{ij}}{r_0^{\pi\pi}}\right)^{p_{bo6}}\right] \quad (2)$$

As shown in Fig. 3, there are four double loops to calculate the bond orders, the raw valencies and their corrections in the bond order calculation algorithm implemented on CPU by Nomura et al. [5]. For a system of N atoms, the computing time complexity is O (N^2). To avoid repeated calculation, the computed derivatives of the bond order (line 11, Fig. 3) based on the raw bond order and valency are kept until the atom positions are changed.

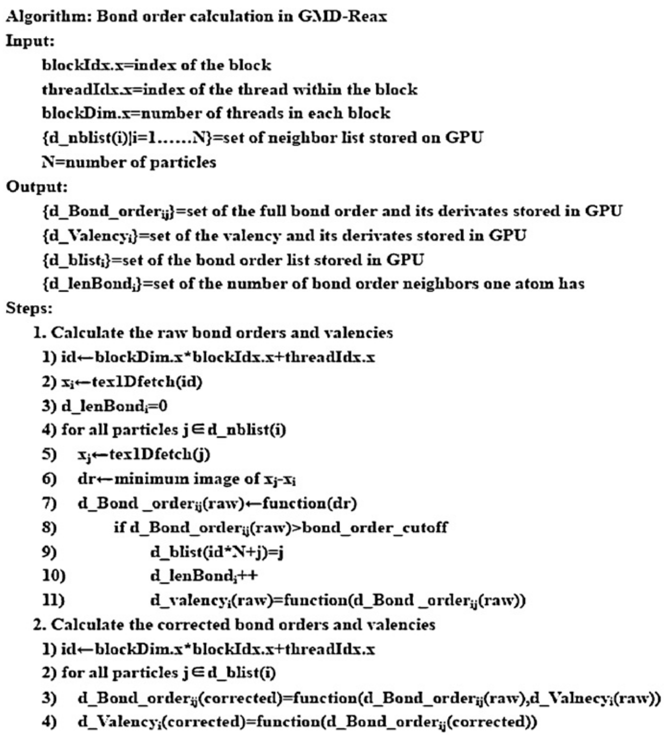
In GPU enabled algorithm for bond order computation in GMD-Reax, the tradeoffs between memory access complexity and number of computations are critical. Special care has been taken in data structure organization. The bond order list in ReaxFF contains the index of bond order for atom *i* and atom *j* when the bond order between them is equal or greater than the "bond order cut-off". The bond order list can be organized sequentially that lines the list up atom by atom, just like stored in a matrix using one row to store bonded list for one atom. Unfortunately, such data structure



```
Algorithm: Bond order calculation in Duin's FORTRAN code
Input:
    {x_i|i=1......N}=set of atomic positions, where N is the number of particles
    {nblist(i)|i=1......N}=set of neighbor list
Output:
    {Bond_order_ij}=set of the full bond order and its derivates
    {Valency_i}=set of the valency and its derivates
    {blist(i)}=set of the bond order list
    {lenBond_i}=set of the number of bond order neighbors one atom has
Steps:
  1. Calculate the raw bond orders and valencies
   1) for all particles x_i
   2)    lenBond_i=0
   3)    for all particles x_j ∈ nblist(i)
   4)        Bond _order_ij(raw) ← $BO'^{\sigma}_{ij} + BO'^{\pi}_{ij} + BO'^{\pi\pi}_{ij}$
   5)        if Bond_order_ij(raw)>=Bond_order_cutoff
   6)            j←blist(i*N+j)
   7)            lenBond_i++
   8) for all particles x_i
   9)    for all particles x_j ∈ nblist(i)
   10)       Valency_i(raw)←function(Bond_order_ij(raw))
   11)       // Calculate derivatives of the raw bond order and valency
             Derivatives =function(Bond_order_ij(raw),Valnecy_i(raw))
  2. Calculate the corrected bond orders and valencies
   12) for all particles x_i
   13)   for all particles x_j ∈ blist(i)
   14)       Bond_order_ij(corrected)=function(Bond_order_ij(raw),Valnecy_i(raw))
   15) for all particles x_i
   16)   for all particles x_j ∈ blist(i)
   17)       Valency_i(corrected)=function(Bond_order_ij(corrected))
```

Fig. 3. Bond order calculation algorithm of ReaxFF proposed by Nomura et al. [5].

cannot meet the requirement of coalesced global memory access in CUDA for better performance. Therefore, the bond order list is stored in an array d_blist (line 9 in step 1, Fig. 4) where the bond order index for each atom is sequenced along the column of the matrix, or column-based, which can facilitate the access to data of



```
Algorithm: Bond order calculation in GMD-Reax
Input:
    blockIdx.x=index of the block
    threadIdx.x=index of the thread within the block
    blockDim.x=number of threads in each block
    {d_nblist(i)|i=1......N}=set of neighbor list stored on GPU
    N=number of particles
Output:
    {d_Bond_order_ij}=set of the full bond order and its derivates stored in GPU
    {d_Valency_i}=set of the valency and its derivates stored in GPU
    {d_blist_i}=set of the bond order list stored in GPU
    {d_lenBond_i}=set of the number of bond order neighbors one atom has
Steps:
  1. Calculate the raw bond orders and valencies
   1) id←blockDim.x*blockIdx.x+threadIdx.x
   2) x_i←tex1Dfetch(id)
   3) d_lenBond_i=0
   4) for all particles j ∈ d_nblist(i)
   5)    x_j←tex1Dfetch(j)
   6)    dr←minimum image of x_j-x_i
   7)    d_Bond _order_ij(raw)←function(dr)
   8)        if d_Bond_order_ij(raw)>bond_order_cutoff
   9)            d_blist(id*N+j)=j
   10)           d_lenBond_i++
   11)           d_valency_i(raw)=function(d_Bond _order_ij(raw))
  2. Calculate the corrected bond orders and valencies
   1) id←blockDim.x*blockIdx.x+threadIdx.x
   2) for all particles j ∈ d_blist(i)
   3)    d_Bond_order_ij(corrected)=function(d_Bond_order_ij(raw),d_Valnecy_i(raw))
   4)    d_Valency_i(corrected)=function(d_Bond_order_ij(corrected))
```

Fig. 4. Bond order calculation algorithm implementation in GPU enabled GMD-Reax.

all the bonded atoms for one atom in a coalesced way. The number of bond order neighbors in each list varies from atom to atom, thus an additional array d_lenbond (line 10 in step 1, Fig. 4) is used to store such data. In total, more than 20 arrays are utilized to save values of the bond order, valency and its derivatives.

