КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2015 Т. 7 № 3 С. 549–558

Ки&М

СЕКЦИОННЫЕ ДОКЛАДЫ

УДК: 004.43, 004.94

OpenCL realization of some many-body potentials

A. A. Knizhnik[1,2,a], A. S. Minkin[1,b], B. V. Potapkin[1,2,c]

[1]National Research Center "Kurchatov Institute", Kurchatov Sq. 1, Moscow, 123182, Russia
[2]Kintech Lab Ltd, Kurchatov Sq. 1, Moscow 123182, Russia

E-mail: [a]knizhnik@kintechlab.com, [b]amink@mail.ru, [c] potapkin@kintechlab.com

Received October 27, 2014

Modeling of carbon nanostructures by means of classical molecular dynamics requires a lot of computations. One of the ways to improve the performance of basic algorithms is to transform them for running on SIMD-type computing systems such as systems with dedicated GPU. In this work we describe the development of algorithms for computation of many-body interaction based on Tersoff and embedded-atom potentials by means of OpenCL technology. OpenCL standard provides universality and portability of the algorithms and can be successfully used for development of the software for heterogeneous computing systems. The performance of algorithms is evaluated on CPU and GPU hardware platforms. It is shown that concurrent memory writes is effective for Tersoff bond order potential. The same approach for embedded-atom potential is shown to be slower than algorithm without concurrent memory access. Performance evaluation shows a significant GPU acceleration of energy-force evaluation algorithms for many-body potentials in comparison to the corresponding serial implementations.

Keywords: GPGPU; OpenCL; many-body potentials; Tersoff potential; embedded-atom potential; atomic operations

Реализация алгоритмов межатомного взаимодействия с использованием технологии OpenCL

А. С. Минкин[1], А. А. Книжник[1,2], Б. В. Потапкин[1,2]

[1]Национальный исследовательский центр "Курчатовский институт", Россия, 123182, г. Москва, пл. Академика Курчатова, д. 1
[2]Кинтех Лаб, Россия, 123182, г. Москва, пл. Академика Курчатова, д. 1

Моделирование углеродных наноструктур методом классической молекулярной динамики требует больших объемов вычислений. Один из способов повышения производительности соответствующих алгоритмов состоит в их адаптации для работы с SIMD-подобными архитектурами, в частности, с графическими процессорами. В данной работе рассмотрены особенности алгоритмов вычисления многочастичного взаимодействия на основе классических потенциалов Терсоффа и погруженного атома с использованием технологии OpenCL. Стандарт OpenCL позволяет обеспечить универсальность и переносимость алгоритмов и может быть эффективно использован для гетерогенных вычислений. В данной работе сделана оценка производительности OpenCL алгоритмов вычисления межатомного взаимодействия для систем на базе центральных и графических процессоров. Показано, что использование атомарных операций эффективно для вычисления потенциала Терсоффа и неэффективно в случае потенциала погруженного атома. Оценка производительности показывает значительное ускорение GPU реализации алгоритмов вычисления потенциалов межатомного взаимодействия по сравнению с соответствующими однопоточными алгоритмами.

Ключевые слова: GPGPU; OpenCL; многочастичные потенциалы взаимодействия; потенциал Терсоффа, потенциал погруженного атома; атомарные операции

Citation: *Computer Research and Modeling*, 2015, vol. 7, no. 3, pp. 549–558.

Introduction

Atomistic modeling is computationally intensive problem. Serial realization of algorithms can be done only for modeling of small system of atoms. Cluster type computing systems used in most cases to solve the problem of high computational complexity can be good but sometimes have problems with long time waiting for resource. Besides creating effective MIMD algorithms is a real challenge. The alternative way is optimization of existing algorithms for SIMD/SIMT hardware devices such as graphic processing units (GPU). GPU is a part of contemporary clusters and heterogeneous systems. Using such devices allows increasing the performance of algorithms along with decreasing of the power consumption. Such approach is good in sense higher availability of computing resources as some hardware such as personal computer with dedicated GPU or GPU-based GRID systems can be built.

