SurfaceGrowth: CUDA-based code for molecular dynamics of metal atoms adsorbed on a graphene sheet

Nikolay V. Prodanov[a,b,*]

[a] *Department of Complex Systems Modeling, Sumy State University, 2 Rimskii-Korsakov Str., 40007 Sumy, Ukraine*
[b] *Institut für Festkorperforschung, Forschungszentrum Jülich, D-52425 Jülich, Germany*

Abstract

This paper describes a program SurfaceGrowth, which is designed for classical molecular dynamics simulations of the evolution of a system consisting of metal atoms interacting with a graphene layer. By choosing one of several regimes for one of six metals with face-centered cubic lattice, a user has the opportunity to investigate the diffusion and friction of a metal nanoparticle adsorbed on a graphene sheet, or to study the deposition of metal atoms on graphene. The code harnesses powerful computational capabilities of graphics processing units, which allows simulating relatively large systems with realistic interactions on a desktop computer over reasonable times.

Keywords: Graphics processing unit, Nanoparticle, Graphene, Atomic scale friction, Diffusion, Deposition
PACS: 66.30.Pa, 68.35.Af, 68.43.Jk, 81.15.Aa

PROGRAM SUMMARY

Manuscript Title: SurfaceGrowth: CUDA-based code for molecular dynamics of metal atoms adsorbed on a graphene sheet
Authors: N.V. Prodanov
Program Title: SurfaceGrowth
Journal Reference:
Catalogue identifier:
Licensing provisions: none
Programming language: CUDA C
Computer: Personal computer with CUDA enabled graphics processing unit (GPU)
Operating system: Windows XP or higher
Supplementary material: User guide with installation notes
Keywords: Molecular dynamics, EAM potential, Neighbor list
Classification: 6.5 Software including Parallel Algorithms, 7.7 Other Condensed Matter inc. Simulation of Liquids and Solids
External routines/libraries:
Subprograms used:
Nature of problem: The code allows simulating the behavior of metal atoms interacting with a graphene sheet using classical molecular dynamics. In particular, there is the possibility to study the formation of metal nanoparticles by dewetting technique, and to investigate deposition of metal atoms on a graphene layer under different conditions. A user can also investigate frictional phenomena occurring during the shear of a nanoparticle or to explore the diffusion of metal nanoclusters adsorbed on graphene
Solution method: Classical molecular dynamics approach based on the parallel implementation of neighbor list technique for a GPU
Restrictions: GPU must have compute capability of 1.2 or higher. Maximum system size is 1048576 atoms
Unusual features:
Additional comments:
Running time: Depends on the parameters

1. Introduction

Interactions of atoms and nanoobjects with graphene which is either free or embedded in a graphite substrate attract significant experimental and theoretical research interest in different areas of nanotechnology, such as, nanoelectronics, nano-electro-mechanical systems (NEMS), nanotribology, thin film growth and nanoparticle production. Due to the dependence of the local carrier concentration on the presence of adsorbates [1, 2], there exists the possibility to manage electronic properties of graphene using adsorbed molecules or clusters. The formation of point defects in graphene due to irradiation damage or ion bombardment also has an impact on the electronic properties of the material [3, 4]. Changing the conformation of the graphene layer may be valuable for some NEMS, and this may be achievable via adding a water nanodroplet on a graphene sheet [5, 6]. In nanotribological studies graphene layers are usually embedded in the substrate made of highly oriented pyrolytic graphite (HOPG). The latter is often used in the experiments concerned with friction and wear at the atomic scale of different systems, such as, tungsten tip of the friction force microscope (FFM) [7, 8], antimony [9, 10] and gold [11, 12] nanoparticles. Also tribological properties of isolated graphene have been characterized recently using FFM [13, 14]. Exfoliation of graphene from a graphite surface [15, 16, 17] and tearing of an atomically thin carbon layer [18] – the techniques which are used for isolation of graphene and production of graphene nanoribbons, are achieved through the interaction with an adhesive surface. Comprehending the behavior of metal atoms agglomerated in clusters or nanoislands and interacting with a graphite surface is important for the controlling the deposition techniques which are used for thin film growth [19, 20] or production of nanoparticles [10, 11]. Experimentally, it was observed that metal nanoclusters formed after deposition on a graphite surface are rather mobile [1, 19, 20], and this was confirmed by molecular dynamics simulations of the diffusion of golden nanoparticles on the graphite surface [21, 22, 23, 24, 25, 26]. Understanding and controlling the diffusion of metal nanoclusters may be valuable for the "bottom-up" preparation techniques of nanostructured materials or future nanodevices [20].

A plethora of experimental data pertained to friction and diffusion of metal nanoparticles adsorbed on HOPG surface has been obtained during last years, and is being collected now at a fast pace. But relatively small amount of the adequate theoretical treatment of the mentioned processes can be found in literature. One of the main reasons for this is rather complicated and multiple-factor nature of the problem which imposes restrictions both on analytical and numerical approaches. On the one hand, analytical or seminumerical models [27, 28] can provide some estimates of the experimentally observed quantities. However, they are often based on a large amount of assumptions, and thus may not be able to yield a clear picture due to neglecting some of the contributions which can play an appreciable role. For example, very often it may not be reliable to neglect the atomistic nature of nanoparticles and the substrate. On the other hand, numerical simulations, in particular, using classical molecular dynamics (MD) are very computationally costly as metal nanoparticles in the nanotribological experiments have relatively large sizes ranging from tens [11, 12] to hundreds [9, 10] nm in diameter. Simulations of the diffusion of nanoclusters are also very computationally heavy, as they should be performed over relatively long simulation times of at list tens of nanoseconds. Thus, in the mentioned works [21, 22, 23, 24, 25, 26] the diffusion of only small nanoparticles consisting of up to about 600 atoms was studied.

Therefore, there exists a clear demand for the speedup of the simulations of metal nanoparticles interacting with a graphite surface, which would give the opportunity to bring the model size and simulation times closer to the experimental conditions. Parallel programming should be involved to achieve this aim, and standard message-passing approach can be employed [29, 30]. However, fast development of NVIDIA CUDA platform [31] and the appropriate MD algorithms for graphics processing units (GPUs) [32, 33, 34, 35] tempt to accelerate MD code using GPU. Such an approach looks quite reasonable as a GPU can provide the computational power comparable with the capabilities of a small central processing unit (CPU) cluster [33, 35]. Hence, with the appropriate design of the code, a user can perform intensive computations on a desktop computer without being bothered with purchase and maintenance of the corresponding CPU cluster hardware and software.

In this paper, the program SurfaceGrowth is described. It is designated for MD simulations of metal atoms adsorbed on a graphene layer, and the code is fully implemented for a GPU using NVIDIA CUDA platform. The main background for the design of SurfaceGrowth was the intention to explain the experiments concerned with friction of metal nanoparticles on HOPG surface [9, 10, 11]. However, as is inherent for the simulations, there are some discrepancies between the model and the experimental setup. The main one is the use of a graphene sheet instead of a graphite substrate. Generally, properties of isolated graphene differ from the ones of a graphene layer embedded in the graphite substrate. Nevertheless, the features essential for tribological processes may not differ significantly, and one layer may serve as a good starting point toward more realistic modeling. In addition, tribological properties of graphene are being studied intensively [13, 14], and experimental manipulation of nanoparticles adsorbed on this material can be implemented in future, thus enabling direct comparison with the Surface-

Growth model. Note that in spite of the mentioned main background, the code turned out to be more universal. It is possible to investigate another surface processes, such as, diffusion or deposition of metal atoms on graphene. Due to the GPU speedup, the program can be used for simulations of rather large systems during long simulation times. For example, SurfaceGrowth was used to obtain the results of Ref. [36], where friction of relatively large Ag and Ni nanoparticles containing up to 30000 atoms interacting with a graphene layer with up to 43808 atoms was studied on a commodity GPU with compute capability 1.3. The release of new GPUs, in particular, based on the Fermi architecture enables to simulate larger systems without any modification of the code.

The paper is organized as follows. In Section 2 the model and the algorithms are presented, Section 3 describes the code, then some results are given in Section 4, and the paper is closed by concluding Section 5. The guide for the use of the graphical user interface (GUI) of the program can be found in the manual accompanying the code.

2. Model

To have some impression about the application, general description of the system is given at first by introducing the regimes of work of SurfaceGrowth. After that the essential parts of the model, such as, interatomic potentials, temperature control etc. are presented, and then parallel algorithms employed in the code are reported.

2.1. Regimes

In SurfaceGrowth a simulation can be carried out in one of three regimes: `Bulk`, `Surface Growth`, `Shear`. They differ from one another by the composition of atoms, their initial configuration, course of the computation and the possibilities of measurements. There are general and some specific input parameters inherent to each regime which should be specified by the user (this question is described in the user guide). Some output quantities, such as, total and potential energies of the system can be measured in all regimes, and they are called *general parameters* (details are given in section 4.1). Another output quantities, such as, characteristics of the nanoparticle, can be measured only in the specific regime. Let us consider each regime.