It is easy to cast the bond order calculation of ReaxFF shown in Fig. 3 into a data parallel algorithm for GPU with CUDA. The outer loop over atoms for the four loops calculating bond orders (line 1–line 12, Fig. 3) is mapped into multithreads on GPU in GMD-Reax where one thread will perform the bond order calculation for one atom as shown in Fig. 4. Two kernels are launched to calculate the raw and corrected bond order respectively.

The first kernel computes the raw bond orders and valencies as expressed in step 1, Fig. 4. Line 1 in step 1 defines the thread index for atoms that corresponds to the atom id. Then each thread reads coordinates of the atom from texture memory that is bounded as faster memory and loops over all of its neighbors from line 4. Because the number of atom neighbors varies, the loop may lead to divergent warp that will decrease data access performance. This kernel's performance has been evaluated by using NVIDIA Visual Profiler [31] and the results show that the achieved instructions per byte ratio is less than the balanced instruction for the device and the kernel is likely memory bandwidth limited, confirmed by the slowed data access. However, such algorithm of GMD-Reax on Fermi GPU is still much more efficient than that of the faster CPU implementation in Lammps, the C code, which is revealed clearly in the algorithm performance benchmark section.

The loop at line 5 reads the position of atom *j* which is one of the neighbors for atom *i* from texture memory to avoid non-coalesced access. Minimum image distance between two atoms is calculated based on Periodic Boundary Condition (PBC) theory in line 6. Line 7–line 11 in step 1 calculate the raw bond order, generate the bond order list and store the results in global memory without transferring to host so that the second kernel could read from the device directly to reduce the data copy latency between the host and device.

The second kernel for the corrected bond order (step 2, Fig. 4) contains plenty of computational expensive calculations. GMD-Reax employs math functions with lower precision but faster speed in Special Function Units (SFU) of CUDA from which several percentages of performance improvement could be obtained, while the accuracy for energy evaluation could be kept reasonable.

Bonded interactions in ReaxFF consisting of bond energy, atom under-/over coordination, valence angle terms and torsion angle terms are functions of the corrected bond orders and their corresponding valencies. Valence and torsion angle terms are deemed memory limited as most of their execution time is due to its complicated intermediate computations, and thus registers become strained resources, not enough for kernel functions for GMD-Reax. A possible solution is to utilize the free shared memory to replace some heavily used registers. Another simple but useful optimization is to substitute division with multiplication to the greatest extent because in the context of GPU computing, multiplication is ten times faster than division.

4.2. Non-bonded interactions

The non-bonded interactions consist of van der Waals and Coulomb energies. Instead of Lennard–Jones potentials used in most classical molecular dynamics force field, ReaxFF employs a distance-corrected Morse-potential in order to ensure the continuity of force filed [3]. van der Waals and Coulomb interactions are screened by a shielded function to adjust for orbital overlap between atoms at close distance such that non-bonded energy functions are taken into account between all atom pairs to avoid the need for computing long range electrostatics interaction, which also results in no need of any exclusion of bonded atoms in non-bonded evaluation as required in classical MD [5].

In Coulomb interactions, the atomic charges are dynamic and updated at each time-step using the charge equilibration (QEq) method with rigorous Slater orbital approach to account for the charge overlap when the atom coordinates are updated [3]. Developed by Rappe and Goddard [32] and formulated by Nakano [33], the QEq method minimizes the electrostatic energy of the system by adjusting the partial charge to individual atom based on interactions with their neighbors. With the Lagrange-multiplier method, the constrained energy minimization is equivalent to solve the electronegativity equalization problems.

To solve this problem, we reference the ideas from PuReMD in Lammps to change the electronegativity equalization condition into two large sparse linear equations that can be solved using the conjugate-gradient method [25]. As mentioned in Section 1, charge equilibration is very computationally expensive and optimized performance is highly needed. It has been proven that the bottleneck of conjugate-gradient method is the sparse matrix-vector multiplication (SpMV). We use two approaches to improve the computing performance for the multiplication. The first approach takes advantage of functions of CUBLAS in CUDA math libraries. If the number of atoms in a simulation system is less than 1000, it has a better performance than PuReMD in Lammps. But because this approach needs to store and calculate all elements in the matrix, it is not suitable for large systems where there are lots of zero elements. The second approach we use is to have the sparse matrix compressed to save space in the device, or ELLPACK [34]. ELLPACK consists of three arrays, val[] (float) and jlist[] (integer) with length N^2 (N is the number of atoms), and an additional integer array called numbrs[] with length N to store the actual length of non-zero elements in each row. The ELLPACK for SpMV implemented on GPU with CUDA takes advantage of coalesced global memory access by using the column-based ordering strategy again to store the elements. Other advantages of such approach are that the penalties for conditional branches and synchronized execution between threads are avoided.

In addition, GMD-Reax employs the strategy of *T* threads, developed by the NVIDIA engineers [35] and Francisco Vazquez et al. [36], in CUDA blocks to calculate a row of matrix multiplying a vector. The optimal value of *T* depends on the size of simulation systems and the blocksize in CUDA. For a coal pyrolysis system with 16,235 atoms shown in Fig. 5(a), the optimized *T* is 8 together with 128 threads in a block while for another coal system with 386 atoms in Fig. 5(b), the optimized *T* is 2 when the blocksize is 64. According to our experience and NVIDIA Visual Profile results, SpMV on GPU is a memory bound operation and its performance is strongly related to the pattern of the sparse matrix and the degree of coalesced memory access. Since every thread reads data numbrs[]/*T* times from the global memory in the strategy of "*T* threads", both the value of *T* and numbrs[] affect the coalesced memory access in a single data transfer operation. Another factor dominating the performance is the number of active stream multiprocessors (SM) used for a single memory access. Therefore, different simulation systems reach their best performance at different *T* threads and blocksize. Particularly, the results of two sample coal systems representing small and large system size respectively in Fig. 5 state that a specific reactive molecular system does have one optimal *T*, i.e., $T = 2^n \leq 32$. This conclusion could be used to choose a proper *T* and corresponding optimal blocksize for efficient ReaxFF MD simulation of a specific reactive molecular system.

