Computer Physics Communications 218 (2017) 10–16

Contents lists available at ScienceDirect

Computer Physics Communications

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

Efficient molecular dynamics simulations with many-body potentials on graphics processing units

Zheyong Fan [a,b,*], Wei Chen [c,*], Ville Vierimaa [b], Ari Harju [b]

[a] School of Mathematics and Physics, Bohai University, Jinzhou, China
[b] COMP Centre of Excellence and Helsinki Institute of Physics, Department of Applied Physics, Aalto University, Helsinki, Finland
[c] Computer Network Information Center, Chinese Academy of Sciences, P.O. Box 349, 100190 Beijing, China

ARTICLE INFO

Article history:
Received 12 October 2016
Received in revised form 2 May 2017
Accepted 4 May 2017
Available online 10 May 2017

Keywords:
Molecular dynamics simulation
Many-body potential
Tersoff potential
Stillinger–Weber potential
Graphics processing units
Virial stress
Heat current

ABSTRACT

Graphics processing units have been extensively used to accelerate classical molecular dynamics simulations. However, there is much less progress on the acceleration of force evaluations for many-body potentials compared to pairwise ones. In the conventional force evaluation algorithm for many-body potentials, the force, virial stress, and heat current for a given atom are accumulated within different loops, which could result in write conflict between different threads in a CUDA kernel. In this work, we provide a new force evaluation algorithm, which is based on an explicit pairwise force expression for many-body potentials derived recently (Fan et al., 2015). In our algorithm, the force, virial stress, and heat current for a given atom can be accumulated within a single thread and is free of write conflicts. We discuss the formulations and algorithms and evaluate their performance. A new open-source code, GPUMD, is developed based on the proposed formulations. For the Tersoff many-body potential, the double precision performance of GPUMD using a Tesla K40 card is equivalent to that of the LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) molecular dynamics code running with about 100 CPU cores (Intel Xeon CPU X5670 @ 2.93 GHz).

1. Introduction

Molecular dynamics (MD) simulation is one of the most important numerical tools in investigating various physical properties of materials. Many applications using MD simulation demand high performance computing. In the past decade, the computational power of general-purpose graphics processing units (GPUs) has been exploited to accelerate many MD simulations. Not only existing MD codes and libraries, such as AMBER [1], Gromacs [2,3], LAMMPS [4–6], NAMD [7], and OpenMM [8] have been benefited from utilizing GPUs as accelerators, but also new codes, such as HOOMD-blue [9–11], HALMD [12], and RUMD [13], have been built from the ground up to achieve high performance using one or more GPUs.

Most of the previous relevant works have only considered pairwise potentials, or a special many-body potential, namely, the embedded atom method [14–16], which are relatively simple to implement on GPUs. GPU-acceleration of many-body potentials such as the Tersoff [17], Stillinger–Weber [18], and Brenner [19] potentials, which play an important role in modelling various materials, is more challenging and has only attracted some attention recently [20–25]. Taking three-body interaction as an example, a naive implementation of the force evaluation function, as usually done in a serial CPU code, requires accumulating the forces on three different atoms within a single thread. In a GPU kernel with many threads, each atom is usually associated with one thread and the force accumulation for an atom from the thread it belongs to will conflict with the force accumulation for the same atom from another thread. This causes a problem called write conflict where two threads try to write data simultaneously into the same global memory [26]. One way to avoid write conflict is to use atomic operations, which are usually quite slow and can also introduce randomness in the computation, which is undesirable for debugging.

There have been some proposals to avoid using atomic operations. Hou et al. [20] proposed an algorithm for implementing the Tersoff potential on a GPU, which has achieved impressive performance, but requires using a special fixed neighbour list and is thus not quite flexible. Brown and Yamada [21] proposed a flexible GPU-implementation of the Stillinger–Weber potential within LAMMPS, which is free of write conflicts. A similar proposal was given by Knizhnik et al. [22]. Recently, Höhnerbach et al. [23] developed a vectorization scheme to achieve performance portability across various parallel computing platforms for the Tersoff potential within LAMMPS. GPU-acceleration of the more complicated second-generation REBO potential [19] has also been studied

* Corresponding authors.
E-mail addresses: brucenju@gmail.com (Z. Fan), weichen@cnic.cn (W. Chen).

http://dx.doi.org/10.1016/j.cpc.2017.05.003
0010-4655/

by Tredak et al. [24]. In a recently published work [25] (after we submitted this paper), Nguyen reported significant speedups for MD simulations with Tersoff-type potentials using one or more high-end GPUs.

Here, we propose a general algorithm of force evaluation for many-body potentials and present details of its GPU-implementation and performance. The new force evaluation algorithm is based on an explicit pairwise force expression for many-body potentials derived recently [27]. In this approach, the force, virial stress, and heat current for a given atom are well defined and can be accumulated within a single thread. Therefore, write conflict is absent by construction. To be specific, we discuss the algorithm explicitly in terms of the Tersoff potential, but performance evaluation is made for both the Tersoff potential and the Stillinger–Weber potential. The implementation is done based on a previous work [28], and the resulting code, which we call GPUMD (Graphics Processing Units Molecular Dynamics), will be made public soon. Using silicon crystal as a test system, we measure the performance of GPUMD and compare it with LAMMPS.

2. Formulations and algorithms

2.1. The Tersoff many-body potential

Although the method to be introduced is applicable to any many-body potential, it is beneficial to start with an explicit example, which is taken as the widely used Tersoff potential. Generalizations to other many-body potentials will be discussed later.