Regime `Bulk` is intended for the simulation of bulk properties of a metal. Only metal atoms are used (without graphene), they are arranged in the ideal face-centered cubic (fcc) lattice, and periodic boundaries are applied in three directions (x, y, z). The microcanonical conditions with constant number of particles N, system volume V and total energy E are maintained. The main aim of the `Bulk` regime is to verify the general MD algorithm and the empirical potential used for the metal by analyzing the binding energy and the atomic structure. In the conditions of this regime, total energy E should be conserved, and the total momentum should have the value near 0 in the course of the run. There can be some small deviations from the mentioned behavior due to rounding errors and the single precision arithmetic used in the code for GPU. In this and two other regimes there is the possibility to study one of the following six metals with fcc lattice: copper (Cu), silver (Ag), gold (Au), nickel (Ni), aluminium (Al) and lead (Pb).

Deposition of atoms of the chosen metal for different deposition energies, temperatures, flux densities and parameters of metal–C interaction can be studied in the regime `Surface Growth`. In this regime only graphene is present in the simulation cell at the beginning of a simulation. The graphene sheet lies in the xy plane with zigzag and armchair edges parallel to x and y directions, respectively (Figure 1). Periodic boundary conditions are applied in all directions. To hold the sample in space, boundary carbon atoms along the perimeter of the graphene layer are held fixed throughout the simulations. Metal atoms are located outside the simulation cell and do not interact with each other. After the equilibration period, metal atoms are injected in the simulation box above the graphene layer by groups in a regular time interval. The size of a group of metal atoms and the delay between injection of differen groups are specified by the user, thus enabling to control the flux density of the deposited atoms. Injected atoms have random initial values of coordinates in the xy plane, and z coordinate is defined by the size of the simulation box. Lateral components of the initial velocities of the injected atoms are equal to 0, and z component is directed to the graphene layer (opposite to z direction), its value is defined as follows:

$$v_{0z} = \sqrt{\frac{2\varepsilon_{\mathrm{dep}}}{m}}, \tag{1}$$

where $\varepsilon_{\mathrm{dep}}$ is the energy of the deposited atoms defined by the user, m is the metal atomic mass. It is possible to measure general parameters in the regime `Surface Growth`. In this regime the Berendsen thermostat [37] is used to dissipate the heat generated during collisions of deposited atoms with graphene, and the thermostat is applied only to carbon atoms. Very intensive bombardment of the layer will lead to its overheating and ultimately to its destruction. Therefore, this regime requires careful adjustment of the parameters in order to avoid the damage of the layer.

The third and the last regime is called `Shear`, and it allows investigating friction or diffusion of a metal nanoparticle adsorbed on graphene. During the simulation, graphene layer is treated in the same way as in the regime `Surface Growth`. A nanoparticle is obtained by the procedure imitating the dewetting of thin metallic films by thermal treatments [39], and in more details it is as follows. At the beginning of a simulation a slab consisting of several layers of metal atoms packed in the ideal fcc lattice is placed above the graphene layer (cf. Figure 2). Vertical distance between the graphene plane and the lowest metal layer depends on the metal type. Metal atoms have zero initial velocities. During the equilibration period which is specified by the user, the system evolves without dissipation of the heat, i. e. the thermostat is not coupled neither to carbon nor to metal atoms. Since many atoms in the slab are on its surface and have coordination which is smaller than the one in the bulk state, such ideal fcc lattice is not energetically profitable, and metal atoms begin to rearrange into the more compact conformation corresponding to the minimal free energy. This process is accompanied by the release of the energy. Therefore, the temperature T of the system is being raised, and the nanoisland melts. The final configuration of metal atoms depends on the values of the metal–C interaction parameters specified by the user. If this interaction energy is much smaller than the metal–metal one, the contact angle of the forming metal cluster should approach 180° [40]. The configuration with minimal energy will correspond to a ball, which may not be suitable for the problem as it has small contact area. So, in order to obtain nanoparticles with the desired semispherical shape, at the appropriate time moment defined empirically Berendsen thermostat is applied both to metal and graphene during some time interval (specified by the user) to cool the system down to the specified temperature. After that the thermostat is decoupled from the metal atoms and is applied only to graphene to dissipate the heat generated during the motion of the nanoparticle. Note that equilibration in this regime does not mean reaching the equilibrium state of the system. The term "equilibration" is used just for consistency with other regimes. In `Shear` regime it denotes the period when the thermostat is not applied to the system. To imitate pushing of the formed nanoparticle by the tip of an atomic force microscope (AFM), immediately after the cooling interval shear force is applied along the zigzag edge of graphene (which coincides with the x direction) to all metal atoms with values of x coordinates that are smaller than the x coordinate of the center of mass (CM) X_{CM} of the nanoparticle. At first, the force is incremented in steps ΔF specified by the user until the x component of the velocity of CM V_{X} reaches the value of 3.55 m/s. Then the shear force acting on each atom remains constant, and the simulations are held with constant total shear force F_{S}. If the value of ΔF is zero, then the nanoparticle is not subjected to shear, and there is the possibility to study its diffusion.

In `Shear` regime the general parameters are measured as well as different characteristics of a nanoparticle: the position and the velocity of the center of mass, shear and friction forces, sizes and the structure accordingly to radial distribution function (RDF). It is important to properly choose the equilibration period in this regime, as it defines the shape and the structure of the nanoisland, and hence the contact area. Too small equilibration time leads to the nanoisland with rectangular-like shape, and too long equilibration will lead to the evaporation of the nanoisland. This time also depends on the type of metal, as for heavier metals, e. g. lead, atoms rearrange much slower than for lighter ones, such as, aluminium.

2.2. Simulation Setup

After the general description of the system layout in different regimes, let us consider some details of the model.

Forces between metal atoms are derived from the alloy form of the embedded atom method (EAM) potential [41, 42]. This potential is expressed in terms of analytical functions without the use of splines in contrast to older versions of the EAM potential [29, 43]. In EAM, the potential energy V_{eam} of the crystal is written as

$$V_{\mathrm{eam}} = \frac{1}{2} \sum_{i,j,i \neq j} \phi_{ij}(r_{ij}) + \sum_{i} F_i(\rho_i), \tag{2}$$

where ϕ_{ij} represents the pair energy between atoms i and j separated by the distance r_{ij}, and F_i corresponds to the embedding energy to embed an atom i into a local site with electron density ρ_i. ρ_i is calculated using:

$$\rho_i = \sum_{j,j \neq i} f_j(r_{ij}), \tag{3}$$

with $f_j(r_{ij})$ the electron density at the site of atom i arising from atom j at a distance r_{ij} away.

The generalized pair potential is defined as follows:

$$\phi(r) = \frac{A \cdot \exp\left[-\alpha\left(\frac{r}{r_\mathrm{e}}-1\right)\right]}{1+\left(\frac{r}{r_\mathrm{e}}-\kappa\right)^{20}} - \frac{B \cdot \exp\left[-\beta\left(\frac{r}{r_\mathrm{e}}-1\right)\right]}{1+\left(\frac{r}{r_\mathrm{e}}-\lambda\right)^{20}}, \tag{4}$$

where r_e is the nearest-neighbor distance at the equilibrium, A, B, α, β are adjustable parameters, and κ, λ are two additional parameters for the cut off.

The electron density function $f(r)$ has the same form as the attractive term in the pair potential with the same values of β and λ:

$$f(r) = \frac{f_\mathrm{e}\exp\left[-\beta\left(\frac{r}{r_\mathrm{e}}-1\right)\right]}{1+\left(\frac{r}{r_\mathrm{e}}-\lambda\right)^{20}}. \tag{5}$$

To have embedding energy functions that can work well over a wide range of electron density, three equations are used to separately fit to different electron density ranges, $\rho < \rho_\mathrm{n}$, $\rho_\mathrm{n} \leq \rho < \rho_\mathrm{o}$ and $\rho_\mathrm{o} \leq \rho$. Values of $\rho_\mathrm{n} = 0.85\rho_\mathrm{e}$ and $\rho_\mathrm{o} = 1.15\rho_\mathrm{e}$, where ρ_e is the equilibrium electron density, are used to ensure that all equilibrium properties can be fitted in the electron density range $\rho_\mathrm{n} \leq \rho < \rho_\mathrm{o}$. These equations provide matching of the values and slopes of the embedding energy at the junctions of the mentioned intervals. The equations have the following form:

$$F(\rho) = \sum_{i=0}^{3} F_{\mathrm{n}i}\left(\frac{\rho}{\rho_\mathrm{n}}-1\right)^i, \rho < \rho_\mathrm{n}, \rho_\mathrm{n} = 0.85\rho_\mathrm{e}, \tag{6}$$

$$F(\rho) = \sum_{i=0}^{3} F_i\left(\frac{\rho}{\rho_\mathrm{e}}-1\right)^i, \rho_\mathrm{n} \leq \rho < \rho_\mathrm{o}, \rho_\mathrm{o} = 1.15\rho_\mathrm{e}, \tag{7}$$

$$F(\rho) = F_\mathrm{e}\left[1-\ln\left(\frac{\rho}{\rho_\mathrm{e}}\right)^\eta\right]\left(\frac{\rho}{\rho_\mathrm{e}}\right)^\eta, \rho_\mathrm{o} \leq \rho. \tag{8}$$

Values of the parameters involved in the EAM potential for 15 and 16 fcc metals can be found in Refs. [41] and [42], respectively, or parameters for six metals listed in section 2.1 can be found in the code of SurfaceGrowth. Values of masses and densities of metals are taken from Ref. [44].