Fig. 6 shows the algorithm of sparse matrix-vector multiplication in GMD-Reax. In line 1 of the kernel, instead of accessing

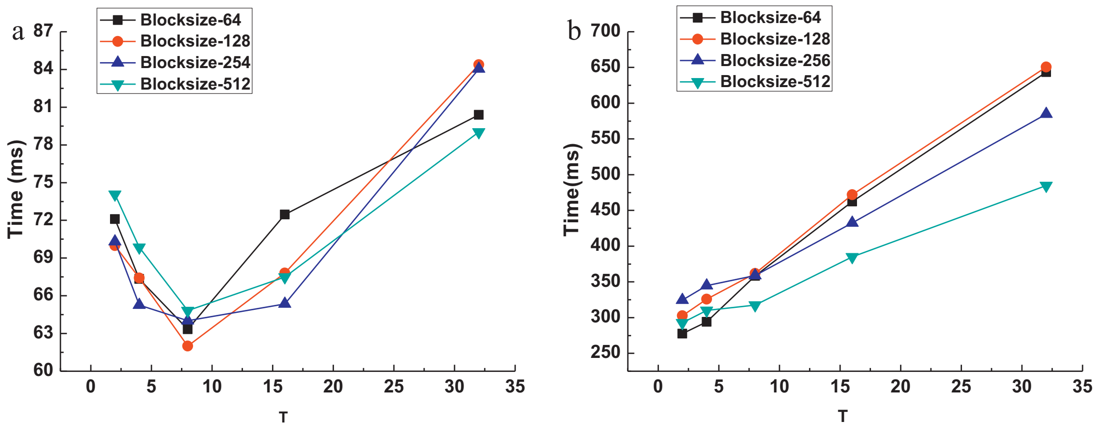


Fig. 5. Per time-step performance dependency of *T* on the system size as well as the blocksize for coal pyrolysis systems: (a) a smaller one with 386 atoms and (b) a larger system scale with 16,235 atoms.

to global memory, we define temporary variables in the shared memory to store the values for some threads in a block to reduce data access latency at some extent. The vectors are read using the texture unit to take advantage of the cache for the random read in line 11. As shown in Fig. 7, two approaches achieve almost the same performance for system with less than 2000 atoms while the "*T* threads" approach outperforms significantly than that of CUBLAS for larger system size.

Implementation of the non-bonded potential functions on GPU is straightforward and not provided here. The kernel for non-bonded interactions is "compute bound" according to the analysis by NVIDIA Visual Profiler, for there are plenty of numerical calculations in non-covalent energy functions, which also reveal the computing power demanding nature of ReaxFF at some extent, although non-bonded potential in ReaxFF is more complicated than that in classical MD.

Algorithm: Sparse Matrix-Vector Multiplication in GMD-Reax

```
__global__ void spmv_vector_kernel(int num_Atoms,sparse_matrix d_H,reax_atoms
d_atoms)
{
1.  extern __shared__ float vals[]; //allocate space in shared memory
2.  int tid=blockDim.x*blockIdx.x+threadIdx.x; //global thread index
3.  int row=tid/T; //use T threads to calculate a row of matrix multiplying a vector
4.  int temp=tid&(T-1); // thread index with the T threads
5.  int j;
6.  if(row<num_Atoms){
7.      int len=d_H.numbrs[row];
8.      vals[threadIdx.x]=0.0f;
9.      for(int pj=0+temp;pj<len;pj+=T){
10.         j=d_H.jlist[pj*num_Atoms+row];
11.         vals[threadIdx.x]+=d_H.val[pj*num_Atoms+row]*tex1Dfetch(tex_x,j);
        }

        //parallel reduction in shared memory
12.     if(temp<16&&temp>8) vals[threadIdx.x]+=vals[threadIdx.x+16];
13.     if(temp<8&&temp>4)  vals[threadIdx.x]+=vals[threadIdx.x+8];
14.     if(temp<4&&temp>2)  vals[threadIdx.x]+=vals[threadIdx.x+4];
15.     if(temp<2&&temp>1)  vals[threadIdx.x]+=vals[threadIdx.x+2];
16      if(temp<1)   vals[threadIdx.x]+=vals[threadIdx.x+1];
17.     if(temp==0)
18.         d_atoms.q[row]+=vals[threadIdx.x];
            //final results to get atomic charge for next iteration
    }
}
```

Fig. 6. Algorithm of sparse matrix-vector multiplication in GMD-Reax.

5. Algorithm performance benchmark

In this section, we provide a comprehensive analysis of the accuracy and performance of the algorithms in GMD-Reax. We choose the ReaxFF packages linked to Lammps as the baseline, which is the only free source code software for ReaxFF available that could facilitate computing performance benchmarks of energy terms in details and to investigate the bottleneck as well by inserting timing functions into the source codes. All proposed algorithms in GMD-Reax implemented using CUDA 4.0 driver were evaluated on a NVIDIA Tesla C2050 with 32 multiprocessors, 3GB device memory and 48KB shared memory/multiprocessor. Tests have been conducted with the GPU card attached to a CentOS 5.4 server with an Intel Xeon E5620 2.4 GHz, 2GB RAM.