The total potential energy for a system of N atoms described by the Tersoff potential can be written as [17]

$$U = \frac{1}{2}\sum_i \sum_{j\neq i} U_{ij}, \tag{1}$$

where

$$U_{ij} = f_C(r_{ij})\left(f_R(r_{ij}) - b_{ij} f_A(r_{ij})\right), \tag{2}$$

$$b_{ij} = \left(1 + \beta^n \zeta_{ij}^n\right)^{-\frac{1}{2n}}, \tag{3}$$

$$\zeta_{ij} = \sum_{k\neq i,j} f_C(r_{ik}) g_{ijk}, \tag{4}$$

$$g_{ijk} = 1 + \frac{c^2}{d^2} - \frac{c^2}{d^2 + (h - \cos\theta_{ijk})^2}. \tag{5}$$

Here, β, n, c, d, and h are material-specific parameters and θ_{ijk} is the angle formed by $\boldsymbol{r}_{ij}$ and $\boldsymbol{r}_{ik}$, which implies that

$$\cos\theta_{ijk} = \cos\theta_{ikj} = \frac{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{ik}}{r_{ij} r_{ik}}. \tag{6}$$

Our convention is that $\boldsymbol{r}_{ij} \equiv \boldsymbol{r}_j - \boldsymbol{r}_i$ represents the position difference pointing from atom i to atom j. The magnitude of $\boldsymbol{r}_{ij}$ is denoted as r_{ij}.

As in many empirical potentials, the energy U_{ij} consists of a repulsive part $f_R(r_{ij})$ and an attractive part $-b_{ij} f_A(r_{ij})$. The many-body nature of the Tersoff potential is embodied in the bond order function b_{ij} appearing in the attractive part, the value of which depends not only on $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$, but also on the positions of other atoms near atom i.

The function $f_C(r_{ij})$ is a cutoff function, which is only nonzero when r_{ij} is less than a cutoff distance. Therefore, a Verlet neighbour list can be used to speed up the force evaluation. For uniform cutoff, the standard cell list method is very efficient, although more sophisticated methods perform better for systems with large size disparities [29].

For simplicity, we have presented the original Tersoff potential formulation in a form suitable for single-element systems. Our algorithm and implementation are more general, which can treat systems with more than one type of atom or systems described by a modified formulation of the Tersoff potential.

2.2. The conventional method of implementing the tersoff potential

Due to the three-body nature of the Tersoff potential, the conventional method for evaluating the interatomic forces is significantly different from that in the case of a simple two-body potential. Algorithm 1 presents a pseudo code for the conventional method as implemented in most existing MD codes such as LAMMPS [4]. The following symbols are used:

- N: number of atoms
- U_i: potential energy of atom i
- $\boldsymbol{F}_i$: total force on atom i
- $\mathbf{W}_i$: per-atom virial stress of atom i
- NN_i: number of neighbour atoms of atom i
- NL_{im}: index of the mth neighbour atom of atom i
- $\boldsymbol{J}_i$: per-atom heat current of atom i.

Algorithm 1 Pseudo code for the conventional method of evaluating many-body force and related quantities.

1: **for** $i = 0$ to $N-1$ **do**
2: Initialize U_i, $\boldsymbol{F}_i$, and $\mathbf{W}_i$ to zero
3: **end for**
4: **for** $i = 0$ to $N-1$ **do**
5: **for** $m = 0$ to $\mathrm{NN}_i - 1$ **do**
6: $j \leftarrow \mathrm{NL}_{im}$
7: $U_i \leftarrow U_i + \frac{1}{2}U_{ij}$
8: $\boldsymbol{F}_i \leftarrow \boldsymbol{F}_i + \boldsymbol{F}_i^{(ij)}$
9: $\boldsymbol{F}_j \leftarrow \boldsymbol{F}_j + \boldsymbol{F}_j^{(ij)}$
10: $\mathbf{W}_i \leftarrow \mathbf{W}_i - \frac{1}{2}\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_i^{(ij)}$
11: **for** $n = 0$ to $\mathrm{NN}_i - 1$ **do**
12: $k \leftarrow \mathrm{NL}_{in}$
13: **if** $k = j$ **then**
14: Continue
15: **end if**
16: $\boldsymbol{F}_i \leftarrow \boldsymbol{F}_i + \boldsymbol{F}_i^{(ijk)}$
17: $\boldsymbol{F}_j \leftarrow \boldsymbol{F}_j + \boldsymbol{F}_j^{(ijk)}$
18: $\boldsymbol{F}_k \leftarrow \boldsymbol{F}_k + \boldsymbol{F}_k^{(ijk)}$
19: $\mathbf{W}_i \leftarrow \mathbf{W}_i + \frac{1}{3}\left(\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_j^{(ijk)} + \boldsymbol{r}_{ik}\otimes\boldsymbol{F}_k^{(ijk)}\right)$
20: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \frac{1}{3}\left(\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_j^{(ijk)} + \boldsymbol{r}_{ik}\otimes\boldsymbol{F}_k^{(ijk)}\right)$
21: $\mathbf{W}_k \leftarrow \mathbf{W}_k + \frac{1}{3}\left(\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_j^{(ijk)} + \boldsymbol{r}_{ik}\otimes\boldsymbol{F}_k^{(ijk)}\right)$
22: **end for**
23: **end for**
24: **end for**
25: **for** $i = 0$ to $N-1$ **do**
26: $\boldsymbol{J}_i \leftarrow \mathbf{W}_i \cdot \boldsymbol{v}_i$
27: **end for**

In Algorithm 1, the potential energy $U_i \equiv \sum_{j\neq i} U_{ij}/2$ is accumulated in line 7, the two-body parts of the force and per-atom virial stress are accumulated in lines 8–9 and 10, respectively, and the many-body parts of the force and per-atom virial stress are accumulated in lines 16–18 and 19–21, respectively. Last, in line 26, the per-atom heat current is calculated from the per-atom virial stress and velocity.