The force acting on metal atom k from all other metal atoms is given by [30]

$$\mathbf{f}_k = -\frac{\partial V_{\mathrm{eam}}}{\partial \mathbf{r}_k} = -\sum_{j\neq k} \frac{\mathrm{d}\phi(r_{kj})}{\mathrm{d}r}\hat{\mathbf{r}}_{kj} - \sum_{i=1}^{N_{\mathrm{m}}} \frac{\partial F_i}{\partial \rho_i}\frac{\partial \rho_i}{\partial \mathbf{r}_k}, \tag{9}$$

where N_{m} is the total number of metal atoms, $\hat{\mathbf{r}}_{kj}$ is a unit vector directed from atom j to atom k. Expressions for the derivatives in (9) are listed as:

$$\begin{aligned}\frac{\mathrm{d}\phi}{\mathrm{d}r} = &-\frac{A\cdot\exp\left[-\alpha\left(\frac{r}{r_{\mathrm{e}}}-1\right)\right]}{1+\left(\frac{r}{r_{\mathrm{e}}}-\kappa\right)^{20}}\left[\alpha+\frac{20\left(\frac{r}{r_{\mathrm{e}}}-\kappa\right)^{19}}{1+\left(\frac{r}{r_{\mathrm{e}}}-\kappa\right)^{20}}\right]\frac{1}{r_{\mathrm{e}}}+\\ &\frac{B\cdot\exp\left[-\beta\left(\frac{r}{r_{\mathrm{e}}}-1\right)\right]}{1+\left(\frac{r}{r_{\mathrm{e}}}-\lambda\right)^{20}}\left[\beta+\frac{20\left(\frac{r}{r_{\mathrm{e}}}-\lambda\right)^{19}}{1+\left(\frac{r}{r_{\mathrm{e}}}-\lambda\right)^{20}}\right]\frac{1}{r_{\mathrm{e}}},\end{aligned} \tag{10}$$

$$\sum_{i=1}^{N_{\mathrm{m}}} \frac{\partial F_i}{\partial \rho_i}\frac{\partial \rho_i}{\partial \mathbf{r}_k} = \sum_{i=1,i\neq k}^{N_{\mathrm{m}}} \frac{\mathrm{d}f(r_{ki})}{\mathrm{d}r}\left(\frac{\partial F_i}{\partial \rho_i}+\frac{\partial F_k}{\partial \rho_k}\right)\hat{\mathbf{r}}_{ki}, \tag{11}$$

$$\frac{\mathrm{d}f}{\mathrm{d}r} = -\frac{f_{\mathrm{e}}\exp\left[-\beta\left(\frac{r}{r_{\mathrm{e}}}-1\right)\right]}{1+\left(\frac{r}{r_{\mathrm{e}}}-\lambda\right)^{20}}\left[\beta+\frac{20\left(\frac{r}{r_{\mathrm{e}}}-\lambda\right)^{19}}{1+\left(\frac{r}{r_{\mathrm{e}}}-\lambda\right)^{20}}\right]\frac{1}{r_{\mathrm{e}}}, \tag{12}$$

$$\frac{\partial F}{\partial \rho} = \frac{1}{\rho_{\mathrm{n}}}\left[F_{\mathrm{n}1}+2F_{\mathrm{n}2}\left(\frac{\rho}{\rho_{\mathrm{n}}}-1\right)+3F_{\mathrm{n}3}\left(\frac{\rho}{\rho_{\mathrm{n}}}-1\right)^2\right], \rho<\rho_{\mathrm{n}}, \rho_{\mathrm{n}}=0.85\rho_{\mathrm{e}}, \tag{13}$$

$$\frac{\partial F}{\partial \rho} = \frac{1}{\rho_{\mathrm{e}}}\left[F_1+2F_2\left(\frac{\rho}{\rho_{\mathrm{e}}}-1\right)+3F_3\left(\frac{\rho}{\rho_{\mathrm{e}}}-1\right)^2\right], \rho_{\mathrm{n}}\leq\rho<\rho_{\mathrm{o}}, \rho_{\mathrm{o}}=1.15\rho_{\mathrm{e}}, \tag{14}$$

$$\frac{\partial F}{\partial \rho} = -\frac{F_{\mathrm{e}}\eta}{\rho_{\mathrm{e}}}\left(\frac{\rho}{\rho_{\mathrm{e}}}\right)^{\eta-1}\ln\left(\frac{\rho}{\rho_{\mathrm{e}}}\right)^{\eta}, \rho_{\mathrm{o}}\leq\rho. \tag{15}$$

Interactions between carbon atoms in graphene are described by the harmonic potential [45]. The simulations are not concerned with the formation or breaking of chemical bonds in the graphene layer, so this potential form may be more appropriate for our task compared with more sophisticated potentials such as Brenner [29, 46, 47] or ReaxFF [18, 48] as it is less time-consuming. Potential energy V_C of interactions between carbon atoms in graphene is defined as follows [45]:

$$V_C = \frac{1}{2}\sum_{i-j}\mu_r(r_{ij}-r_0)^2 + \frac{1}{2}\sum_{i-j-k}\mu_\theta r_0^2(\theta_{ijk}-\theta_0)^2 + \frac{1}{2}\sum_{i-(j,k,l)}\mu_p\left(\delta z_i - \frac{\delta z_j+\delta z_k+\delta z_l}{3}\right)^2. \tag{16}$$

Summation is performed over the nearest-neighbor bonds, bond pairs, and bond triples, respectively. The bond stretching, and the bond bending energies are described by the first and the second terms, respectively. r_{ij} is the nearest-neighbor bond length between the bond $i-j$. θ_{ijk} denotes the angle between the bond $i-j$ and the bond $j-k$ within graphene. The third term corresponds to the bending energy of the local planar structure due to the normal displacement of the ith atom from the coplanar position with respect to the three neighboring atoms j, k, l, and δz_i denotes the normal displacement of the ith atom from the initial position. The parameters of V_C are $r_0 = 1.4210$ Å, $\mu_r = 41.881$ eV/Å^2, $\theta_0 = 2\pi/3$, $\mu_\theta = 2.9959$ eV/Å^2, and $\mu_p = 18.225$ eV/Å^2.

The expression for the force acting on the ith carbon atom from other carbons reads:

$$\mathbf{f}_i = -\sum_{i-j}\mu_r(r_{ij}-r_0)\frac{\mathbf{r}_{ij}}{r_{ij}} + \sum_{j\neq i,k\neq i}\mu_\theta r_0^2(\theta_{jik}-\theta_0)\left[1-\left(\frac{\mathbf{r}_{ji}\mathbf{r}_{ki}}{r_{ji}r_{ki}}\right)^2\right]^{-\frac{1}{2}} \times \frac{\left(1-\frac{r_{ji}}{r_{ki}}\cos\theta_{jik}\right)\mathbf{r}_{ik} + \left(1-\frac{r_{ki}}{r_{ji}}\cos\theta_{jik}\right)\mathbf{r}_{ij}}{r_{ji}r_{ki}} + \sum_{j\neq i,k\neq i}\mu_\theta r_0^2(\theta_{ijk}-\theta_0)\left[1-\left(\frac{\mathbf{r}_{ij}\mathbf{r}_{kj}}{r_{ij}r_{kj}}\right)^2\right]^{-\frac{1}{2}} \times \left[\frac{\mathbf{r}_{kj}}{r_{ij}r_{kj}} - \frac{(\mathbf{r}_{ij}\mathbf{r}_{kj})}{r_{ij}^3 r_{kj}}\mathbf{r}_{ij}\right] -$$

$$\sum_{j,k,l} \frac{2}{3}\mu_p \left[2\delta z_i - (\delta z_j + \delta z_k + \delta z_l)\right] - \frac{1}{9}\mu_p \sum_{m,n} (\delta z_m + \delta z_n)\,. \tag{17}$$

Indexes m, n in the last sum denote the next-nearest neighbors of the atom i.

Metal–carbon interaction is described using the Lennard-Jones (LJ) potential:

$$V_{LJ} = \begin{cases} 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right], & r < r_{\mathrm{c}} \\ 0, & r \geq r_{\mathrm{c}} \end{cases}, \tag{18}$$

where the default values of the parameters ε and σ are chosen as in Ref. [45], but there is also the possibility to specify the values using GUI of the program. The cutoff distance of LJ potential is $r_c = 2.5\sigma$. For the EAM potential, the value of the cutoff distance is $r_c = 1.45a$, where a is the lattice constant of the chosen metal.