We present all our tests using simulation systems of coal pyrolysis under the micro-canonical (NVE) ensemble. These simulation systems are chosen for performance evaluation not only because of the reasonably larger scale of system size but also that ReaxFF has been successfully used in condensed phase simulation for systems of hydrocarbon with oxygen, nitrogen and sulfur [3]. The simulation systems are constructed based on widely accepted Shinn and Wiser models for bituminous coal [37] with different scales containing atoms ranging from 1378 to 27,238, which are only for benchmarks purpose to demonstrate the capability of GMD-Reax for increased simulation system size and for longer time scale. The simulation

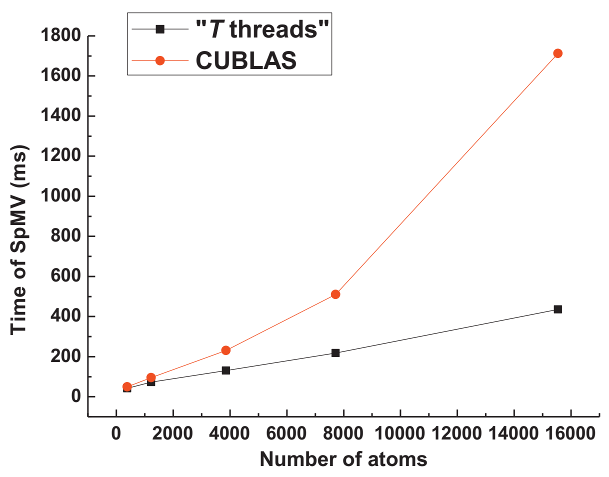


Fig. 7. Comparison of per time-step execution time of SpMV between "T threads" and CUBLAS in CUDA.

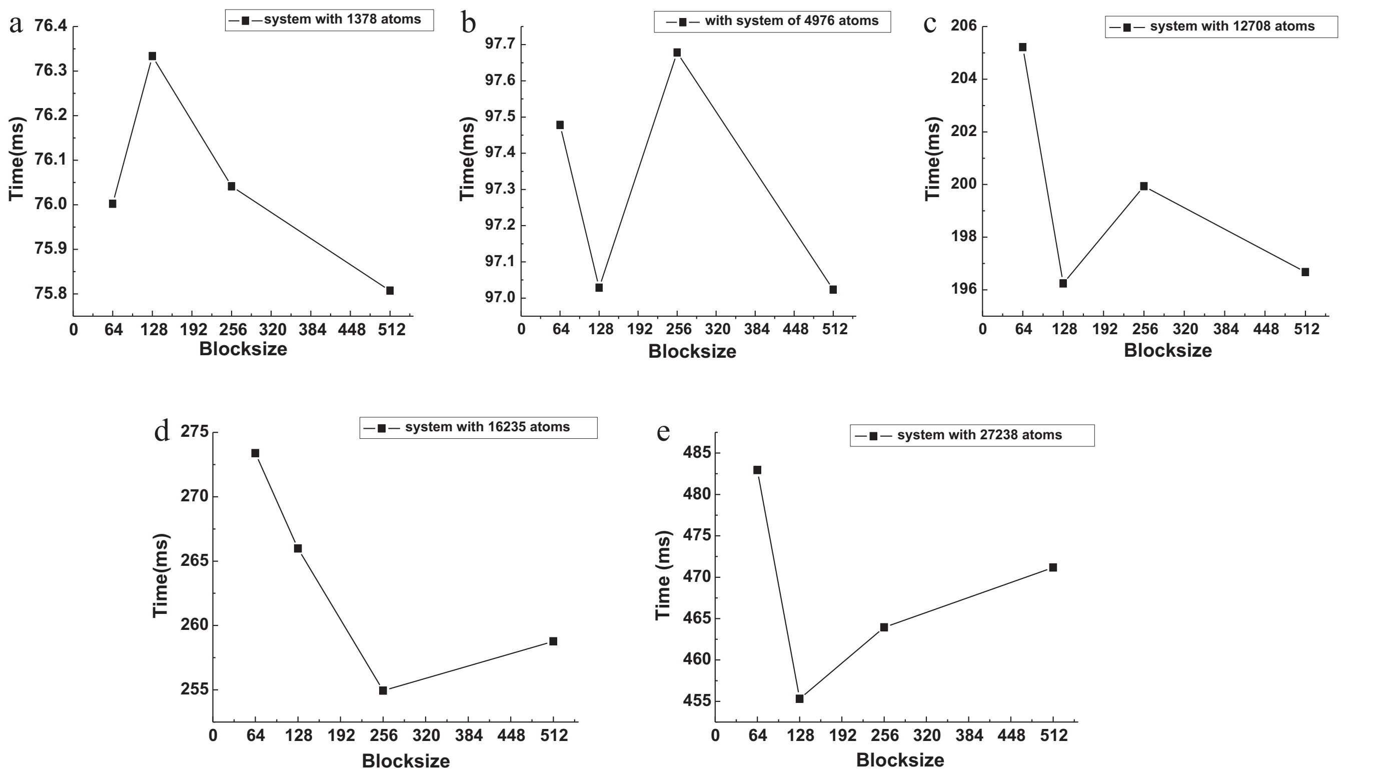


Fig. 8. Per time-step performance dependency on blocksize and system size for 5 coal pyrolysis simulation systems with (a) 1378 atoms, (b) 4976 atoms, (c) 12,708 atoms, (d) 16,235 atoms, and (e) 27,283 atoms.

details are out of the scope of this paper and will not be discussed here.

5.1. The accuracy of GMD-Reax

The accuracy of GMD-Reax has been benchmarked using different coal pyrolysis simulation systems. We choose the reax FORTRAN code in Lammps by Duin et al. [3] as the comparison baseline, simply because the code exactly matches with Duin's original FORTRAN code of ReaxFF. The benchmark results for two coal pyrolysis systems consist of 1378 atoms and 27,283 atoms with a density of 1.25 g/cm^3 and periodic boundary condition as well at room temperature are presented in Table 1, which shows the typical accuracy of GMD-Reax in simulation.