The forces defined in the pseudo code can be explicitly written as

$$\boldsymbol{F}_i^{(ij)} = -\frac{1}{2}\frac{\partial}{\partial\boldsymbol{r}_i}\left(f_C(r_{ij}) f_R(r_{ij})\right) + \frac{1}{2} b_{ij}\frac{\partial}{\partial\boldsymbol{r}_i}\left(f_C(r_{ij}) f_A(r_{ij})\right), \tag{7}$$

$$\boldsymbol{F}_j^{(ij)} = -\frac{1}{2}\frac{\partial}{\partial \boldsymbol{r}_j}\left(f_C(r_{ij})f_R(r_{ij})\right) + \frac{1}{2}b_{ij}\frac{\partial}{\partial \boldsymbol{r}_j}\left(f_C(r_{ij})f_A(r_{ij})\right), \tag{8}$$

$$\boldsymbol{F}_i^{(ijk)} = \frac{1}{2}f_C(r_{ij})f_A(r_{ij})\frac{\partial b_{ij}}{\partial \zeta_{ij}}\frac{\partial}{\partial \boldsymbol{r}_i}\left(f_C(r_{ik})g_{ijk}\right), \tag{9}$$

$$\boldsymbol{F}_j^{(ijk)} = \frac{1}{2}f_C(r_{ij})f_A(r_{ij})\frac{\partial b_{ij}}{\partial \zeta_{ij}}\frac{\partial}{\partial \boldsymbol{r}_j}\left(f_C(r_{ik})g_{ijk}\right), \tag{10}$$

$$\boldsymbol{F}_k^{(ijk)} = \frac{1}{2}f_C(r_{ij})f_A(r_{ij})\frac{\partial b_{ij}}{\partial \zeta_{ij}}\frac{\partial}{\partial \boldsymbol{r}_k}\left(f_C(r_{ik})g_{ijk}\right). \tag{11}$$

Here, only the repulsive part of $\boldsymbol{F}_i^{(ij)}$ [the first term on the right hand side of Eq. (7)] is pairwise; all the other force expressions are not. Newton's third law could be exploited to reduce calculations regarding the pairwise part, but this would not result in a noticeable improvement of the overall performance, because the pairwise part only takes up a tiny fraction of the whole force evaluation. Therefore, Newton's third law has not been used in Algorithm 1. Regardless of using Newton's third law for the pairwise part or not, the above conventional method can be straightforwardly implemented on the CPU.

Newton's third law has also not been used on the GPU implementation [9] of two-body potentials due to the problem of concurrent writes (write conflict) to the same location in global memory. When two or more threads in the same warp write to the same location in global memory, only one thread performs the write and it is not defined which thread does it [26]. To understand why this feature of CUDA restricts the use of Newton's third law on the GPU, we recall that in the commonly used force-evaluation kernel [9] for two-body potentials, which we call the thread-scheme [28], one thread is devoted to the calculation of the total force on one atom. Using Newton's third law would require accumulating the total force on atom i by two threads. If these two threads are in the same warp, the total force on atom i would not be correctly accumulated. One may try to use atomic operations to solve this problem, but this would hardly result in a gain of performance.

In view of the above discussion, one would immediately realize the difficulty of implementing Algorithm 1 on the GPU: the partial accumulations of the forces and stresses on atoms j and k by the thread associated with atom i (cf. lines 9, 17–18, 20–21 in Algorithm 1) would conflict with those by the threads associated with atoms j and k. Therefore, Algorithm 1 is not suitable for GPU-implementation. This difficulty has also been realized previously and some strategies are proposed to circumvent it [20,21]. Below, we present a general algorithm for many-body potentials which can lead to efficient GPU-implementation.

2.3. A new method of implementing the Tersoff potential

Despite the many-body nature of the Tersoff potential, a pairwise force expression that complies with Newton's third law has been derived recently [27]:

$$\boldsymbol{F}_i = \sum_{j\neq i}\boldsymbol{F}_{ij}^{\text{Tersoff}}, \tag{12}$$

$$\boldsymbol{F}_{ij}^{\text{Tersoff}} = -\boldsymbol{F}_{ji}^{\text{Tersoff}} = \frac{1}{2}\frac{\partial}{\partial \boldsymbol{r}_{ij}}\left(U_{ij} + U_{ji} + \sum_{k\neq i,j}\left(U_{ik} + U_{jk}\right)\right), \tag{13}$$

which can be simplified in terms of the per-atom potential to be

$$\boldsymbol{F}_{ij}^{\text{Tersoff}} = \frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} - \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}. \tag{14}$$

An explicit expression for $\partial U_i/\partial \boldsymbol{r}_{ij}$ can be found in Ref. [27].

One may wonder whether the new pairwise force expression $\boldsymbol{F}_{ij}^{\text{Tersoff}}$ produces per-atom forces $\boldsymbol{F}_i$ that are equivalent to those obtained by the conventional method. The answer should be definitely yes; otherwise, either the conventional or our new method of force evaluation is wrong. To show this equivalence, let us note that according to Algorithm 1, the total force acting on atom i can be written as:

$$\boldsymbol{F}_i = \sum_{j\neq i}\left(\boldsymbol{F}_i^{(ij)} + \boldsymbol{F}_i^{(ji)}\right) + \sum_{j\neq i}\sum_{k\neq i,j}\left(\boldsymbol{F}_i^{(ijk)} + \boldsymbol{F}_i^{(jik)} + \boldsymbol{F}_i^{(jki)}\right). \tag{15}$$