The following units of measurements are used in SurfaceGrowth: carbon atomic mass $m_0 = 19.9441 \cdot 10^{-27}$ kg, the length of the covalent bond in graphene $a_0 = 1.42$ Å, time unit $t_0 = 0.2$ ps, energy unit $\varepsilon_0 = m_0 a^2/t_0^2 = 6.275049 \cdot 10^{-2}$ eV, 1 K dimensionless is 0.001373, and 1 nN dimensionless is equal to 14.123984.

Berendsen thermostat [37] is used for the temperature control. It was chosen because of the simplicity of the implementation, and due to the ability to control the temperature of different parts of the system. The basis for the implementation of this thermostat is rescaling of the velocity v of the appropriate atoms, i. e. $v_i^{n+1} = \beta v_i^n$, where n is the current number of time steps. The factor β is not constant but depends on the current T_n and the desired T_0 temperatures of the system. It is defined as follows:

$$\beta = \sqrt{1 + \gamma\left(\frac{T_0}{T_n} - 1\right)}, \tag{19}$$

where $\gamma \in [0; 1]$. If T_n is higher than T_0, then $\beta < 1$ and v_i^{n+1} diminishes. If $T_n < T_0$, then $\beta > 1$ and v_i^{n+1} is increased. The quantity γ characterizes the speed of the heat dissipation. The closer γ to 1, the faster the dissipation is, and, hence, the stronger the coupling to the heat bath is. In SurfaceGrowth $\gamma = 0.005$ which corresponds to relatively weak coupling to the thermostat.

Rescaling of atomic velocities may influence the energy distribution in the system. To reduce this effect, the implementation of the Berendsen thermostat described in Ref. [29] is employed in the code, where the velocities are rescaled not every time step, but every k steps, and in the intermediate steps the system is integrated without scaling. In SurfaceGrowth $k = 25$ which allows to restore the "equilibrium" between potential and kinetic energies of the system [29].

Equations of motion are integrated using leapfrog method described in Ref. [30]. The value of the time step can be specified through the GUI in the dimensionless form.

2.3. GPU implementation

2.3.1. An overview

Algorithms intended for GPU and implemented in SurfaceGrowth are given in this section. The principles of classical MD are not described here, this information can be found in literature [29, 30, 49, 50]. The main attention is focused on the implementation of the algorithms of classical MD using NVIDIA CUDA platform for GPUs. This requires the knowledge of some terms and the GPU hardware architecture. Corresponding overviews can be found in Refs. [31, 33, 34, 35, 51, 52]. In particular, two terms are usually used when there is the need to distinguish the resources belonging to CPU and GPU. Henceforth in this article, the term *host* is used for CPU and the corresponding memory address space, and *device* means GPU and its memory address space.

In works [33, 34, 35] various GPU algorithms for classical MD of simple LJ liquids are proposed. The speedup ranging from ten to several tens of times compared to one CPU were obtained using the GPU NVIDIA GeForce 8800 GTX (CPUs of approximately the same processing power were used in the three works). However, some peculiarities of the mentioned algorithms are worth elucidating in order to assess and choose the most appropriate one. In Ref. [33] cell lists are used as the basis for computations, and the algorithm is completely implemented on the device without intermediate copying of data to the host. Binning the particles, i. e. placing them to the cells is achieved through the use of the double buffer technique. The authors report the computation time of 20 ms per time step for 100000 atoms on GeForce 8800 GTX GPU. In Ref. [34] the neighbor list (NL) technique is used instead of cell lists. In spite of the fact that all the computations are performed completely on the GPU, the worst speedup is obtained among the three cited works. Two main reasons for such a result may be the use of the all pairs method [30] for building the NL instead of cell lists, and also the nonoptimal memory utilization as the approach in work [34] is based upon noncoalesced memory reads. Algorithms in Ref. [35] are also based on the NL technique. But in contrast to Ref. [34], cell lists are employed for building the neighbor list, and subtleties of the work with GPU memory are taken into account. Thus, the authors in work [35] state that their algorithm is three times faster than the one in [33] giving less then 10 ms per time step for 100000 atoms on GeForce 8800 GTX GPU. However, the latter work has a considerable deficiency that not the whole MD time step is computed on GPU, but binning the particles occurs on the host. This requires copying all the coordinates from device to host which may impact the efficiency of computations for large systems. The authors explain such an approach by the lack of time for the implementation of double buffer technique, and they claim that another way would require atomic functions which are not supported in GeForce 8800 GTX.

2.3.2. Main algorithms

Taking into account the information presented above, it was decided to use algorithms from Ref. [35] as the basis in SurfaceGrowth, because they give the best performance. In particular, Algorithm 2 (A2) is used for the calculation of forces, and Algorithm 4 (A4) is employed for building the NL. These algorithms can be found in [35], and are not given here. Note that the current version of SurfaceGrowth does not use textures for random access memory, and sorting of particles is not implemented. These features are planned to be added in future versions of the code.

In contrast to [35], in SurfaceGrowth the computation of the whole time step is implemented on GPU, including binning the particles into cells. This has become possible due to the announcement of GPUs with compute capability of 1.2 and higher which support atomic operations. Algorithm 1 listed below is used for binning atoms into cells. But at first some main data structures used in the program should be considered.

Usually linked lists are used per cell to store the identities of the particles located in it [29, 30, 33, 50]. The disadvantage of such an approach is that memory access is random, not sequential. Therefore, a linked list cannot be loaded in parallel and is not appropriate for the GPU. Another way is to assign a fixed sized array of placeholders (AOP) [33] to every cell and copy particles position (or indexes of the positions array) into AOP. The advantage of this scheme is that interacting particles are physically close together in memory allowing for fast parallel loading. The disadvantage is that it generally requires more memory, because each AOP has to provide space enough to store particles at the highest possible density.

In SurfaceGrowth the latter approach is implemented accordingly to [35]. The arrays CELL and NBL serve as AOPs and play the main role in organizing the calculation of particles interactions. The structure of these arrays is defined by the algorithms for building the NL and force calculation (A4 and A2 in Ref. [35], respectively). In particular, during the NL build each cell is assigned to a separate block, and the particles of each cell are managed by the threads of the separate block. This approach allows the memory operations reading particle positions to be done on a per block basis and then accessed via shared memory.

The array CELL (see Fig. 3) holds the indexes of particles belonging to a cell, and it is used to build the NL. The layout of the CELL array takes into account that on steps 6 and 7 of the A4 all the threads in a block load in parallel all the particle indexes of the current or one of the 26 neighboring cells from global to shared memory. The indexes of the particles are stored in the rows of the array in order to provide coalesced reads. The length of rows should be multiple of the warp size, which is equal to 32 (currently for all the GPU models). Not all the cells will contain the maximum number of particles, so before the building of the cells the array CELL is filled by -1 which indicates virtual atoms. The number of the row (starting from 0) of the CELL array is associated with the index of a cell, which is denoted by the arrow in Fig. 3. Total number of cells N_{cell} depends on the cutoff distance chosen for metal atoms and on sizes of the simulation box.

The array NBL represents the NL, i. e. it holds the indexes of all the neighbors of each atom. The layout of NBL array (see Fig. 4) is designated for the force calculation algorithm. In this case each thread is responsible for its own atom and deals with the unique number of neighbors. Calculations are not based on a per block basis, so a column of the NBL array is associated with single thread. Indexes of neighbors of an atom are stored in columns and then are read sequentially by the corresponding thread during the course of the calculations (this is denoted by arrows along the columns in Fig. 4). The number of rows in NBL array in this case corresponds to the maximum number of neighbors NN_{max}. Not all the atoms have the maximum number of neighbors, that is why before the building the NL, the array NBL is filled by -1 indicating virtual atoms.

A neighboring atom is mainly defined using the cutoff distance of the EAM potential specified in Section 2.2. Also the additional skin of about 0.568 Å is added to the cutoff to take into account that the NL is updated not every time step. The value of the skin is specified in the code (variable `g_hrNebrShell`), and the program should be recompiled in order to change it. The criterion for regenerating the NL is based on the observation for the maximum velocity of the atoms. When the following condition

$$\sum_{\text{steps}} (\max_i |\mathbf{v}_i|) > \frac{\Delta r}{2\Delta t}, \tag{20}$$

is true, the NL is updated. Here, $\mathbf{v}_i$ is the velocity of an atom, Δr is the skin value added to the cutoff radius, Δt is the time step.

Binning atoms into cells, that is, building the array CELL proceeds accordingly to the specially developed Algorithm 1. The simulation box is subdivided into rectangular cells with the size defined by the cutoff distance of the metal and the skin Δr (see Fig. 5). Each block is assigned to a cell, and the block index bid corresponds to the index of its cell. On the line 1 number of reads K for a block is defined. Then on line 4 an index n of the atomic position which will be read by the current thread (defined by index tid) is found, and the thread reads the atomic coordinates from global memory on the line 5 (see also Fig. 6). Then every thread defines the index of a cell c to which its atom n belongs (line 6). If c coincides with the index of the current block (i. e. bid), then the free position in the CELL array is defined using atomic functions, and index n of the atom is saved to the CELL array. Helper array molsInCells is used to store the number of atoms currently present in a cell.