Table 1(a) and (b) compare the energy terms evaluated by GMD-Reax for different atom scales with that of the Lammps' FORTRAN code by Duin et al. in one time-step. The relative deviation is calculated by Eq. (3).

$$\text{Deviation}\,(\%) = \frac{|\text{value}_{\text{GMD-Reax}} - \text{value}_{\text{Lammps-Reax}}|}{\text{value}_{\text{Lammps-Reax}}} \times 100 \qquad (3)$$

As shown in Table 1, all energy terms computed by our GPU enabled GMD-Reax are in good agreement with that of Duin et al.'s CPU FORTRAN codes of reax in Lammps for different simulation system sizes and the deviations are reasonable for molecular dynamics.

5.2. Major algorithm performance benchmarks

To optimize the performance of GMD-Reax, the following major algorithms in GMD-Reax were evaluated.

- Bond_order_list: Bond order list generation.
- Bonded: bonded interactions including bond, valence-angle, torsion-angle, lone-pair, penalty, over-/under-coordination, conjugate interactions, and hydrogen bond as well.
- CG_QEq: charge equilibration calculation using conjugate-gradient method.
- Non-bonded: non-bonded interactions for both the van der Waals and Coulomb.

In order to test the scalability of GMD-Reax for different system size, coal pyrolysis systems with 4976, 12,708, 16,235 and 27,283 atoms respectively have been constructed in addition to 1378 atoms mentioned above. To demonstrate its excellent performance, we choose the C code of ReaxFF in Lammps based on PuReMD as the baseline that has been claimed to be 4 to 5 times faster than Duin's FORTRAN code in Lammps [29]. Thus the performance benchmarks will be more meaningful for simulation practitioners. Both of the baseline serial code, the C code for performance benchmarks and the FORTRAN code for accuracy benchmarks, was compiled in GMU GCC 4.1.2 with the full optimization (-O3) enabled.

On Fermi GPU, the performance of GMD-Reax is sensitive to the number of threads in one block. Since it is difficult and virtually impossible to precisely predict how many threads in a block might lead to the best performance, we investigated all possible blocksizes of 2^n, from 64 to 512, in order to optimize a blocksize that would be beneficial to GMD-Reax in achieving its optimal performance in simulation applications. Smaller blocksize occupies fewer resources leading to more active blocks in a SM to hide latencies caused by data access and/or synchronization, while greater blocksize means that more threads participate in the calculation to get higher instructions flow efficiency. Which factor dominates the performance depends on the system size and the computational complexity of algorithm itself. For a larger system with less arithmetic operations, more threads should be issued in a block. On the contrary, fewer threads should be launched in a block when plenty

Table 1
Comparison of ReaxFF energy terms per time-step between GMD-Reax and the FORTRAN code in Lammps.

(a) System with 1378 atoms				(b) System with 27,283 atoms			
Energy term	GMD-Reax (kcal/mol)	Lammps-Reax (kcal/mol)	Deviation	Energy term	GMD-Reax (kcal/mol)	Lammps-Reax (kcal/mol)	Deviation
Valence angle	3308.750	3308.752	<0.01%	Valence angle	64,753.930	64,753.955	<0.01%
Bond	−242,819.4	−242,819.3	<0.01%	Bond	−4,804,695	−4,804,694	<0.01%
Coalition	0	0	0	Coalition	0	0	0
Conjugation	−3268.953	−3268.954	<0.01%	Conjugation	−65,570.34	−65,570.35	<0.01%
Coulomb	−5813.860	−5813.860	<0.01%	Coulomb	−122,689.7	−122,689.7	<0.01%
Lone pair	107.7292	107.7313	<0.01%	Lone pair	2281.520	2281.515	<0.01%
Over and under coordination	−437.3380	−437.3360	<0.01%	Over and under coordination	−10,489.68	−10,489.70	<0.01%
Penalty	94.16270	94.16230	<0.01%	Penalty	1928.871	1928.872	<0.01%
Polarization	2795.033	2795.030	<0.01%	Polarization	62,316.62	62,316.60	<0.01%
Torsion angle	2776.189	2776.189	<0.01%	Torsion angle	49,308.33	49,308.32	<0.01%
van der Waals	67,576.15	67,576.17	<0.01%	van der Waals	1,338,871	1,338,871	<0.01%

of complex operations occupy a mass of hardware resources. In Fig. 8, the performance dependency of GMD-Reax on blocksize for coal pyrolysis simulation with different atom scales is presented. As shown in Fig. 8, an optimal blocksize for each sample coal pyrolysis simulation system does exist that are strongly dependent on the system size, for example, the optimal blocksize for system with 16,235 atoms is 256.

Fig. 9 indicates that compared with the baseline, the well optimized C code of ReaxFF in Lammps, most of the major algorithms in GMD-Reax greatly outperform at different system scales. The speedup in the following benchmarks is calculated by Eq. (4).

$$\text{Speedup} = \frac{\text{simulation time implemented on CPU}}{\text{simulation time by GMD-Reax}} \tag{4}$$

For the non-bonded calculation in GMD-Reax, we obtain speedups as high as 107 times faster than the C code in Lammps running on a single CPU core for the system with 27,283 atoms, at least 39 times higher, which shows that the performance of the algorithm with time complexity of O (N^2) was significantly improved by our implementation of ReaxFF on GPU with CUDA.

Accordingly, the performance of the bonded interaction calculations in GMD-Reax is up to 23.9 folds faster than that of the C code in Lammps, as shown in Fig. 9. The valence angle terms and torsion angle terms in bonded interactions of ReaxFF involve triple loops and quadruple loops accounting for three-body and four-body interactions that are not very suitable for GPU implementation. In addition, due to the dynamic nature of bond order, it is impossible to determine atom lists of three-body or four-body

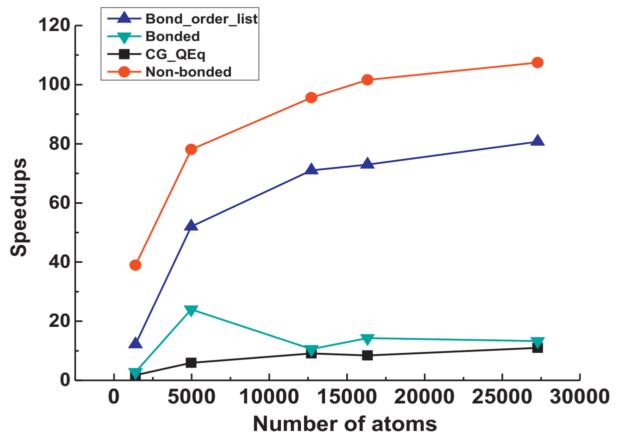


Fig. 9. Speed-ups of the major algorithms in GMD-Reax for per time-step against the well optimized ReaxFF codes in C language based on PuReMD in Lammps running on a single CPU core.

interaction at each time-step in advance, which makes it difficult to unroll triple or quadruple loops for paralleled computation. Therefore, the bonded interaction calculation gets not quite as impressive a speedup as with non-bonded interactions.