Plugging in Eqs. (7)–(11), and changing the absolute positions $\boldsymbol{r}_i$ to relative ones $\boldsymbol{r}_{ij}$ using the chain rule, we get

$$\begin{aligned}
\boldsymbol{F}_i =& \frac{1}{2}\sum_{j\neq i}\frac{\partial}{\partial \boldsymbol{r}_{ij}}\left(f_C(r_{ij})f_R(r_{ij})\right) - \frac{1}{2}\sum_{j\neq i}b_{ij}\frac{\partial}{\partial \boldsymbol{r}_{ij}}\left(f_C(r_{ij})f_A(r_{ij})\right)\\
&+ \frac{1}{2}\sum_{j\neq i}\frac{\partial}{\partial \boldsymbol{r}_{ij}}\left(f_C(r_{ji})f_R(r_{ji})\right) - \frac{1}{2}\sum_{j\neq i}b_{ji}\frac{\partial}{\partial \boldsymbol{r}_{ij}}\left(f_C(r_{ji})f_A(r_{ji})\right)\\
&- \frac{1}{2}\sum_{j\neq i}\sum_{k\neq i,j}f_C(r_{ij})f_A(r_{ij})\frac{\partial b_{ij}}{\partial \zeta_{ij}}\frac{\partial}{\partial \boldsymbol{r}_{ij}}\left(f_C(r_{ik})g_{ijk}\right)\\
&- \frac{1}{2}\sum_{j\neq i}\sum_{k\neq i,j}f_C(r_{ij})f_A(r_{ij})\frac{\partial b_{ij}}{\partial \zeta_{ij}}\frac{\partial}{\partial \boldsymbol{r}_{ik}}\left(f_C(r_{ik})g_{ijk}\right)\\
&- \frac{1}{2}\sum_{j\neq i}\sum_{k\neq i,j}f_C(r_{ji})f_A(r_{ji})\frac{\partial b_{ji}}{\partial \zeta_{ji}}\frac{\partial}{\partial \boldsymbol{r}_{ij}}\left(f_C(r_{jk})g_{jik}\right)\\
&- \frac{1}{2}\sum_{j\neq i}\sum_{k\neq i,j}f_C(r_{jk})f_A(r_{jk})\frac{\partial b_{jk}}{\partial \zeta_{jk}}\frac{\partial}{\partial \boldsymbol{r}_{ij}}\left(f_C(r_{ji})g_{jki}\right).
\end{aligned} \tag{16}$$

In this equation, the sum of the 1st and 4th lines is $1/2\sum_{j\neq i}\partial U_{ij}/\partial \boldsymbol{r}_{ij}$, the sum of the 2nd and 5th lines is $1/2\sum_{j\neq i}\partial U_{ji}/\partial \boldsymbol{r}_{ij}$, the 3rd line, with the dummy indices j and k interchanged, is $1/2\sum_{j\neq i}\sum_{k\neq i,j}\partial U_{ik}/\partial \boldsymbol{r}_{ij}$, and the 6th line is $1/2\sum_{j\neq i}\sum_{k\neq i,j}\partial U_{jk}/\partial \boldsymbol{r}_{ij}$. Therefore, the conventional expression Eq. (16) is equivalent to Eqs. (12) and (13).

While there is no difference between the force calculations based on the conventional and the new methods, the same cannot be said for some other quantities such as the virial stress and heat current. The total virial stress tensor is defined as

$$\mathbf{W} = \sum_i \boldsymbol{r}_i \otimes \boldsymbol{F}_i. \tag{17}$$

In MD simulations with periodic boundary conditions in one or more directions, the absolute positions cause problems and in the case of two-body potentials, one can change them to relative positions using Newton's third law. The resulting virial stress tensor takes a simple form:

$$\mathbf{W} = -\frac{1}{2}\sum_i\sum_{j\neq i}\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_{ij}. \tag{18}$$

Here, $\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji}$ is the pairwise force acting on atom i by atom j. One can also decompose the total virial stress into per-atom ones:

$$\mathbf{W} = \sum_i \mathbf{W}_i, \tag{19}$$

$$\mathbf{W}_i = -\frac{1}{2}\sum_{j\neq i}\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_{ij}. \tag{20}$$

Since pairwise forces $\boldsymbol{F}_{ij}^{\text{Tersoff}}$ also exist for the Tersoff potential, the per-atom virial stress tensor for the Tersoff potential takes the same simple form as in the case of two-body potentials:

$$\mathbf{W}_i^{\text{Tersoff}} = -\frac{1}{2}\sum_{j\neq i}\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_{ij}^{\text{Tersoff}}. \tag{21}$$

However, the existence of a pairwise force expression has not been widely recognized and the virial stress tensor in standard MD packages such as LAMMPS is not implemented in this way. Referring to Algorithm 1, the per-atom virial stress tensor as implemented in LAMMPS takes a rather complicated form:

$$\begin{aligned}\mathbf{W}_i = &-\frac{1}{2}\sum_{j\neq i}\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_i^{(ij)}\\ &+\frac{1}{3}\sum_{j\neq i}\sum_{k\neq i,j}\left(\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_j^{(ijk)}+\boldsymbol{r}_{ik}\otimes\boldsymbol{F}_k^{(ijk)}\right)\\ &+\frac{1}{3}\sum_{j\neq i}\sum_{k\neq i,j}\left(\boldsymbol{r}_{ji}\otimes\boldsymbol{F}_i^{(jik)}+\boldsymbol{r}_{jk}\otimes\boldsymbol{F}_k^{(jik)}\right)\\ &+\frac{1}{3}\sum_{j\neq i}\sum_{k\neq i,j}\left(\boldsymbol{r}_{kj}\otimes\boldsymbol{F}_j^{(kji)}+\boldsymbol{r}_{ki}\otimes\boldsymbol{F}_i^{(kji)}\right).\end{aligned}\tag{22}$$

This expression is not likely equivalent to Eq. (21). For example, the first line in this equation suggests that the force component $\boldsymbol{F}_i^{(ij)}$ has been taken to be pairwise, which is not true because $b_{ij}\neq b_{ji}$.