The main problem of using heterogeneous parallel systems is creating of the effective algorithms by means of appropriate memory access patterns and load balancing. But the optimal algorithms should take into account the architecture of computing system so we are coming to development of algorithm for specific devices. That way is proposed by CUDA technology [Jason Sanders, Edward Kandrot, 2010]. CUDA is designed for NVidia GPUs only and does not assume free support of CPU computation. Such approach is very tedious and one prefers to have an algorithm for all devices. There are also some new technologies. For example, such technology as OpenACC give OpenMP-like pragmas for GPGPU adaptation but actually it is not free. OpenMP 4.0 is supported by gcc but it is under development and not available for all platforms. The only crossplatform technology with free SDKs available for all popular operating systems is OpenCL [Gaster et al, 2011]. It is supported by most of hardware vendors, gives access to most of GPU features via the frontend and makes it possible to use the same kernel code with different devices. So we come to a compromise between the performance and universality. We use OpenCL technology for CPU-based systems and for GPU programming as GPUs are the devices with the best ratio of performance and power consumption. The latter is also important in Russia as the reforms of 90-th result in no hope that electrical energy would ever be less expensive.

GPU are basically computer graphics devices and they are optimal for rendering. But with unified architecture we have a capability of GPGPU computation. There are the following advantages of GPU over traditional CPU systems:

- Large number of processor cores.
- Threads are lightweight and large number of threads hides global memory latency.
- The L1 cache (local memory) is available for direct access by programmer.
- Large memory bandwidth.
- Complete RISC architecture without CISC converter.

These advantages can be got for good use in computing as long as appropriate technology would be applied. OpenCL was chosen as such GPGPU technology.

There are some algorithms that can be effectively done by means of SIMD-similar hardware. One of such problems is molecular dynamics simulation. The critical aspect of classical molecular dynamics [Allen, Tildesley, 1990] and Monte-Carlo methods [Bielajew, 2001] is the choice of an appropriate energy function (potential) for describing the interatomic interactions. Our objective is modeling of carbon nanostructures such as graphene, fullerenes and nanotubes. So we need special potentials for adequate computation of their energetic. The most of these potentials are many-body. In this work we evaluate the performance of OpenCL algorithms of Tersoff potential in comparison to EAM potential for modeling of metallic systems.

Complicated form and larger set of parameters of many-body potentials results in large computational requirements. So the acceleration of existing algorithms of interatomic interaction is especially needed.

Interatomic potentials

In atomistic modeling quantum chemistry is the best way to reproduce experimental results but it is usually used for simulation of relatively small systems of atoms. We have about 10^{23} atoms in 2

gramms of carbon. So we need Exascale supercomputer for modeling of real patterns and that become almost impossible by now to do that via quantum mechanical approach. Classical and semi-classical simulations are a real compromise and they are used for modeling of relatively large atomic systems.

Interatomic potential can be described by the potential function. The potential function $U(\mathbf{r}_1,\mathbf{r}_2, \ldots, \mathbf{r}_N)$ is the dependence of the potential energy of N-atom system on their coordinates. The forces in MD simulation are defined by the potential,

$$\mathbf{F}_l = -\frac{\partial U(\mathbf{r}_1,\mathbf{r}_2,...,\mathbf{r}_N)}{\partial \mathbf{r}_l}, \tag{1}$$

where $\mathbf{r}_l$ is a position (3D vector) of the atom l, $\mathbf{F}_l$ — force (3D vector) acting on the atom l.

The most popular types of potentials used in numerical software are pair potentials. The interaction of any pair of atoms depends only on their spacing and is not affected by the presence of other atoms. Full potential energy in that case is the sum of pair interactions:

$$U(\mathbf{r}_1,\mathbf{r}_2,...,\mathbf{r}_N) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1\,(j\neq i)}^{N} U_{ij}^{pair}(r_{ij}), \tag{2}$$

where r_{ij} is a distance between atom i and atom j. The most popular pair potential implemented in the most MD software is Lennard-Jones (LJ) potential.

Though it can be effectively implemented on GPU [Anderson, Lorenz, 2010] it is not good for modeling of specific systems such as carbon nanostructures and metallic systems. Indeed, low density structures of covalent systems are instable with pair potentials.

That is why for modeling of covalent crystals, carbon nanostructures, metallic systems and polymers more complicated many-body potentials [Erkoc, 1997] are widely used. The set of parameters of many-body potentials are usually derived via quantum mechanical methods [Kumagai et al., 2009, Lebedeva et al., 2012]. That is why many-body potentials are much more accurate and can reproduce mechanical and transport phonon properties of carbon and metallic nanostructures.

A general form of many-body potentials such as Tersoff potential [Tersoff, 1989] and embedded-atom potential [Murray, Foiles, 1993] can be written as a sum of pair and many-body terms [Brenner, 1989]:

$$U_i = \frac{1}{2}\sum_{j\neq i} U_{ij}^{pair}\left(r_{ij}\right) + U_i^{mb}, \tag{3}$$

where U_i is an energy per atom i.