Algorithm 1. Binning atoms into cells on GPU.

```
Require: bid is the block index
Require: tid is the index of a thread within the block
Require: $M$ is the number of threads per block
Require: molsInCells stores the number of atoms in each cell

1. $K \leftarrow floor(N/M)$
2. count $\leftarrow 0$
3. while count $\leq K$ do
4.     n $\leftarrow M*$count+tid
5.     $r \Leftarrow$ dr[n]
6.     Define the index $c$ of the cell using $r$
```

```
 7.     **if** $c$ == bid **then**
 8.         shift $\leftarrow$ atomicAdd(&molsInCells[c], 1)
 9.         CELL[ c*$M$ + shift ] $\Leftarrow$ n
10.     **end if**
11.     count $\leftarrow$ count +1
12. **end while**
```

Similarly to Ref. [35], main memory reads and writes are denoted by $\Leftarrow$, and memory operations with local registers are denoted by $\leftarrow$. Algorithm 1 is not optimal, as the blocks read coordinates sequentially, so at first not all the blocks work in parallel. Also at the end of computations the block which reads the positions first, has to wait until the last block finishes. Also there are some places where divergent warps occur, and the use of atomic functions reduces the performance. Nevertheless, this approach should be faster than binning atoms on CPU especially for large systems, as it eliminates the redundant copying of coordinates from device to host and back, which is prohibitively slow. Additionally, binning is performed once per 10 steps or per larger number of steps, which minimizes its influence on the total performance.

Few words about the algorithm of the calculation of EAM contribution. As the expressions for the potential energy of metal Eq. (2) and for the force acting on a metal atom Eq. (9) consist of two parts – pairwise, which depends on the relative distance between the atoms, and EAM part which depends on electronic density of all atoms different from the given one, the calculation of V_{eam} and of the force is achieved in two stages. At first, the electronic density ρ is calculated for each atom, and then the mentioned quantities are computed. For calculation of ρ the algorithm very similar to A2 from Ref. [35] is employed. The difference is that on step 9 of A2 ρ is calculated instead of the force $\mathbf{F}$, and then the algorithm deals with ρ instead of $\mathbf{F}$.

Not all details of the program algorithms have been presented in this section, but only the basis. Some supplementary algorithms, such as, building the initial coordinates of graphene and metal atoms, insertion of atoms in `Surface Growth` regime, computing radial distribution function etc. implemented on GPU are not given here. The details of these algorithms can be deduced from the code.

3. Program

This section discusses functional possibilities of the program and also describes the code.

3.1. Interface and functionality

SurfaceGrowth is a window application intended for the work in Microsoft (MS) Windows XP 32 or 64-bit operating system or higher. The program has a simple GUI implemented using the Windows application programming interface (API). GUI allows to choose the regime (`Bulk`, `Surface Growth`, `Shear`), a metal and to specify the corresponding parameters. Besides that, it is possible to choose the file paths for saving the results, and to display the system in an OpenGL (OGL) window during the calculations. It should be noted that the calculations become slower in about 3–8 times when the OGL window is used. This is caused by the fact that in this case NL is updated every step but not every 10 steps, so OGL window should be used only for testing or demonstrative purposes. When OGL window is not used, there is the possibility to create backup files and also to restart the calculations using these files if the simulation was interrupted.

The detailed description of GUI is given in the user guide accompanying the code.

3.2. Code

3.2.1. Solution

SurfaceGrowth is implemented in MS Visual Studio 2008 using NVIDIA CUDA Toolkit 3.2 [31]. Solution SurfaceGrowth contains one project SurfaceGrowth. CUDA SDK must be installed for its work, because some of the files from the SDK are used (see user guide for the installation details). The project contains the following files:

- three C++ files: <*render_particles.cpp*>, <*shaders.cpp*> are taken from the "particles" sample in CUDA SDK, and they contain the code responsible for displaying the system in the OGL window. File <*SurfaceGrowth.cpp*> contains the code responsible for the GUI, and also it calls wrappers of the CUDA kernels;

- one CUDA file <*SurfaceGrowth.cu*>. It contains the implementation of wrappers, all CUDA kernels and also some host functions;

- six header files: <*render_particles.h*>, <*shaders.h*> are taken from the "particles" sample in CUDA SDK and contain declarations necessary for functioning of OGL window; <*resource.h*> is the resource header file for GUI; <*SurfaceGrowthProto.h*> contains declarations of functions which are invoked on the host in <*SurfaceGrowth.cpp*>, and also defines the array of structures containing the parameters for EAM potential for six metals; <*SurfaceGrowth.h*> contains necessary headers, the declaration of `SimParams` structure (see Section 3.2.2) and the prototypes of wrappers; <*ComputeDefs.h*> declares macroses which are used in the computations, major part of these macroses is taken from the code accompanying the book [30];

- icon and resource script files;
- files with .vcproj and .sln extensions inherent to the MS Visual Studio 2008 solutions.

3.2.2. Code details

This subsection highlights the main sides of the code implementation.

`SimParams` structure declared in <*SurfaceGrowth.h*> file plays one of the key roles in the program functionality. It contains about 100 parameters necessary for the operation of the program, providing the connection between GUI, host and device due to the global host instance `g_hSimParams` defined in <*SurfaceGrowth.cpp*>, and global device instance `dparams` defined in <*SurfaceGrowth.cu*>. `dparams` is invoked almost in every kernel, so it is stored in constant memory of the GPU to speedup the data access. Also due to the presence of `SimParams` structure, it is possible to perform backup of the computations. There are enough comments in the declaration of `SimParams` in <*SurfaceGrowth.h*>, so its members are not described here.

The sizes of the simulation box which are stored in `g_hSimParams.region` and calculated in `SetParams()` function are defined by the regime of the simulation. In `Bulk` regime the edge of the simulation box is equal to the product of the number of the unit cells `g_hSimParams.initUcell.x(y,z)` along the corresponding direction on the edge length of the unit cell of the chosen metal. The latter is defined as the length of the cube edge of the volume $m/(4\rho)$, where m is the atomic mass of the metal, ρ is its density, and it is taken into account that there are 4 atoms per unit cell in fcc lattice. In `Surface Growth` and `Shear` regimes the dimensions of the simulation box along x and y directions are defined by the corresponding sizes of the graphene layer as follows:

$$\texttt{g\_hSimParams.region.x} = 8a_C\cos(\pi/6)\cdot\texttt{g\_hSimParams.initUcell.x},$$
$$\texttt{g\_hSimParams.region.y} = 6a_C\cdot\texttt{g\_hSimParams.initUcell.y}, \quad (21)$$

where $a_C = 1.42$ Å is the length of the covalent bond in graphene. The vertical size `g_hSimParams.region.z` of the simulation box is defined by the regime and by the number of metal atoms. If there are no metal atoms, then the size is equal to three lattice edges used in binning for NL building. If the metal atoms are present in the system, then in both regimes the length of the simulation box under graphene is defined by the number of cells `g_hSimParams.cellShiftZ` under the layer. In `Surface Growth` regime, the length of the simulation cell above graphene is equal to the product of 1.8 on the edge length of the cube which one could obtain from the given number of metal atoms by packing them in fcc lattice, plus one lattice constant a of the current metal. In `Shear`, the corresponding length is also defined by the cube edge without 1.8 factor, and it is added to $3.6a$. The total value of `g_hSimParams.region.z` is the sum of the length under and over the layer. Generally, these values are chosen empirically, so they can be adjusted if the need for this arises.

The arrays `g_hr` and `g_dr` play an important role. They store the coordinates of atoms on the host and on the device, respectively. The type of an atom (metal or carbon) is defined using index of an atom in these arrays. Atoms which are stored in the elements from 0 to `g_hSimParams.nMolMe-1` are metal, and indexes from `g_hSimParams.nMolMe` to `g_hSimParams.nMol-1` are carbon. Note that in contrast to velocities `g_hv` and accelerations `g_ha` which have data type `float3`, the coordinates are declared as `float4`. This is made deliberately as `w`-component of the coordinates stores the potential energy which is calculated during the force evaluation.

Host code. For eliminating the complexities of debugging of the code, in <*SurfaceGrowth.cpp*> CUDA kernels are invoked using special functions which are called *wrappers*. They are declared in <*SurfaceGrowth.h*> and implemented in <*SurfaceGrowth.cu*>. Their prototype names contain 'W' letter at the end. Most of them are responsible for the CUDA or OGL interoperability initialization, or for the work with buffers. All wrappers implement error handling by analyzing the value returned from the function `cudaGetLastError()`. If the exception occurred, then its description is requested, the message about the exception is displayed to the user and the application is closed. But as the exceptions are catched not after each kernel, the messages can reflect not the true reason of the error. Also there is no error handling pertained to file functions. The purpose of most of the wrappers can be deduced from the code. The description of some wrappers is given below:

- `SetParametersW` copies the structure `g_hSimParams` into `dparams`;
- `InitCoordsW` initializes coordinates of atoms;
- `DoComputationsGLW` calls all kernels necessary for advancing the system by one time step with OGL window;
- `DoComputationsW` calls all kernels necessary for advancing the system by one time step without OGL window.