A related quantity is the potential part of the heat current $\boldsymbol{J}$. For two-body potentials, it can be written as

$$\boldsymbol{J}=-\frac{1}{2}\sum_i\sum_{j\neq i}\boldsymbol{r}_{ij}(\boldsymbol{F}_{ij}\cdot\boldsymbol{v}_i).\tag{23}$$

Since $\boldsymbol{r}_{ij}(\boldsymbol{F}_{ij}\cdot\boldsymbol{v}_i)=(\boldsymbol{r}_{ij}\otimes\boldsymbol{F}_{ij})\cdot\boldsymbol{v}_i$, we can also express $\boldsymbol{J}$ in terms of the per-atom virial stress tensor:

$$\boldsymbol{J}=\sum_i\mathbf{W}_i\cdot\boldsymbol{v}_i.\tag{24}$$

This is the heat current expression implemented in LAMMPS. However, as pointed out in Ref. [27], this stress-based formula does not apply to many-body potentials. The correct potential part of the heat current formula for many-body potentials reads

$$\boldsymbol{J}=\sum_i\sum_{j\neq i}\boldsymbol{r}_{ij}\left(\frac{\partial U_j}{\partial\boldsymbol{r}_{ji}}\cdot\boldsymbol{v}_i\right).\tag{25}$$

Similar to the case of virial stress, one can also decompose the total heat current into per-atom ones:

$$\boldsymbol{J}=\sum_i\boldsymbol{J}_i,\tag{26}$$

$$\boldsymbol{J}_i=\sum_{j\neq i}\boldsymbol{r}_{ij}\left(\frac{\partial U_j}{\partial\boldsymbol{r}_{ji}}\cdot\boldsymbol{v}_i\right).\tag{27}$$

From the above discussion, we see that all the relevant quantities have a simple per-atom expression. This is exactly what one needs for an efficient GPU-implementation: the per-atom quantities (U_i, $\boldsymbol{F}_i$, $\boldsymbol{W}_i$, and $\boldsymbol{J}_i$) can be accumulated solely by the thread associated with atom i and no write conflict would occur. In Algorithm 2, we present a pseudo code for the force evaluation kernel on the GPU. This algorithm is much simpler than Algorithm 1 and can be straightforwardly implemented using CUDA or OpenCL.

There is a subtle technical point for Tersoff-type potentials. As can be seen from Eq. (13), the bond-order function b_{ij} and their derivatives $\partial b_{ij}/\partial\zeta_{ij}$ will be frequently used in the force evaluation kernel. We can reduce the amount of redundant calculations by using a two-kernel approach, where the first kernel is used to precompute b_{ij} and $\partial b_{ij}/\partial\zeta_{ij}$, and the second kernel is used to perform the force evaluation as in Algorithm 2. The pseudo code for the first kernel is presented in Algorithm 3. This kernel only takes up about 20% of the computation time for the whole force evaluation, but using the two-kernel approach reduces the amount of calculations for b_{ij} and $\partial b_{ij}/\partial\zeta_{ij}$ by a factor of about M, where M is the maximum number of neighbours per atom. In contrast, there is no need of precomputing for the SW potential, and only a single kernel is needed for the force evaluation.

Algorithm 2 Pseudo code for the force evaluation kernel for many-body potentials in GPUMD.

Require: b is the block index
Require: t is the thread index
Require: S_b is the block size
Require: $i = S_b \times b + t$
Initialize U_i, $\boldsymbol{F}_i$, $\mathbf{W}_i$, and $\boldsymbol{J}_i$ to zero
if $i < N$ **then**
  Read in $\boldsymbol{r}_i$ from global memory
  Read in $\boldsymbol{v}_i$ from global memory
  for $m = 0$ to $\mathrm{NN}_i - 1$ **do**
    $j \leftarrow \mathrm{NL}_{im}$
    Read in $\boldsymbol{r}_j$ from global memory and calculate $\boldsymbol{r}_{ij}$
    $\boldsymbol{r}_{ij} \leftarrow$ minimum image of $\boldsymbol{r}_{ij}$
    Calculate U_{ij}
    Calculate $\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}}$ (with a for-loop over neighbours of i)
    Calculate $\frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}$ (with a for-loop over neighbours of j)
    $U_i \leftarrow U_i + \frac{1}{2}U_{ij}$
    $\boldsymbol{F}_i \leftarrow \boldsymbol{F}_i + \left(\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} - \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}\right)$
    $\mathbf{W}_i \leftarrow \boldsymbol{W}_i - \frac{1}{2}\boldsymbol{r}_{ij} \otimes \left(\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} - \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}\right)$
    $\boldsymbol{J}_i \leftarrow \boldsymbol{J}_i + \boldsymbol{r}_{ij}\left(\frac{\partial U_j}{\partial \boldsymbol{r}_{ji}} \cdot \boldsymbol{v}_i\right)$
  end for
  Save the per-atom quantities U_i, $\boldsymbol{F}_i$, $\mathbf{W}_i$, and $\boldsymbol{J}_i$ to global memory
end if

Algorithm 3 Pseudo code for precomputing the bond-order functions and their derivatives for Tersoff-type potentials in GPUMD.

Require: b is the block index
Require: t is the thread index
Require: S_b is the block size
Require: $i = S_b \times b + t$
if $i < N$ **then**
  Read in $\boldsymbol{r}_i$ from global memory
  for $m = 0$ to $\mathrm{NN}_i - 1$ **do**
    $j \leftarrow \mathrm{NL}_{im}$
    Read in $\boldsymbol{r}_j$ from global memory and calculate $\boldsymbol{r}_{ij}$
    $\boldsymbol{r}_{ij} \leftarrow$ minimum image of $\boldsymbol{r}_{ij}$
    Calculate ζ_{ij} (with a for-loop over neighbours of i)
    Calculate b_{ij} and $\partial b_{ij}/\partial\zeta_{ij}$
    Save b_{ij} and $\partial b_{ij}/\partial\zeta_{ij}$ to global memory
  end for
end if

2.4. Generalization to other many-body potentials

The above formalism for the Tersoff potential also applies to other many-body potentials. In Ref. [27], it has been shown that for any many-body potential, the force, virial stress tensor, and heat current have the following per-atom forms:

$$\boldsymbol{F}_i^{\text{many-body}}=\sum_j\left(\frac{\partial U_i}{\partial\boldsymbol{r}_{ij}}-\frac{\partial U_j}{\partial\boldsymbol{r}_{ji}}\right),\tag{28}$$

$$\mathbf{W}_i^{\text{many-body}}=-\frac{1}{2}\sum_{j\neq i}\boldsymbol{r}_{ij}\otimes\left(\frac{\partial U_i}{\partial\boldsymbol{r}_{ij}}-\frac{\partial U_j}{\partial\boldsymbol{r}_{ji}}\right),\tag{29}$$

$$\boldsymbol{J}_i^{\text{many-body}}=\sum_{j\neq i}\boldsymbol{r}_{ij}\left(\frac{\partial U_j}{\partial\boldsymbol{r}_{ji}}\cdot\boldsymbol{v}_i\right).\tag{30}$$