Embedded-atom potential takes into account many-body interaction by means of nonlinear embedding function in energy functional:

$$U_i^{mb} = E_i(\rho_i), \rho_i = \sum_{j\neq i} c_j\left(r_{ij}\right), \tag{4}$$

where ρ_i is an electron density per atom i, E_i is embedded energy per atom i, $c_j(r_{ij})$ represents the influence of atom j to the electron density (ρ_i) of atom i.

The main advantage of many-body potentials over pair potentials is the ability to describe the variation of the bond strength with coordination. Tersoff and Tersoff-Brenner potentials can reproduce the same mechanism by means of bond order formalism. Bond order function includes angle dependence that makes Tersoff potential many-body. The simplest functional form of Tersoff potential can be written in the following form:

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

$$U_{ij}^{pair}\left(r_{ij}\right)=V_R\left(r_{ij}\right),$$

$$U_i^{mb}=-\frac{1}{2}\sum_{j\neq i}b_{ij}V_A\left(r_{ij}\right),$$

$$U_{ij}=U_{ij}^{pair}+U_{ij}^{mb}=V_R\left(r_{ij}\right)-b_{ij}V_A\left(r_{ij}\right),$$

$$V_R\left(r_{ij}\right)=f_{ij}\left(r_{ij}\right)A_{ij}\exp\left(-\lambda_{1,ij}r_{ij}\right), \qquad \text{[repulsive term]}$$

$$V_A\left(r_{ij}\right)=f_{ij}\left(r_{ij}\right)B_{ij}\exp\left(-\lambda_{2,ij}r_{ij}\right), \qquad \text{[attractive term]}$$

$$b_{ij}=\left[1+\beta^n\varsigma_{ij}^n\right]^{-\frac{1}{2n}}, \qquad \text{[bond order function]}$$

$$\varsigma_{ij}=\sum_{k(\neq i,j)}G_i\left(\theta_{ijk}\right)f_{ik}\left(r_{ik}\right)\exp\left[\lambda_{3,ijk}^m\left(r_{ij}-r_{ik}\right)^m\right],$$

$$G_i\left(\theta_{ijk}\right)=\gamma_{ijk}\left[1+\frac{c^2}{d^2}-\frac{c^2}{d^2+\left(\cos\theta-\cos\theta_0\right)^2}\right], \qquad \text{[angle dependence function]}$$

$$f_{ik}\left(r_{ik}\right)=\begin{cases}1, \text{if } r_{ik}<R-D,\\ \frac{1}{2}\left[1-\sin\left[\frac{\pi\left(r-R\right)}{2D}\right]\right], \text{if } R-D\le r_{ik}\le R+D,\\ 0, \text{if } r_{ik}>R+D.\end{cases} \qquad \text{[cutoff function]}$$

The Tersoff potential is short-ranged as it uses the cutoff function. So the local environment has limited number of atoms. That is why neighbor list can be used to represent the local environment [Allen, Tildesley, 1990]. Neighbor lists allow to speed up the computation of interatomic interaction by eliminating of N^2 search. That is especially important for computational expensive many-body potentials.

So many-body potentials have the following advantages:

- That is more adequate model of interatomic interaction than pair potentials. So more macroscopic parameters and physical constants can be represented by many-body potentials as they have more complicated parametric form.
- Appropriate parallel algorithms can be effectively designed because of interaction locality.

The disadvantages of many-body potentials are concerned with the need to match a lot ofconstants customized just for specific chemical compounds and their computational complexity.

The problem of high computational complexity of many-body potentials can be solved by the adaptation of the algorithms to such computing architectures as GPU and accelerators. Unfortunately the basic serial algorithms cannot be used on GPU without adaptation as some features of the accelerator architecture such as memory hierarchy are needed to be taken into account [Jason Sanders, Edward Kandrot, 2010, Gaster et al., 2011].

The following subsections are dedicated to description of various algorithms with concurrent memory access without it.

GPU algorithms for Tersoff potential

GPU algorithms for potential can be represented by computation of per atom energy along with force acting on atom. So we have parallel thread for each atom in systems. The more threads we have the best as global memory latency would be hided.