The bond order calculation, another major algorithm, is important for the overall performance simply because all bonded interactions are bond order dependent. The calculation has O (N^2) time complexity and GMD-Reax gets speedups as high as 80.7 times over the C code in Lammps, which confirms that algorithms with O (N^2) time complexity is suitable for implementing on GPU.

Fig. 10 compares execution time per time-step in percentage occupied by major algorithms in GMD-Reax against their counterparts in Lammps' C code based on PuReMD. Fig. 10(a) shows that the non-bonded calculation and the charge equilibration calculation in the Lammps' C codes account for over 90% of execution time with each occupying over 40%. While in GMD-Reax, the CG_QEq itself occupies most of time accounting for about 80% to 90% of the execution time for most of the system size tested, which suggests that our implementation on GPU handles much more efficiently with non-bonded interaction algorithms of O (N^2) time complexity than that of the baseline program. The non-bonded interactions in GMD-Reax accounts for less than 10% of execution time only and in the meanwhile it has good scalability with system size as shown in Figs. 9 and 10(b). The charge equilibration corresponds to the problem of assigning partial charges to atoms for the purpose of minimizing electrostatic energy under constraints of charge neutrality, which significantly increases the computation complexity and necessary data communication [27]. The speedup of CG_QEq in GMD-Reax is not so impressive and clearly it would be the performance bottleneck. However, the implementation of CG_QEq in GMD-Reax still achieves 1.8–10.9 times speedups against the Lammps C codes. Obviously, our GPU based implementation of ReaxFF has particular highly favorable performance over the CPU implementation in Lammps for larger system size simulation. The higher overall performance of GMD-Reax could be expected if the charge equilibration calculation could be optimized further.

5.3. Benchmark of the overall performance of GMD-Reax

We present here the benchmark results for the overall performance of GMD-Reax for the same simulation systems mentioned above with different atom scales. The benchmarks have been carried out against the two baseline codes of ReaxFF, the FORTRAN and C code of ReaxFF in Lammps respectively. Both baseline codes were compiled using gcc 4.1.2 optimized with -O3 and with OpenMPI library supported.

The benchmark results of the overall performance per time-step averaged over 100 time-steps for various system sizes are plotted in

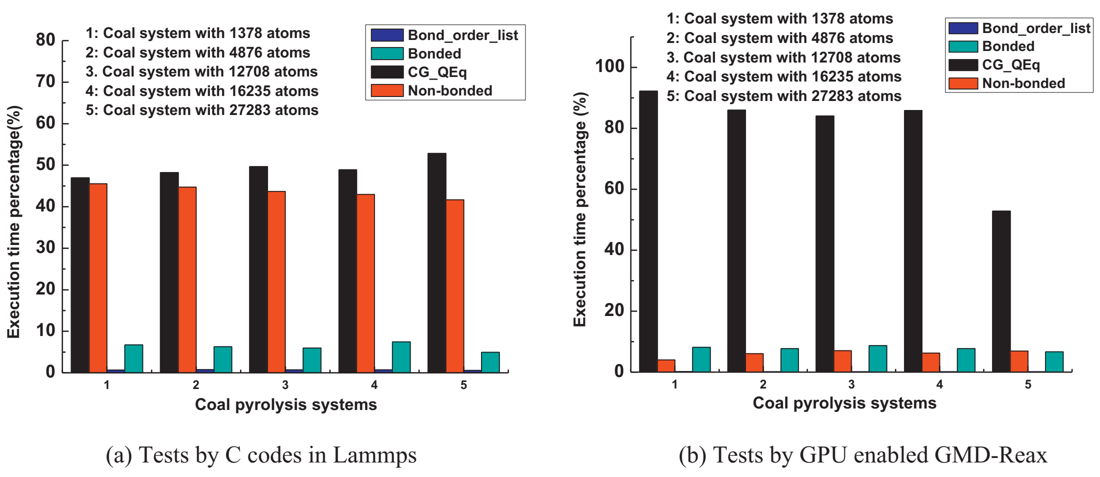


Fig. 10. Comparison of execution time per time-step occupied by major algorithms in two ReaxFF codes for coal pyrolysis simulations: (a) PuReMD based C codes in Lammps on one CPU core and (b) our GMD-Reax on a single GPU.

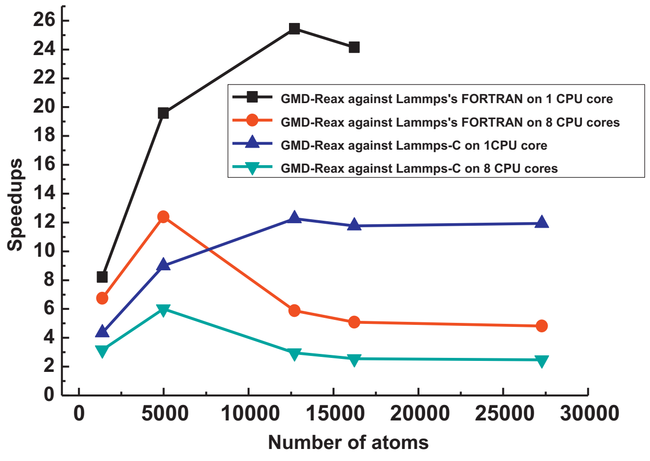


Fig. 11. Per time-step speedups of GMD-Reax averaged over 100 time steps against the C codes and FORTRAN codes of ReaxFF in Lammps. Note that the speedup data is not available for simulation system with 27,238 atoms that is out of the scope permitted in the Duin et al.'s FORTRAN code in Lammps on a single CPU core.