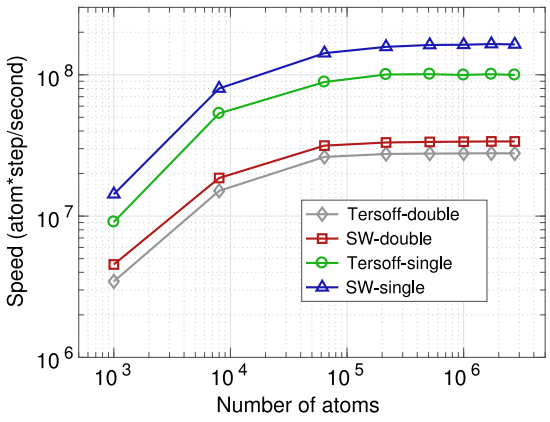

Fig. 1. (Colour online) Computational speed of GPUMD as a function of number of atoms for Tersoff and Stillinger–Weber potentials using double- or single-precision. The test system is silicon crystal at 300 K and zero pressure. A Tesla K40 card is used to run the code.

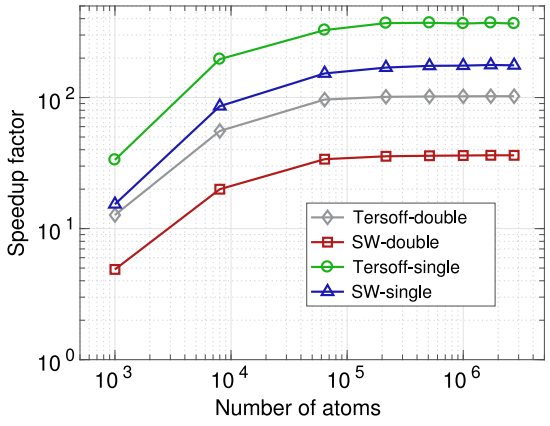

Fig. 2. (Colour online) Speedup factor of GPUMD (running on a Tesla K40 GPU) as a function of the number of atoms with respect to the serial version of LAMMPS running on Intel Xeon CPU X5670 @ 2.93 GHz. The test system is silicon crystal at 300 K and zero pressure.

Thanks to these per-atom expressions, one can construct a force evaluation CUDA kernel for any many-body potential without worrying about write conflicts.

2.5. Comparison with previous works

Our force evaluation algorithm should be largely equivalent to that proposed by Brown and Yamada [21] for the SW potential, which was recently generalized to Tersoff-type potentials by Nguyen [25]. For example, the first and last two `attractive` functions in Listing 1 of Ref. [25] should be equivalent to lines 10 and 11, respectively, in our Algorithm 2. However, we stress that (1) Our algorithm is original and general, which is based on the formalisms developed in Ref. [27]; (2) Useful quantities such as per-atom virial stress and heat current can also be unambiguously and efficiently calculated along with the force evaluation in our algorithm; (3) We only need a single kernel (except for Tersoff-type potentials where we intentionally precompute the bond order functions) for force evaluation, while more kernels are used in previous works [21,25].

3. Performance evaluation

We have implemented the force evaluation kernels for the Tersoff and Stillinger–Weber potentials into our GPUMD code using CUDA, and the code has already been used to study various problems in large-scale (up to a few million atoms) graphene-based materials [31–34]. Systematic applications on thermal transport calculations have been presented in Ref. [27]. Here, we measure the performance of GPUMD and compare it with that of LAMMPS, which is the standard production code for simulations with many-body potentials.

Simulations with LAMMPS (version 9 Dec 2014) are performed using the "TianHe-1A" supercomputer. Each compute node has two CPUs, which are Intel Xeon X5670 @ 2.93 GHz released in 2010. Each CPU has 6 cores, 16 GB memory, a peak performance of 70 GFLOPS, and a maximum power consumption of 95 W. Simulations with GPUMD are performed using a workstation equipped with a Tesla K40 GPU released in 2013, which has 12 GB device memory, a peak performance of 1.4 TFLOPS in double-precision arithmetic and 4.3 TFLOPS in single-precision arithmetic, and a maximum power consumption of 235 W.

The test system is silicon crystal of cubic domain with the number of atoms varying from $N = 5 \times 5 \times 5 \times 8 = 1000$ to $N = 70 \times 70 \times 70 \times 8 = 2\,744\,000$. For each domain size, we run a simulation in the *NPT* ensemble (300 K and 0 Pa, using the Berendsen weak-coupling thermostat and barostat, and the velocity-Verlet integration approach) for 1000 steps and record the computation time $t_{\rm comp}$ (in unit of second). The computational speed is then calculated as $N \times 1000/t_{\rm comp}$, which has a unit of atom × step / second. A Verlet neighbour list (each atom has 4 neighbours) is constructed in the beginning and is not rebuilt during the time evolution (there is one exception, though, which will be stated later). The time-integration and the force-evaluation parts take up about 10% and 90% of the whole computation time, respectively.

Fig. 1 shows the scaling of the computational speed of GPUMD with respect to the number of atoms for the Tersoff and Stillinger–Weber potentials, with either double- or single-precision, using a Tesla K40 GPU. For all the cases, the speed increases quickly with increasing N and almost saturates when N exceeds 10^5. The single-precision version for each potential is about 3-4 times as fast as the double-precision version. This is partly due to the faster single-precision floating point arithmetic, and partly due to the fact that the double-precision version uses more registers in the force-evaluation kernel and has lower GPU occupancy. Detailed profiling shows that the occupancy is about 50% for the single-precision version and about 25% for the double-precision version. Despite these relatively low occupancies, the computational speeds shown in Fig. 1 are impressive. For example, the single-precision version of the Stillinger–Weber potential can achieve a speed of 25 ns per day (with a time step of 2 fs) for a system with 10^6 atoms.