Parallel GPU algorithm for Tersoff potential can be represented in two forms:

1. Algorithm with atomic operations with memory (Algorithm A).

2. Algorithm without atomics (Algorithm WA).

Algorithm A can be represented as a transformation of the serial algorithm in the following form:

- Algorithm A is assumed to run for N work-items in 1D index space (*N* GPU threads).
- Each thread *i*

1. calculates U_i:

$$U_i = \frac{1}{2}\sum_{j\neq i} V_R\left(r_{ij}\right) - \frac{1}{2}\sum_{j\neq i} b_{ij} V_A\left(r_{ij}\right);$$

where *j* are the numbers of neighboring atoms of atom *i* (*j*≠*i*);

2. calculates $\mathbf{F}_i$:

$$\mathbf{F}_i = \mathbf{F}_i - \sum_j \frac{\partial V_R\left(r_{ij}\right)}{\partial \mathbf{r}_i} + \frac{1}{2}\sum_j b_{ij}\frac{\partial V_A\left(r_{ij}\right)}{\partial \mathbf{r}_i} + \frac{1}{2}\sum_{j,k}\frac{\partial b_{ij}}{\partial \mathbf{r}_i} V_A\left(r_{ij}\right), \frac{\partial b_{ij}}{\partial \mathbf{r}_i} = p(\varsigma_{ij})\cdot\frac{\partial \varsigma_{ij}}{\partial \mathbf{r}_i},$$

3. Calculates and sums with atomic operations $\mathbf{F}_j$ и $\mathbf{F}_k$ (*j* and *k* are the numbers of neighboring atoms of atom *i* (*j*≠*i, k*≠*i*):

$$\mathbf{F}_j = \mathbf{F}_j - \frac{1}{2} b_{ij}\frac{\partial V_A\left(r_{ij}\right)}{\partial \mathbf{r}_i} + \frac{1}{2}\sum_k \frac{\partial b_{ij}}{\partial \mathbf{r}_j} V_A\left(r_{ij}\right), \frac{\partial b_{ij}}{\partial \mathbf{r}_j} = p(\varsigma_{ij})\cdot\frac{\partial \varsigma_{ij}}{\partial \mathbf{r}_j},$$

$$\mathbf{F}_k = \mathbf{F}_k + \frac{1}{2}\sum_k \frac{\partial b_{ij}}{\partial \mathbf{r}_k} V_A\left(r_{ij}\right), \frac{\partial b_{ij}}{\partial \mathbf{r}_k} = p(\varsigma_{ij})\cdot\frac{\partial \varsigma_{ij}}{\partial \mathbf{r}_k},$$

$$p(\varsigma_{ij}) = -\frac{1}{2}\cdot\left[1+\left(\beta\varsigma_{ij}\right)^n\right]^{\frac{1}{2n}-1}\left(\beta\varsigma_{ij}\right)^n / \varsigma_{ij};$$

Atomic operations are intended to summarize correctly the force contributions to the interaction in parallel streams. In this case, memory is a shared resource and atomicity is achieved in several stages: blocking of the resource, summation, release of the resource. Three of these steps provide exclusive access to a portion of memory and atomicity prevents wrong updates, i.e. atomic operation is either successful or returns the occupation of the shared resource. In the case of occupation, the update of the memory cell by the other stream is delayed, i.e. the access of multiple threads to the same memory location serializes, resulting in a relative decrease of parallel efficiency.

Algorithm A is the most evident modification of serial algorithm but it can be transformed in the form without atomic operations with memory. That is possible for most potentials but for the sake of increasing the number of operations inside the thread. We need atomic operations with memory only to compute forces. In case of energy no concurrent access is needed. But the force computation can also be done without atomics as following (Algorithm WA):

- Algorithm WA is assumed to run for N work-items in 1D index space (*N* GPU threads).
- Each thread *i*

1. Calculates

$$U_i = \frac{1}{2}\sum_{j\neq i} U_{ij}.$$

2. Calculates

$$\mathbf{F}_i = -\frac{1}{2}\sum_{j\neq i}\left(\frac{\partial U_{ij}}{\partial \mathbf{r}_i} + \frac{\partial U_{ji}}{\partial \mathbf{r}_i}\right) - \frac{1}{2}\sum_{j\neq i,k\neq i}\frac{\partial U_{jk}}{\partial \mathbf{r}_i};$$

$$\frac{\partial U_{ij}}{\partial \mathbf{r}_i} = \frac{\partial V_R(r_{ij})}{\partial \mathbf{r}_i} - b_{ij}\frac{\partial V_A(r_{ij})}{\partial \mathbf{r}_i} - \frac{\partial b_{ij}}{\partial \mathbf{r}_i}V_A(r_{ij}),\ \frac{\partial b_{ij}}{\partial \mathbf{r}_i} = p(\varsigma_{ij})\cdot\frac{\partial \varsigma_{ij}}{\partial \mathbf{r}_i};$$