Fig. 11. GMD-Reax can be up to 25 times faster than Duin's FORTRAN codes on a single CPU core and 12 times faster than that of C codes based on PuReMD for a system containing 12,708 atoms. GMD-Reax achieves speedups as high as 4.5–12 times faster than the FORTRAN code running on 8 CPU cores and 2.5–6 times faster than the C code. GMD-Reax significantly outperforms the ReaxFF codes in Lammps.

The best performance of GMD-Reax over that of the Lammps code both in FORTRAN and C code was accomplished on 8 CPU cores for the same system scale with 4976 atoms. Moreover, GMD-Reax also achieves good scalability with the size of simulation systems. Obviously GPU enabled GMD-Reax offers a very competitive alternative for ReaxFF molecular dynamics simulation on a desktop workstation.

6. Conclusion

In this paper, we present the novel design and algorithms of GMD-Reax that is the first GPU enabled ReaxFF molecular dynamics program. The implemented algorithms have significantly increased the computational capability of ReaxFF MD on a desktop workstation with a single GPU attached for larger system size and longer time scale by taking advantage of GPU to manage with the computing power and memory demanding nature of ReaxFF. GMD-Reax is a completely new ReaxFF code in an effort to have it tuned to make use of GPU as optimal as possible.

The CUDA based implementation details for the two target algorithms in GMD-Reax, namely the bond order list generation and charge equilibration calculation, are discussed and proven to be effective in accelerating ReaxFF molecular dynamics.

Using a NVIDIA C2050 GPU for coal pyrolysis simulation systems with the number of atoms ranging from 1378 to 27,283, our implemented algorithms of GMD-Reax have been comprehensively benchmarked for the accuracy and performance against the two parallel ReaxFF packages running on CPUs in Lammps, the PuReMD based C code and Duin et al.'s FORTRAN code as well. Although single precision floating point arithmetic and some transcendental functions with low precision but fast speed in CUDA math libraries as well were utilized in GMD-Reax, reasonable accuracy were obtained in the benchmarks.

Of the major algorithms in GMD-Reax and with reasonable accuracy, the non-bonded calculation achieves the best performance as high as over 100 times speedup than that of the C code in Lammps, the faster version of ReaxFF, running on a single CPU core in terms of the simulation time per time-step. The result demonstrates that the performance of the algorithm with time complexity of O (N^2) could be significantly improved on GPU with CUDA. The other algorithms for the calculation of bond order, bonded interactions and charge equilibration also achieve up to 80.7, 23.9 and 10.9 times speedups than that of the C codes respectively.

When compared against the performance on a single CPU core, the overall performance of GMD-Reax can be up to 25 times faster than Duin et al.'s FORTRAN code and 12 times faster than that of the C code in Lammps in terms of the simulation time per time-step averaged over 100 steps.

Accordingly, when compared against the performance on 8 CPU cores, the overall performance of GMD-Reax achieves speedups as high as 4.5–12 times faster than the FORTRAN code and 2.5–6 times faster than that of the C code.

However, to improve the performance of the charge equilibration calculation further in GMD-Reax remains a challenge that limits the overall performance of GMD-Reax. The development of new iteration technology instead of conjugated-gradient method or more efficient data storage approach to reduce data access latency in GPU might be of help. Future work should focus on the improvements of this algorithm.

To improve the simulation system size further leads to the paralleled implementation of ReaxFF on multi-GPUs that would bring additional issues. With a single GPU, most of algorithms in

GMD-Reax implemented on the device and data transfer latency could be minimized as little as possible to improve the performance at most. However, because of the limit of hardware and memory demanding of ReaxFF, no greater than 40,000 atoms can be simulated on a single GPU with GMD-Reax. Naturally, multi-GPUs algorithms on a desktop workstation should be considered in future work. Parallelization on the multi-GPUs could bring domain decomposition and communication problems which need different algorithm strategies from that on a single GPU.

Our implemented algorithms of GMD-Reax offers a very competitive computational tool that surpass the available CPU codes for ReaxFF molecular dynamics simulation with larger system size and longer time scale on desktop workstations equipped with a single GPU. With the further extension of ReaxFF's force field parameters to cover more elements, and with the continuous rapid increase in the computational power and memory bandwidth of GPU as being progressing currently, GMD-Reax may potentially revolutionize reactive molecular dynamics simulation for complex systems from batch-mode clusters to desktop workstations.

Conflict of interest

The authors declare no competing financial interest.

Acknowledgements

This work was supported in part by the funding granted from the National Natural Science Foundation of China (21073195) and from China's State Key Laboratory of Multiphase Complex Systems.