As was noted above, <*SurfaceGrowthProto.h*> contains declarations of functions which are invoked on the host. Most of them carry out the initialization of the parameters and of the structure `g_hSimParams`. Some of these functions are listed below:

- `EamInit()` is called in `SetParams()`, carries out the initialization of EAM parameters and copies the values into `g_hSimParams` structure;
- `GrapheneInit()` is called in `SetParams()`, initializes the parameters which are used during the calculation of the forces acting in the graphene layer;
- `SetParams()` initializes most of the parameters stored in `g_hSimParams` structure;
- `SetupJob()` allocates host memory for atomic coordinates, velocities and accelerations, initializes these quantities, defines the name of the output file and opens this file.

Standard Windows APIs which handle GUI and OGL functions are also invoked on the host. Their description can be found in literature [53, 54].

Kernels. Kernels are invoked from wrappers for parallel processing of some task. Kernel prototypes are not declared separately in a header file. Prototype names of kernels contain 'K' letter at the end. Kernels implemented in SurfaceGrowth can be subdivided into three groups:

- kernels working with coordinates or with the integration of equations of motion;

- kernels which are involved in force calculations;
- kernels performing measurements of some quantities.

The number of blocks and threads almost for all functions of the first two groups (except `InitFccCoordsK(float4 *pos)`, `InitSlabCoordsK(float4 *pos)`, `InitGrapheneCoordsK(float4 *pos)`) is defined by the principles of the construction of cells used for NL updating described in Section 2.3.2. Each atom is handled by a separate thread, thus the number of threads *performing useful computations* is equal to the total number of atoms `g_hSimParams.nMol`. The number of blocks (or the block grid size) is defined by the number of cells `g_hSimParams.cells`, which in turn is defined by the size of the simulation box `g_hSimParams.region`, by the cutoff distance of the current metal `g_hSimParams.rCutEam` and the skin value `g_hSimParams.rNebrShell` (see function `SetParams()`). As it is not known in advance how many atoms will be present in a cell, the constant `BLOCK_SIZE` is introduced (defined in <*SurfaceGrowth.h*>) which specifies the number of threads per block, and hence the maximum number of atoms in a cell. This value should be multiple of 32 to allow for the coalesced reads from CELL array. In SurfaceGrowth the value of 64 is chosen, and it takes into account all the six metals. *Total* number of threads is equal to the product of `BLOCK_SIZE` value on the number of blocks, and this number can be much larger than the number of threads performing useful computations. This is caused by the fact that the size of a cell (and hence the maximum number of atoms in it) is defined by the lattice constant a of the chosen metal atom. For metals with smaller lattice, such as Ni, the cells have smaller size, so it is possible to use smaller value of `BLOCK_SIZE` for Ni. But for metals with larger a, such as Pb, it is not possible to use smaller `BLOCK_SIZE`. However, when `BLOCK_SIZE`=64 for Ni many threads will work with virtual atoms, thus the GPU will not operate optimally. Also such an approach imposes the restrictions on the system's size, as there is the limit of the number of blocks which can be invoked in parallel on a GPU.

For the kernels of the third group the number of blocks is defined by the total number of atoms. The block size is fixed and is equal to 512 threads.

The kernels of the first group are listed below.

- `InitFccCoordsK(float4 *pos)` is called from the wrapper `InitCoordsW` in `Bulk` regime. This kernel places metal atoms in the fcc lattice. Metal is subdivided into stripes, which are composed of one unit cell in the xy

plane and of `g_hSimParams.initUcell.z` unit cells in the z direction. Each thread in a block generates coordinates for each of the four atoms located in one unit cell from the stripe of unit cells;

- `InitSlabCoordsK(float4 *pos)` is called from `InitCoordsW` in `Shear` regime, specifies initial coordinates of metal atoms by locating them in a slab with fcc lattice above the graphene layer. Two dimensional indexes of blocks are used here. The number of blocks is equal to the number of unit cells of metal along the x and y directions (this number is calculated in the wrapper). The number of threads in each block is equal to the number of layers. Similarly to the previous kernel, each thread generates coordinates for four atoms of one unit cell of the fcc lattice;

- `InitGrapheneCoordsK(float4 *pos)` is called from `InitCoordsW` in `Surface Growth` and `Shear` regimes, it produces initial coordinates of the graphene layer. Every block contains 32 threads, each thread generates coordinates of one atom, the number of blocks is equal to the total number of carbon atoms divided by 32;

- `LeapfrogStepK(int part, float4 *dr, float3 *dv, float3 *da)` integrates equations of motion using leapfrog method. Each thread calculates data of one atom. Carbon atoms located over the perimeter of the graphene sheet are skipped to provide their rigidity;

- `ApplyBoundaryCondK(float4 *dr)` applies periodic boundary conditions to each atom;

- `ApplyBerendsenThermostat(float3 *dv, real *vvSum, int step Count)` applies Berendsen thermostat by multiplying the atom velocity by the factor defined in Eq. (19). In the kernel, several conditions are checked which may cause divergent warps. Note that usually the thermostat function is located after the force computation routine (`ComputeForcesK` in our case). But due to technical reasons in SurfaceGrowth the thermostat is applied after the computation of the sum of squares of velocities. Nevertheless, such an architecture does not principally influence the results of the simulations;

- `InsertAtomsK(float4 *dr, float3 *dv, int nMolDeposited, int nMolToDeposit)` is called in `Surface Growth` regime. Injects atoms imitating the deposition process. The index of an atom is compared with the number of deposited atoms and with atoms that should be deposited at the current instance. If the atomic index lies in this interval, its z coordinate is assigned value of `0.49f * dparams.region.z`, and the initial velocity is defined using Eq. (1). If atomic index lies outside the mentioned range, the atom is left outside the simulation box with zero velocity;

- `ApplyShearK(float4 *dr, float3 *da, real shear, real centerOfMassX, uint *numOfSharedMols)` is called in `Shear` regime after equilibration and cooling periods. The kernel applies shear force in the x direction to an atom by adding to the x component of the acceleration the corresponding value. This function selects the necessary atoms which have x component of the coordinates which are smaller than X_{CM} of the nanoparticle. Also the number of such atoms is evaluated in the kernel, and this value is used for the calculation of the total shearing force acting on the nanoparticle;

- `SetColorK(float4 *color)` this kernel is not related to computations, it specifies the colors of atoms which will be displayed in OGL window.

Kernels of the second group, which are used for the force computations are as follows:

- `BinAtomsIntoCellsK(float4 *dr, int *CELL, uint *molsInCells)` implements Algorithm 1 for binning atoms into cells. See Section 2.3.2 for details;

- `BuildNebrListK(float4 *dr, int *CELL, int *NN, int *NBL)` builds neighbor list using Algorithm 4 from the Ref. [35];

- `EamComputeRhoK(real *rho, float4 *dr, int *NN, int *NBL)` calculates electronic density for EAM potential;

- `ComputeForcesK(float3 *a, float4 *dr, int *NN, int *NBL, real *rho, real *fForce)` calculates forces acting between all the atoms. For computing the forces from EAM and LJ potentials, Algorithm 2 from the Ref. [35] is used. Forces in the graphene layer are calculated using specially developed algorithm, which can be deduced from the code.

Almost all the kernels of the third group are based on the reduction algorithm. The general scheme presented in CUDA Programming Guide (p. 111) [51] is employed. The function `calculatePartialSum` is implemented using the algorithm presented in the videolecture "Control Flow.m4v" from Ref. [52]. The approach employs the tree-like structure of the threads during different stages of reduction. In the current version, the tree has only two levels. This fact in combination with the block size of 512 threads imposes the restriction on the maximum size of the system equal to $1024 \times 1024 = 1048576$. The following kernels belong to the third group:

- `ComputeVSumK(float3 *dv, float3 *hlpArray)` calculates the total momentum of the system. The mass of the carbon atom is equal to 1 in dimensionless units, so atomic mass is used only for metal atoms;

- `ComputeVvSumK(float3 *dv, float3 *hlpArray, real cmVelX)` finds the sum of squares of velocities of atoms necessary for computing the kinetic energy of the system. For metal atoms mass is taken into account. In `Shear` regime, the x component V_{X} of the velocity of the CM of the nanoparticle is subtracted form the x component of the velocity of each atom. However, if the diffusion of the nanoparticle is studied, all the three components of the velocity of CM of the nanoisland may have rather large value. This fact is not taken into account in the current version of the program, so y and z velocity components are not subtracted during the calculations. Hence, the resulting temperature may be larger than the specified one. The difference may be larger for higher temperatures;

- `ComputeVvMaxK(float3 *dv, float3 *hlpArray)` defines the maximum square of the atomic velocity used for the check of the update of the neighbor list using formula (20);