$$\frac{\partial U_{ji}}{\partial \mathbf{r}_i} = \frac{\partial V_R(r_{ij})}{\partial \mathbf{r}_i} - b_{ji}\frac{\partial V_A(r_{ij})}{\partial \mathbf{r}_i} - \frac{\partial b_{ji}}{\partial \mathbf{r}_i}V_A(r_{ij}),\ \frac{\partial b_{ji}}{\partial \mathbf{r}_i} = p(\varsigma_{ji})\cdot\frac{\partial \varsigma_{ji}}{\partial \mathbf{r}_i};$$

$$\frac{\partial U_{jk}}{\partial \mathbf{r}_i} = -\frac{\partial b_{jk}}{\partial \mathbf{r}_i}V_A(r_{jk}),\ \frac{\partial b_{jk}}{\partial \mathbf{r}_i} = p(\varsigma_{jk})\cdot\frac{\partial \varsigma_{jk}}{\partial \mathbf{r}_i}.$$

The Algorithm WA has the following features:

1. The neighbor list must be calculated with double cut radius (two coordination radius) for correct calculation of $\partial b_{ji}/\partial \mathbf{r}_i$ as

$$b_{ji} = \left[1+\beta^n\varsigma_{ji}^n\right]^{-\frac{1}{2n}} = \left[1+\beta^n\left(\sum_{k(\neq j,i)} G_j(\theta_{jik}) f_{jk}(r_{jk})\exp\left[\lambda_{3,jik}^m (r_{ji}-r_{jk})^m\right]\right)^n\right]^{-\frac{1}{2n}}$$

has the functional dependence from the $f_{jk}(r_{jk})$ term, i.e. the whole neighbor list of the atom i needs also to include all atoms j with $r_{ij} > R+D$ and $r_{jk} < R+D$.

2. Additional calculations of b_{ij}, b_{ji}, $\partial b_{ij}/\partial \mathbf{r}_i$, $\partial b_{ji}/\partial \mathbf{r}_i$ and $\partial b_{jk}/\partial \mathbf{r}_i$ must be done inside of each thread. So we have more computations than in the Algorithm A.
3. The present version of the Algorithm WA has more local variable per kernel than the Algorithm A. The limitation of register number per thread (64 registers for NVidia Fermi architecture) results in significant register spilling.

So we need extra computations and global memory transactions for Tersoff potential for the sake of eliminating of atomic operations.

GPU algorithms for embedded-atom potential

Many-body embedded-atom potential can also be implemented on GPU with atomic operations (Algorithm A):

- Algorithm A is assumed to run for N work-items in 1D index space работ (N GPU threads).
- Each thread i
  1. Calculates

$$U_i = \frac{1}{2}\sum_{j\neq i} U_{ij}^{pair}(r_{ij}) + E_i\left(\sum_{j\neq i} c_j(r_{ij})\right).$$

  2. Calculates

$$\mathbf{F}_i = \mathbf{F}_i - \frac{1}{2}\sum_{j\neq i}\left(\frac{\partial E_i}{\partial \rho_i}\frac{\partial c_j(r_{ij})}{\partial \mathbf{r}_i} + \frac{\partial U_{ij}^{pair}(r_{ij})}{\partial \mathbf{r}_i}\right).$$

  3. Calculates and sums with atomic operations all $\mathbf{F}_j$ (j are the numbers of neighboring atoms of atom i, $j\neq i$):

$$\mathbf{F}_j = \mathbf{F}_j + \frac{1}{2}\left(\frac{\partial E_i}{\partial \rho_i}\frac{\partial c_j(r_{ij})}{\partial \mathbf{r}_i} + \frac{\partial U_{ij}^{pair}(r_{ij})}{\partial \mathbf{r}_i}\right).$$

The more natural way is to compute embedded-atom potential without atomic operations:

- Algorithm WA is assumed to run for N work-items in 1D index space работ (*N* GPU threads).
- Each thread *i*
  1. Calculates

$$U_i = \frac{1}{2}\sum_{j\neq i} U_{ij}^{pair}\left(r_{ij}\right) + E_i\left(\sum_{j\neq i} c_j\left(r_{ij}\right)\right).$$

  2. Calculates

$$\mathbf{F}_i = -\sum_{j\neq i}\left(\frac{\partial E_j}{\partial \rho_i}\frac{\partial c_i}{\partial \mathbf{r}_i} + \frac{\partial E_i}{\partial \rho_j}\frac{\partial c_j}{\partial \mathbf{r}_i} + \frac{1}{2}\frac{\partial U_{ij}^{pair}\left(r_{ij}\right)}{\partial \mathbf{r}_i}\right).$$

Results

The performance of the described algorithms for Tersoff and embedded-atom potentials is evaluated via the following computer systems:

- CPU: Intel Core i5 760, GPU: GeForce GTX 470, Windows 7, Visual Studio 2008, *.
- CPU: Intel Xeon X5650, GPU: Tesla M2050, Linux, **.