To better appreciate the high performance achieved by GPUMD, we compare its performance against that obtained by LAMMPS running on a single CPU core, which is Intel Xeon CPU X5670 @ 2.93 GHz. The single-core speed of LAMMPS on this CPU model is 2.7×10^5 atom × step / second for the Tersoff potential and 9.3×10^5 atom × step / second for the SW potential. The speedup factors for GPUMD running on a Tesla K40 GPU are presented in Fig. 2. For relatively large systems, the speedup factors range from a few tens to a few hundred, depending on the types of potential and floating point arithmetic. We note that the CPU available to us is about 3 years older than the GPU used and using a CPU released at the same year as the GPU would roughly double the LAMMPS speeds and halve the speedup factors achieved by GPUMD.

To give a more realistic comparison between the performance of GPUMD and the CPU version of LAMMPS, we consider a system of 512 000 atoms and run the LAMMPS code with varying number (from 1 to 256) of CPU cores of the same specification as above using MPI parallelism. It can be seen from Fig. 3 that the MPI version of LAMMPS scales quite well up to about 100 CPU cores, but starts

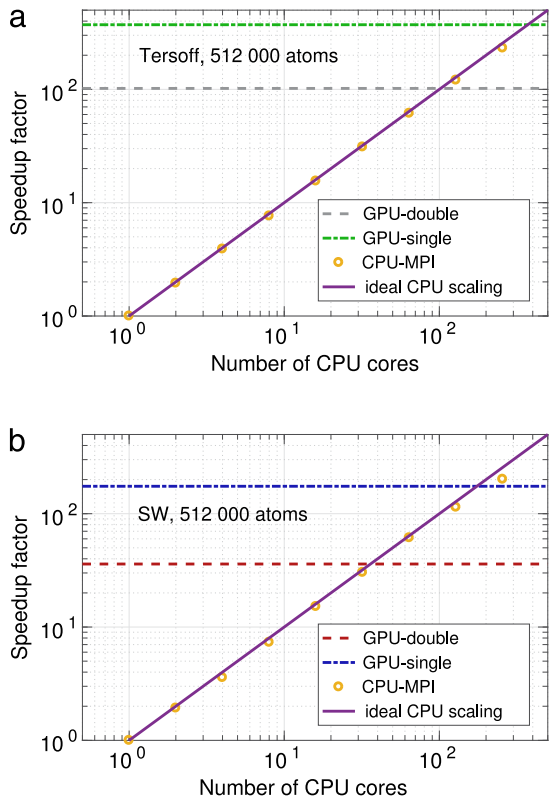

Fig. 3. (Colour online) The markers show the scaling of the performance (relative to a single CPU core) of the MPI version of LAMMPS with respect to the number of CPU cores (Intel Xeon CPU X5670 @ 2.93 GHz). The dashed lines show the performance of GPUMD running on a Tesla K40 GPU. Panel (a) refers to the Tersoff potential and panel (b) the Stillinger–Weber potential. The solid line in each panel indicates the ideal scaling that can be achieved by MPI parallelism. The test system is silicon crystal with 512 000 atoms.

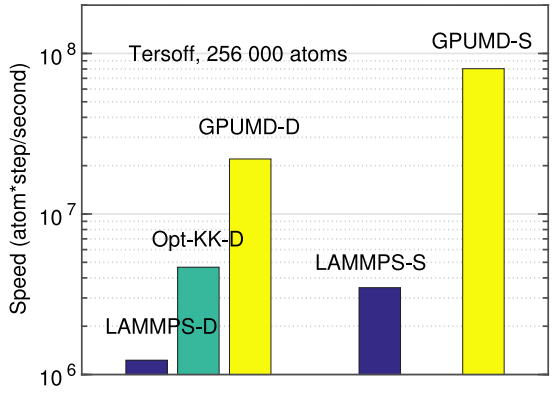

Fig. 4. (Colour online) A comparison between the performances of GPUMD (labelled as GPUMD-D and GPUMD-S for single- and double-precision), a GPU version of LAMMPS (labelled as LAMMPS-D and LAMMPS-S for single- and double-precision), and an optimized version by Höhnerbach et al. [23] based on LAMMPS (labelled as Opt-KK-D for double-precision). The test system is silicon crystal with 256 000 atoms and the Tersoff potential is used. The data for LAMMPS-S, LAMMPS-D, and Opt-KK-D are taken from Ref. [30] provided by Höhnerbach et al. All the codes run on a Tesla K40 GPU.

to scale less ideally afterwards. Taking the Tersoff potential as an example, the computational speeds for the double- and single-precision versions of GPUMD are equivalent to those of LAMMPS with about 100 and 500 CPU cores, respectively.

Finally, Fig. 4 also gives a comparison between the performance of GPUMD and those of some available GPU versions of LAMMPS (taken from Ref. [30]) using the Tersoff potential. Here, to be consistent with the simulations by Höhnerbach et al. [23], we update the neighbour list every 5 time steps, which reduces the performance by about 20% compared to the case without rebuilding the neighbour list. For both double- and single-precision, GPUMD is more than one order of magnitude faster than the original version of LAMMPS. The optimized version by Höhnerbach et al. [23] is a few times faster than the original version of LAMMPS, but is still a few times slower than GPUMD.

4. Conclusions

In summary, we analysed the difficulty in implementing many-body potentials in MD simulations on graphics processing units and presented an efficient algorithm based on an explicit pairwise force expression for many-body potentials. In this algorithm, the virial stress tensor and the heat current also have well-defined per-atom expressions. Therefore, the force, virial stress, and heat current for a given atom can be accumulated within in a single thread and the algorithm is free of write conflict by construction. This crucial property allows for a simple, flexible, and efficient implementation of any many-body potential on the GPU.