References

[1] R.G. Parr, Density-functional theory, Chemical and Engineering News 68 (1990) 45.
[2] R.A. Friesner, Ab initio quantum chemistry: methodology and applications, Proceedings of the National Academy of Sciences of the United States of America 102 (2005) 6648–6653.
[3] A.C.T. van Duin, S. Dasgupta, F. Lorant, W.A. Goddard, ReaxFF: a reactive force field for hydrocarbons, Journal of Physical Chemistry A 105 (2001) 9396–9409.
[4] S.J. Stuart, Y. Li, O. Kum, J.W. Mintmire, A.F. Voter, Reactive bond-order simulations using both spatial and temporal approaches to parallelism, Structural Chemistry 15 (2004) 479–486.
[5] K.I. Nomura, R.K. Kalia, A. Nakano, P. Vashishta, A scalable parallel algorithm for large-scale reactive force-field molecular dynamics simulations, Computer Physics Communications 178 (2008) 73–87.
[6] A.C.T. van Duin, Using the ReaxFF Program. http://www.wag.caltech.edu/home/duin/Reax (accessed 19.07.12).
[7] Lammps. http://www.lammps.sandia.gov/bench.html (accessed 18.07.12).
[8] J.E. Stone, D.J. Hardy, I.S. Ufimtsev, K. Schulten, GPU-accelerated molecular modeling coming of age, Journal of Molecular Graphics and Modelling 29 (2010) 116–125.
[9] A.W. Gotz, M.J. Williamson, D. Xu, D. Poole, S. Le Grand, R.C. Walker, Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. Generalized Born, Journal of Chemical Theory and Computation 8 (2012) 1542–1555.
[10] Amber GPU. http://www.ambermd.org/gpus (accessed 18.07.12).
[11] NAMD GPU. http://www.ks.uiuc.edu/Research/gpu (accessed 18.07.12).
[12] Gramacs. http://www.gromacs.org (accessed 18.07.12).
[13] P.N. Glaskowsky, NVIDIA's Fermi: The First Complete GPU Computing Architecture. http://www.nvidia.com/object/fermi-architecture.html (accessed 09.08.12).
[14] K. Chenoweth, S. Cheung, A.C.T. van Duin, W.A. Goddard, E.M. Kober, Simulations on the thermal decomposition of a poly(dimethylsiloxane) polymer using the ReaxFF reactive force field, Journal of the American Chemical Society 127 (2005) 7192–7202.
[15] K. Chenoweth, A.C.T. van Duin, P. Persson, M.J. Cheng, J. Oxgaard, W.A. Goddard, Development and application of a ReaxFF reactive force field for oxidative dehydrogenation on vanadium oxide catalysts, Journal of Physical Chemistry C 112 (2008) 14645–14654.
[16] D. Raymand, A.C.T. van Duin, M. Baudin, K. Hermansson, A reactive force field (ReaxFF) for zinc oxide, Surface science 602 (2008) 1020–1031.
[17] M.F. Russo, R. Li, M. Mench, A.C.T. van Duin, Molecular dynamic simulation of aluminum–water reactions using the ReaxFF reactive force field, International Journal of Hydrogen Energy 36 (2011) 5828–5835.
[18] E. Zaminpayma, K. Mirabbaszadeh, Interaction between single-walled carbon nanotubes and polymers: a molecular dynamics simulation study with reactive force field, Computation Materials Science 58 (2012) 7–11.
[19] D. Bedrov, G.D. Smith, A.C.T. van Duin, Reactions of singly-reduced ethylene carbonate in lithium battery electrolytes: a molecular dynamics simulation study using the ReaxFF, Journal of Physical Chemistry A 116 (2012) 2978–2985.
[20] A.D. Mayernick, M. Batzill, A.C.T. van Duin, M.J. Janik, A reactive force-field (ReaxFF) Monte Carlo study of surface enrichment and step structure on yttria-stabilized zirconia, Surface Science 604 (2010) 1438–1444.
[21] F. Castro-Marcano, A.M. Kamat, M.F. Russo, A.C.T. van Duin, J.P. Mathews, Combustion of an Illinois No. 6 coal char simulated using an atomistic char representation and the ReaxFF reactive force field, Combustion and Flame 159 (2012) 1272–1285.
[22] A.C.T. van Duin, K. Chenoweth, B. Goddard, ReaxFF Force Fields – Development of a Transferable Empirical Method for Atomic-Scale Simulations of Chemical Reactions. http://www.wag.caltech.edu/home/duin/Reax/ (accessed 14.09.12).
[23] W.J. Mortier, S.K. Ghosh, S. Shankar, Electronegativity equalization method for the calculation of atomic charges in molecules, Journal of the American Chemical Society 108 (1986) 4315–4320.
[24] A. Nakano, R.K. Kalia, K. Nomura, A. Sharma, P. Vashishta, F. Shimojo, A.C.T. van Duin, W.A. Goddard, R. Biswas, D. Srivastava, A divide-and-conquer/cellular-decomposition framework for million-to-billion atom simulations of chemical reactions, Computation Materials Science 38 (2007) 642–652.
[25] H.M. Aktulga, J.C. Fogarty, S.A. Pandit, A.Y. Grama, Parallel reactive molecular dynamics: numerical methods and algorithmic techniques, Parallel Computing 38 (2012) 245–259.
[26] Lammps Pair-Style Reax. http://www.lammps.sandia.gov/doc/pair_reax.html (accessed 18.07.12).
[27] A. Thompson, ReaxFF in Lammps. http://www.lammps.sandia.gov/workshops/Feb10/Aidan_Thompson (accessed 18.07.12).
[28] J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, Journal of Computational Physics 227 (2008) 5342–5359.
[29] NVIDIA, NVIDIA CUDA C Programming Guide 4.0. http://www.developer.nvidia.com/cuda/nvidia-gpu-computing-documentation (accessed 08.08.12).
[30] ACEMD. http://www.acellera.com/?arg=acemd (accessed 18.07.12).
[31] NVIDIA, CUDA C Best Practice Guide 4.0. http://www.developer.nvidia.com/cuda/nvidia-gpu-computing-documentation (accessed 08.08.12).
[32] A.K. Rappe, W.A. Goddard, Charge equilibration for molecular-dynamics simulations, Journal of Physical Chemistry 95 (1991) 3358–3363.
[33] A. Nakano, Parallel multilevel preconditioned conjugate-gradient approach to variable-charge molecular dynamics, Computer Physics Communications 104 (1997) 59–69.
[34] F. Vazquez, J.J. Fernandez, E.M. Garzon, A new approach for sparse matrix vector product on NVIDIA GPUs, Concurrency and Computation: Practice and Experience 23 (2011) 815–826.
[35] N. Bell, M. Garlandy, NVIDIA. Efficient Sparse Matrix-Vector Multiplication on CUDA, http://www.nvidia.com/object/nvidia_research_pub_001.html (accessed 08.08.12).
[36] F. Vazquez, J.J. Fernandez, E.M. Garzon, Automatic tuning of the sparse matrix vector product on GPUs based on the ELLR-T approach, Parallel Computing 38 (2012) 408–420.
[37] J.P. Mathews, A.L. Chaffee, The molecular representations of coal – a review, Fuel 96 (2012) 1–14.