For the following testing configurations AMD APP SDK 2.8 was used as a realization of OpenCL CPU platform and CUDA Toolkit (CUDA 4.2.1) as a realization of OpenCL GPU platform. All GPU computations and comparisons were done on one node of computer system mostly with single precision arithmetic.

The main point of the comparison is to find the impact of the atomic operations with memory on the performance of calculation of the forces due to many-body interaction between the atoms. The results of benchmarks are summarized in Table 1. The corresponding optimal algorithm is highlighted with italic (lower is better).

Table 1. Performance of the OpenCL algorithms for many-body interaction

Execution time ratio \ Number of atoms	**1000**	**2000**	**4000**	**8000**	**16000**
Tersoff **CPU**, Algorithm WA / *Algorithm A**	33.43	33.49	33.54	32.8	33.68
Tersoff **GPU**, Algorithm WA / *Algorithm A**	9.67	9.62	10.42	10.68	11.94
Tersoff **GPU**, Algorithm WA / *Algorithm A***	48.89	54.25	55.37	60.29	70.55
EAM **CPU**, *Algorithm WA* / Algorithm A*	0.66	0.86	0.76	0.69	0.49
EAM **GPU**, *Algorithm WA* / Algorithm A*	0.51	0.48	0.47	0.48	0.49
EAM **GPU**, *Algorithm WA* / Algorithm A**	0.63	0.41	0.09	0.23	0.20

The next point is a speedup comparison. The following variants of energy and force computation algorithms are considered:

- Serial CPU algorithm;
- OpenCL algorithm with atomic operations;
- OpenCL algorithm without atomic operations.

The speedup of OpenCL algorithms is estimated by comparison of their execution time with the corresponding execution time of the serial algorithms (Fig. 1 and 2).

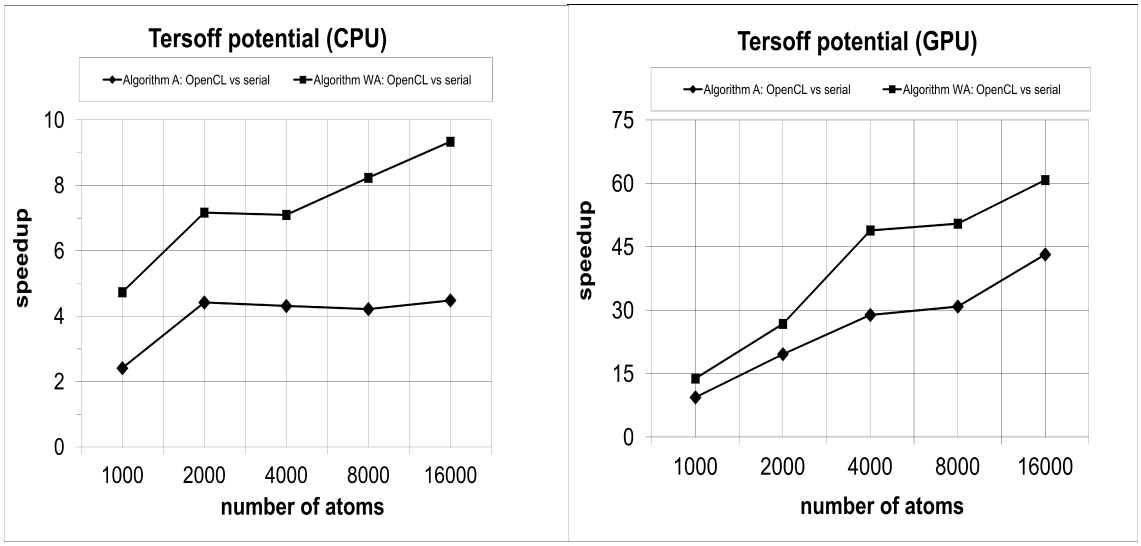

Fig. 1. Comparison of speedup of Tersoff potential (evaluated on the testing platform *)