- `ComputePotEnergyK(float4 *dr, float3 *hlpArray)` computes total potential energy as a sum of the quantities saved in `w` components of the coordinates;

- `ComputeCenterOfMassK(float4 *dr, float3 *hlpArray)` computes coordinates of the CM of the nanoparticle using the relation $\sum_i \mathbf{r}_i / N_m$, where N_m is the number of metal atoms, $\mathbf{r}_i$ is the atomic coordinate. Division by N_m is performed on the host. Atomic mass is not present, as summation is performed only over metal atoms;

- `ComputeCmVelK(float3 *dv, float3 *hlpArray)` defines the velocity of the CM of the nanoparticle using the relation $\sum_i \mathbf{v}_i / N_m$, where N_m is the number of metal atoms, $\mathbf{v}_i$ is the velocity of an atom;

- `ComputeParticleSizeK(float4 *dr, float3 *hlpArray, int min_max)` depending on the value of `min_max`, finds the maximum or minimum value of atomic coordinates of metal atoms and saves it to the element of `hlpArray` array with 0 index;

- `ComputeNetForceK(real *hlpArray)` calculates x component of the net force acting on metal atoms from carbon atoms. This is assumed to be the friction force along the x direction;

- `EvalRdfK(float4 *dr, int *CELL, int *histRdf, int countRdf)` computes radial distribution function $g(r)$ (RDF). In contrast to the all aforementioned kernels performing measurements, it is the just one kernel which does not use the reduction. This function is based on the A4 from Ref. [35]. The details can be deduced from the code.

In <*SurfaceGrowth.cu*> there are also several `__device__` functions (besides wrappers, host functions and kernels). Their aim is to calculate the EAM functions ϕ, f, F and the corresponding derivatives using formulas given in Section 2.

4. Results

After rather detailed description of the program, this section presents the outcomes of SurfaceGrowth and examples of the results obtained in every regime. The content of the output of the program is described at first, its examples are given after that. It should be noted, that only GPU version of SurfaceGrowth is available, so there is no possibility to compare the computational time spent on GPU with the one of the CPU.

4.1. Measurements

As was mentioned in Section 2.1, in the application it is possible to measure general parameters of the system and the characteristics of the nanoparticle. If the user makes the appropriate choice, the results are periodically printed in a text file. The name of the output file depends on the chosen regime and consists of the part specified by the user, and the part containing the values of some parameters. The number of steps after which the results are saved is specified by the user, and this quantity also defines the averaging time interval. The output is organized in the form of columns with headers. Values of the time step and of some other parameters are printed after the headers in the upper part of the file. The following quantities pertain to the general parameters (the names correspond to the column headers in the output file):

- stepCnt – number of time steps elapsed from the beginning of the simulation;
- impulse – total momentum of the system;
- totEn(eV), totEn.rms(eV) – total energy of the system and its root mean square displacement (RMSD) (in eV) over the averaging period;
- potEn(eV), potEn.rms(eV) – potential energy of the system and its RMSD (in eV) over the averaging period;
- Tempr(K), T.rms(K) – temperature of the system and its RMSD (in K) over the averaging period. As was mentioned in Section 3.2.2, this value can be higher than the specified input one;
- oneStep(ms) – average duration of the calculation of one time-step (in ms). It takes into account only the duration spent on the GPU, and may not reflect the time spent for printing the results in the output files. So the true wall-clock time can be larger.

The follwing characteristics of the nanoparticle are printed in the same file as the general parameters:

- Veloc_CM – x-component of the velocity of the center of mass (CM) of the nanoparticle (dimensionless);
- CM(angstr) – x-component of the coordinate of the CM of the nanoparticle (Å). The origin is in the center of the simulation box. This is “true” value of the CM coordinate, taking into account periodical boundaries;
- CMY(angst) – y-component of the coordinate of the CM of the nanoparticle (Å). The origin is in the center of the simulation box. This is “true” value of the CM coordinate;

- friction(nN) – friction force acting on the nanoparticle (in nN). It is equal to the x component of the force acting on the metal atoms from the carbon ones;
- sizex(angstr), sizey(angstr), sizez(angstr) – sizes of the nanoparticle along the x, y, z directions, respectively (in Å). They are defined as differences between the maximum and minimum coordinate values of metal atoms along the corresponding direction;
- shearForce(nN) – total shearing force acting on the nanoparticle (in nN). It is equal to the product of the value of the shearing force acting on one atom on the number of all metal atoms with values of x coordinates that are smaller than X_{CM}.

Characteristics of the nanoparticle are measured in all regimes, if the user specified non-zero number of metal atoms. If this value is zero, these characteristics are set to zero. However, one has to understand that these characteristics have the complete meaning only in `Shear` regime. In `Bulk` regime all the system corresponds to a nanoparticle, so the values of Veloc_CM, CM(angstr) give the additional opportunity to verify the results of reduction algorithms as their values should be close to zero, and sizex(angstr), sizey(angstr), sizez(angstr) should correspond to the system size. In `Surface Growth` regime for major instances the characteristics of the nanoparticle are meaningless. Only if the continuous layer of atoms or one nanoparticle is formed after the deposition, then these quantities can be used to characterize it.

Besides mentioned parameters, it is also possible to describe the structure of the nanoparticle by measuring the radial distribution function $g(r)$ (RDF) of metal atoms. RDF shows the probability to find a neighboring atom on some distance from the chosen one [30]. The user specifies the time interval in which the measurements of $g(r)$ are performed. After 100 of such measurements the obtained results are averaged and printed in a file in the form of a histogram, which shows how often a coordinate value is encountered in one of the 200 intervals, on which the distance of about 9 Å is subdivided. As was noted above, RDF should be used in `Bulk` and `Shear` regimes, as in `Surface Growth` regime it is not the accurate characteristic in most cases.

Additionally to the output parameters, the application allows to periodically save coordinates of all the atoms in Protein Data Bank (PDB) [55] format (in files with .pdb extension) for the visualization of the system. For this aim it is recommended to use the Visual Molecular Dynamics (VMD) [38] software, which is freeware and also utilizes the NVIDIA CUDA technology.

After the output data of the SurfaceGrowth has been described, some results are briefly discussed below. The results are obtained on NVIDIA GeForce GTX 260 GPU without OGL window. The program may produce similar, but not the same results on another GPUs for the same input parameters.

4.2. Results for `Bulk` *Au*

The results of the simulation of a system consisting of $24 \times 24 \times 24$ unit cells or 55296 gold atoms during 100000 time steps are described here. The snapshot of the system is given in Fig. 7, and a part of the output file is shown in Fig. 8. The first part of the name of the output file has the form "sg_blk_", and then it contains the number of unit cells, number of metal atoms, averaging period, time steps for printing the PDB file, initial temperature and metal type.

Total momentum of the system (denoted by number 1 in Fig. 8) is equal to zero during initial part of the simulation. However, after about 20000 time steps some small deviations from this value occur due to rounding errors and the use of single precision functions. Total energy of the system (totEn(eV), denoted by 2) also conserves its value with relatively good accuracy as it should be in microcanonical ensemble. The value of the potential energy (potEn(eV), number 3 in Fig. 8) reflects the metal binding energy. Note, that the temperature (Tempr(K), number 4) considerably decreased compared to the specified value of 298 K, which is related to the transformation of a part of the kinetic energy into the potential [30]. The velocity of the center of mass of the system is equal to zero and the value of the CM is also close to zero, this is not shown here. Quantities sizex(angstr), sizey(angstr), sizez(angstr) correspond to the system sizes, which are approximately identical and equal to 96.3 Å. The duration of the simulation on the aforementioned GPU for this system is about one hour.

Fig. 9 plots RDF $g(r)$ obtained during the first 5000 time steps. It is represented by a series of relatively sharp spikes of different lengths, which reflect the crystalline structure of Au. The abscissa coordinate of the first spike corresponds to the nearest-neighbor distance in the given metal, and is equal to the value of the parameter r_e of EAM potential, for Au it is 2.89 Å. The second spike corresponds to the lattice constant of about 4.07 Å. The mentioned results can serve the judgement of the correct computation of the EAM potential.

4.3. Results for `Surface Growth` *Al*

The results of deposition of 2000 Al atoms on a graphene sheet consisting of $11 \times 11 \times 32 = 3872$ carbon atoms are briefly discussed in this section. A snapshot of the system after 800000 time steps is given in Fig. 10. In this regime the first part of the name of the output file is "sg_" by default. Then it contains the number of unit cells in graphene, number of metal atoms, equilibration period, how often metal atoms are deposited, the size of the deposited group of atoms, frequency of the generation of PDB files, initial temperature and metal type.

For this regime it is important to select the appropriate values of the parameters. The following parameters were used: deposition energy 0.03 eV, parameters of LJ potential: epsilonLJ = 0.05 eV, sigmaLJ = 2.4945 Å. Metal atoms were deposited by groups of 20 atoms after each 5000 time steps. The groups were generated $2000/20 = 100$ times, and the duration of deposition was $5000 \cdot 100$=500000 time steps. If the number of metal atoms is zero, the total momentum of the system is not zero as the boundary carbon atoms are rigid. During the deposition, islands of Al are formed (cf. Fig. 10). This indicates that for the chosen parameters the growth of thin metal film proceeds accordingly to Volmer-Weber mechanism [56]. Part of the atoms is deposited under the layer which is caused by the use of the periodic boundaries in the z direction. The deposition process can be observed from the video animations, the wall-clock time of the calculations is about 1.7 hour.