We have implemented the algorithm for the Tersoff and Stillinger–Weber potentials in our GPUMD code, which has excellent performance. GPUMD running on a single Tesla K40 GPU can be as fast as LAMMPS running with tens to hundreds of CPU cores. Our code is available upon request and will be made public later.

Acknowledgements

This work was supported by the Academy of Finland through its Centres of Excellence Programme (2015-2017) under project number 284621 and National Natural Science Foundation of China under Grant Nos. 11404033 and 11504384. We acknowledge the computational resources provided by Aalto Science-IT project, Finland's IT Center for Science (CSC), and China Scientific Computing Grid (ScGrid). We thank the great help from the GPU experts from CSC and NVIDIA during the GPU hackathon organized by Sebastian von Alfthan.

References

[1] A.W. Götz, M.J. Williamson, D. Xu, D. Poole, S. Le Grand, R.C. Walker, J. Chem. Theory Comput. 8 (2012) 1542–1555.
[2] S. Páll, B. Hess, Comput. Phys. Comm. 184 (2013) 2641–2650.
[3] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B.L. de Groot, H. Grubmüller, J. Comput. Chem. 36 (2015) 1990–2008.
[4] S. Plimpton, J. Comput. Phys. 117 (1995) 1–19.
[5] W.M. Brown, P. Wang, S.J. Plimpton, A.N. Tharrington, Comput. Phys. Comm. 182 (2011) 898–911.
[6] W.M. Brown, A. Kohlmeyer, S.J. Plimpton, A.N. Tharrington, Comput. Phys. Comm. 183 (2012) 449–459.
[7] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, J. Comput. Chem. 28 (2007) 2618–2640.
[8] P. Eastman, V.S. Pande, J. Comput. Chem. 31 (2010) 1268–1272.
[9] J.A. Anderson, C.D. Lorenz, A. Travesset, J. Comput. Phys. 227 (2008) 5342–5359.
[10] J.A. Anderson, S.C. Clotzer, The development and expansion of HOOMD-blue through six years of GPU proliferation. arXiv:13085587v1.
[11] J. Glaser, T.D. Nguyen, J.A. Anderson, P. Lui, F. Spiga, J.A. Millan, D.C. Morse, S.C. Glotzer, Comput. Phys. Comm. 192 (2015) 97–107.
[12] P.H. Colberg, F. Höfling, Comput. Phys. Comm. 182 (2011) 1120–1129.
[13] N.P. Bailey et al., RUMD: A general purpose molecular dynamics package optimized to utilize GPU hardware down to a few thousand particles. arXiv:1506.05094v2.
[14] I.V. Morozov, A.M. Kazennov, R.G. Bystryi, G.E. Norman, V.V. Pisarev, V.V. Stegailov, Comput. Phys. Comm. 182 (2011) 1974–1978.
[15] W.M. Brown, T.D. Nguyen, M. Fuentes-Cabrera, J.D. Fowlkes, P.D. Rack, M. Berger, A.S. Bland, Procedia Comput. Sci. 9 (2012) 186–195.
[16] Q. Hou, M. Li, Y. Zhou, J. Cui, Z. Cui, J. Wang, Comput. Phys. Comm. 184 (2013) 2091–2101.

[17] J. Tersoff, Phys. Rev. B 39 (1989) 5566–5568.
[18] F.H. Stillinger, T.A. Weber, Phys. Rev. B 31 (1985) 5262–5271.
[19] D.W. Brenner, O.A. Shenderova, J.A. Harrison, S.J. Stuart, B. Ni, S.B. Sinnott, J. Phys.: Condens. Matter 14 (2002) 783–802.
[20] C. Hou, J. Xu, P. Wang, W. Huang, X. Wang, Comput. Phys. Comm. 184 (2013) 1364–1371.
[21] W.M. Brown, M. Yamada, Comput. Phys. Comm. 184 (2013) 2785–2793.
[22] A.A. Knizhnik, A.S. Minkin, B.V. Potapkin, Comput. Res. Model. 7 (2015) 549–558.
[23] M. Höhnerhach, A.E. Ismail, P. Bientinesi, The vectorization of the Tersoff multi-body potential: An exercise in performance portability. arXiv:1607.02904v1.
[24] P. Tredak, W.R. Rudnicki, J.A. Majewski, J. Comput. Phys. 321 (2016) 556–570.
[25] T.D. Nguyen, Comput. Phys. Comm. 212 (2017) 113–122.
[26] http://docs.nvidia.com/cuda/cuda-c-programming-guide/.
[27] Z. Fan, L.F.C. Pereira, H.-Q. Wang, J.-C. Zheng, D. Donadio, A. Harju, Phys. Rev. B 92 (2015) 094301.
[28] Z. Fan, T. Siro, A. Harju, Comput. Phys. Comm. 184 (2013) 1414–1425.
[29] M.P. Howard, J.A. Anderson, A. Nikoubashman, S.C. Glotzer, A.Z. Panagiotopoulos, Comput. Phys. Comm. 203 (2016) 45–52.
[30] http://github.com/HPAC/lammps-tersoff-vector.
[31] P. Hirvonen, M.M. Ervasti, Z. Fan, M. Jalalvand, M. Seymour, S. M.V. Allaei, N. Provatas, A. Harju, K.R. Elder, T. Ala-Nissila, Phys. Rev. B 94 (2016) 035414.
[32] B. Mortazavi, Z. Fan, L.F.C. Pereira, A. Harju, T. Rabczuk, Carbon 103 (2016) 318–326.
[33] Z. Fan, A. Uppstu, A. Harju, 2D Mater 4 (2017) 025004.
[34] Z. Fan, L.F.C. Pereira, P. Hirvonen, M.M. Ervasti, K.R. Elder, D. Donadio, T. Ala-Nissila, A. Harju, Phys. Rev. B 95 (2017) 144309.