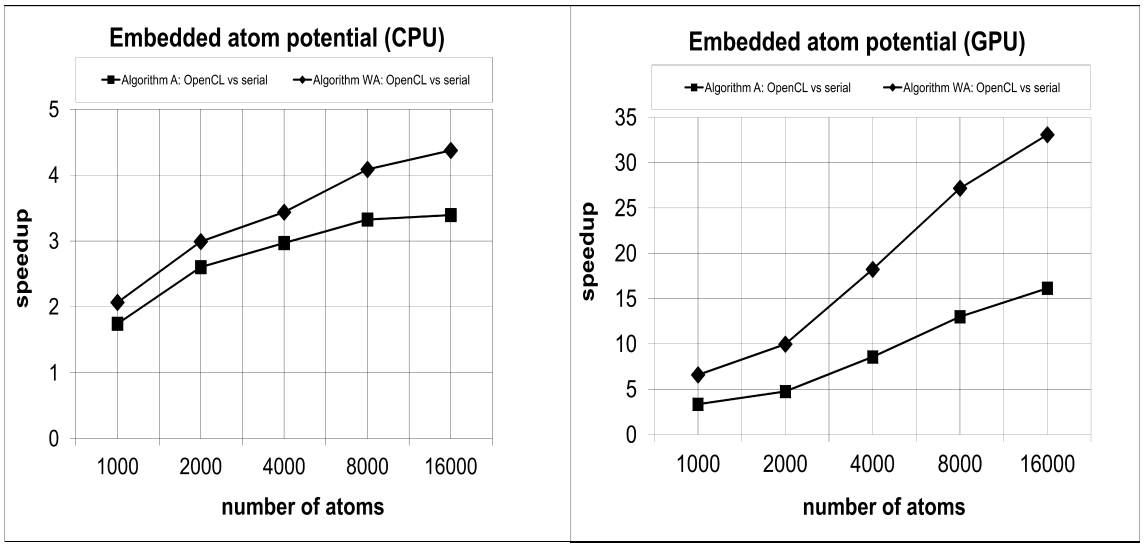

Fig. 2. Comparison of speedup of EAM potential (evaluated on the testing platform *)

One can see that Algorithm WA for the Tersoff potential provides greater acceleration than Algorithm A, both for the CPU and GPU. That result is mostly associated with a large number of arithmetic operations in the Algorithm WA. However, the higher value of the acceleration relative to the serial implementation does not mean the optimality of the algorithm itself according to the Table. 1. Algorithm A for Tersoff potential works 10 times faster on average in spite of atomic operations with memory.

Embedded-atom potential contains less arithmetic operations compared to Tersoff potential. The algorithms with and without atomic operations have approximately the same computational complexity. In that case, the effect of atomic operations with memory results in decrease of the performance. Thus, embedded-atom potential can be effectively implemented without atomic operations and the optimal algorithm is similar to that for pair potentials. So the result for embedded-atom potential is completely opposite to Tersoff potential (Table 1).

The last point is to note the influence of double precision arithmetic on the performance of GPU computations. The corresponding comparison for Tersoff potential (evaluated on the testing platform**) can be seen on Fig. 3. The speedup of algorithm depends on the floating point precision used

for performance evaluation and the hardware. For Algorithm A large values of speedup can be obtained for single precision arithmetic. For double precision the speedup is not as large and is similar to the Algorithm WA. For Algorithm WA the difference in speedup between single and double precision is not so clearly pronounced.

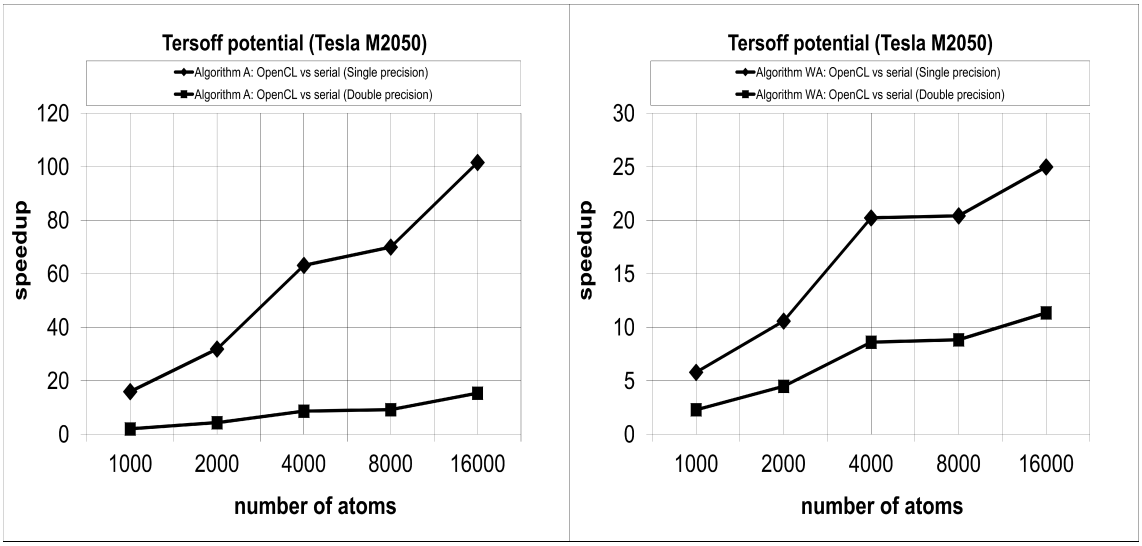

Fig. 3. The influence of floating point precision on the performance of Tersoff potential

Conclusions

Adequate modeling of nanostructures using classical methods requires accounting effects of the many-body interaction which leads to the increase of the computational complexity of the algorithms. Using GPU is the effective way of their acceleration. The performance of algorithms is greatly affected by the memory access patterns, floating-point arithmetic and some hardware features. Atomic operations are one of such patterns that are good or bad depending on the algorithm. As for the interatomic potentials one should always take into account their types and the architecture of the computing system for implementation. Some general notes can be seen as a conclusion.

Tersoff potential requires a significant amount of computation compared to pair potentials and embedded-atom potential. Using atomic operations is the optimal approach to the implementation of Tersoff potential. That gives significant reduction of the computational complexity. So the theoretical analysis and performance evaluation show that using atomic operations with memory does not always lead to poor performance. That is the case for Tersoff potential and can be assumed for other bond order potentials.

Embedded-atom potential gives the opposite result. The best way to compute EAM does not mean using atomic operations and critical sections. In that work such implementation was given just to show that difference.

Performance evaluation shows a significant acceleration of the GPU algorithms for many-body potentials. The average performance of the OpenCL algorithms is about 50 times compared to serial implementations. Using GPU is a good way for accelerating algorithms for interatomic interaction and particularly for the many-body interactions. OpenCL technology is universal tool to run the same algorithm on different hardware architectures. GPU algorithms for interatomic potentials can be used as building block of general molecular dynamics implementation for supercomputer systems.

References

Allen M. P., Tildesley D. J. Computer simulation of liquids. Oxford University Press, New York, 1990.

Anderson J. A., Lorenz C. D., Travesset A. General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units // J. Comp. Phys. — 2010. — 227(10). — P. 5342–5359.

Bielajew A. F. Fundamentals of the Monte Carlo method for neutral and charged particle transport. 2001. [online]: http://www-personal.umich.edu/~bielajew/MCBook/book.pdf

Brenner D. W. Relationship between the embedded-atom method and Tersoff potentials // Phys. Rev. Lett. — 1989. — 63(9). — P. 1022–1022.

Erkoc S. Empirical Many-Body Potential Energy Function Used In Computer Simulations Of Condensed Matter Properties // Physics Reports. — 1997. — 278(2). — P. 79–105.

Gaster B., Howes L., Kaeli D. R., Mistry P., Schaa D. Heterogeneous Computing with OpenCL // Morgan Kaufmann. — 2011. — 296 p.

Jason Sanders, Edward Kandrot. CUDA by Example: An Introduction to General-Purpose GPU Programming // Addison-Wesley Professional. — 2010.

Kumagai T., Hara S., Choi J., Izumi S., Kato T. Development of empirical bond-order-type interatomic potential for amorphous carbon structures // Journal of Applied Physics. — 2009. — 105(6). — P. 064310

Lebedeva I. V., Knizhnik A. A., Popov A. M. Potapkin B. V. Ni-Assisted Transformation of Graphene Flakes to Fullerenes // J. Phys. Chem. *C*. — 2012. — 116(11). — P. 6572–6584.

Murray S. Daw, Foiles S. The embedded-atom method: a review of theory and applications // Mat. Sci. Reports. — 1993. — 9.

Tersoff J. Modeling solid-state chemistry: Interatomic potentials for multicomponent systems // Phys. Rev. B. — 1989. — 39(8). — P. 5566–5568.