4.4. Results for `Shear` *Ni*

Comprehensive investigation of friction of Ni and Ag nanoparticles obtained using SurfaceGrowth in `Shear` regime can be found in Ref. [36]. In this section only the details which were skipped in that work are elucidated for the Ni nanoparticle consisting of 13000 atoms adsorbed on a graphene layer composed of 23×23 unit cells (and contains $23 \times 23 \times 32 = 16928$ carbon atoms). The total number of atoms is 29928. The snapshot of the formed nanoparticle is given in Fig. 11. In `Shear` regime the first part of the name of the output file is by default "sg_sh_". Then it contains the number of unit cells in graphene, number of metal atoms, equilibration, cooling and averaging periods, step for generating PDB files, initial temperature and the type of metal.

Fig. 12 plots time dependencies of friction and shearing fores acting on the nanoparticle, x-components of the velocity and the coordinate of CM, sizes of the nanoisland along x and y directions, and the temperature of the system. During the equilibration period of 115000 time steps, the temperature of the system rises more than to 900 K, the metal atoms melt and the nanoparticle with sizes of $7 \times 6.9 \times 5.15$ nm is formed. Time dependence of the friction force has chaotic shape during this period. After this period the thermostat is applied both to metal and carbon atoms during 150000 time steps, which brings the system temperature to the value of about 300 K. After 265000 time steps, the thermostat is switched off from metal atoms and is coupled only to graphene. Then, shearing force is applied as can be seen from the top part of Fig. 12. The nanoparticle is being sheared, and the x components of the velocity and the coordinate of its CM are being changed linearly and quadratically, respectively, indicating the constant shear force. It can be noted that the friction force has the sawtooth shape which is characteristic of the stick-slip regime, which is also illustrated in Fig. 13. Such a behavior is mainly attributed to the local commensurability of the surface of the nanoparticle and graphene, as the nearest-neighbor distance in Ni of 2.49 Å is very close to the value of the lattice vector in graphene equal to 2.46 Å. The reader is advised to refer to Ref. [36] for more details, where some studies of the diffusion of nanoislands using `Shear` regime of SurfaceGrowth can also be found.

5. Conclusion

This article provides quite comprehensive description of SurfaceGrowth program, starting from basic operation of the application in different regimes, proceeding with the model, algorithms for GPUs and code description, and ending with brief discussion of the results. The work with GUI is placed in separate user guide distributed with the code of the program. In principle, not so much effort is required to redesign the program for Linux systems, as this involves only the modification of the part responsible for GUI.

SurfaceGrowth can be optimized in several ways, in particular, using textures for random accessed memory, sorting of the particles and hardware-implemented (intrinsic) mathematical functions. Also enlarging the number of levels in the trees used for reduction will enable to simulate the systems containing more than 1048576 atoms. Computation of the diffusion constant of the center of mass of the nanoparticle is to be added to the future releases of the program. Additionally, the code may serve as the basis for another models, as it is possible to add another substrates, metals or several nanoparticles with relatively small effort. This may enable simulating surface diffusion and friction process of a nanoparticle adsorbed on different surfaces. The inclusion of several nanoparticles provides the way for studying the coalescence of nanoislands, which also may be important as nanoparticles in the experiments are usually composed of smaller coalesced nanoparticles [10]. Algorithms for the effective calculation of long-range electrostatic forces developed recently [57, 58] can be included in the program allowing the simulation of a wide variety of systems. The mentioned modifications of the code should be made in future releases of SurfaceGrowth.

6. Acknowledgements

I would like to thank Bo Persson for providing me the opportunity to visit Jülich Forschungszentrum, and the staff of the Theory 1 group of the Institute for Solid-State Research of Jülich Forschungszentrum for their hospitality during my visit when this article was written. This research was supported in part by the Fundamental Researches State Fund of Ukraine.

References

[1] M.I. Katsnelson, F. Guinea, A.K. Geim, Phys. Rev. B 79 (2009) 195426.

[2] M. Neek-Amal, R. Asgari, M.R.R. Tabar, Nanotechnology 20 (2009) 135602.

[3] M.H. Gass, U. Bangert, A.L. Bleloch, P. Wang, R.R. Nair, A.K. Geim, Nat. Nanotechnol. 3 (2008) 676.

[4] O.V. Khomenko, M.V. Prodanov, Yu.V. Scherbak, J. Nano-Electron. Phys. 1 (2009) 66.

[5] N. Patra, B. Wang, P. Král, Nano Lett. 9 (2009) 3766.

[6] V.H. Crespi, Nature 462 (2009) 858.

[7] M. Dienwiebel, G.S. Verhoeven, N. Pradeep, J.W.M. Frenken, J.A. Heimberg, H.W. Zandbergen, Phys. Rev. Lett. 92 (2004) 126101.

[8] A.V. Khomenko, N.V. Prodanov, Carbon 48 (2010) 1234.

[9] D. Dietzel, M. Feldmann, H. Fuchs, U.D. Schwarz, A. Schirmeisen, Appl. Phys. Lett. 95 (2009) 053104.

[10] D. Dietzel, T. Mönninghoff, C. Herding, M. Feldmann, H. Fuchs, B. Stegemann, C. Ritter, U.D. Schwarz, A. Schirmeisen, Phys. Rev. B 82 (2010) 035401.

[11] G. Paolicelli, M. Rovatti, A. Vanossi, S. Valeri, Appl. Phys. Lett. 95 (2009) 143121.

[12] M. Rovatti, G. Paolicelli, A. Vanossi, S. Valeri, Meccanica (2011, in press) doi:10.1007/s11012-010-9366-0.

[13] C. Lee, X. Wei, Q. Li, R. Carpick, J.W. Kysar, J. Hone, Phys. Status Solidi B 246 (2009) 2562.

[14] C. Lee, Q. Li, W. Kalb, X.-Z. Liu, H. Berger, R.W. Carpick, J. Hone, Science 328 (2010) 76.

[15] A.K. Geim, Science 324 (2009) 1530.

[16] N.V. Prodanov, A.V. Khomenko, Surf. Sci. 604 (2010) 730.

[17] A.V. Khomenko, N.V. Prodanov, Functional Materials 17 (2010) 230.

[18] D. Sen, K.S. Novoselov, P.M. Reis, M.J. Buehler, Small 6 (2010) 1108.

[19] L. Bardotti, P. Jensen, A. Hoareau, M. Treilleux, B. Cabaud, A. Perez, F. Cadete Santos Aires, Surf. Sci. 367 (1996) 276.

[20] P. Jensen, Rev. Mod. Phys. 71 (1999) 1695.

[21] W.D. Luedtke; U. Landman, Phys. Rev. Lett. 82 (1999) 3835.

[22] L.J. Lewis, P. Jensen, N. Combe, J.-L. Barrat, Phys. Rev. B 61 (2000) 16084.

[23] B. Yoon, W.D. Luedtke, J. Gao, U. Landman, J. Phys. Chem. B 107 (2003) 5882.

[24] Y. Maruyama, Phys. Rev. B 69 (2004) 245408.

[25] P. Jensen, A. Clément, L.J. Lewis, Comput. Mater. Sci. 30 (2004) 137.

[26] R. Guerra, U. Tartaglino, A. Vanossi, E. Tosatti, Nat. Mater. 9 (2010) 634.

[27] D.A. Aruliah, M.H. Müser, U.D. Schwarz, Phys. Rev. B 71 (2005) 085406.

[28] E. Gnecco, E. Meyer, Fundamentals of Friction and Wear on the Nanoscale (Springer, Berlin, 2007).

[29] M. Griebel, S. Knapek, G. Zumbusch, Numerical Simulation in Molecular Dynamics (Springer, Berlin, Heidelberg, 2007).

[30] D.C. Rapaport, The Art of Molecular Dynamics Simulation, 2nd ed. (Cambridge University Press, Cambridge, 2004).

[31] NVIDIA CUDA webpage, http://www.nvidia.com/object/cuda_home_new.html.

[32] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, J. Comp. Chem. 28 (2007) 2618.

[33] J.A. van Meel, A. Arnold, D. Frenkel, S.F. Portegies Zwart, R.G. Belleman, Mol. Sim. 34 (2008) 259.

[34] W. Liu, B. Schmidt, G. Voss, W. Müller-Wittig, Comp. Phys. Commun. 179 (2008) 634.

[35] J.A. Anderson, C.D. Lorenz, A. Travesset, J. Comp. Phys. 227 (2008) 5342.

[36] A.V. Khomenko, N.V. Prodanov, J. Phys. Chem. C 114 (2010) 19958.

[37] H. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola, J.R. Haak, J. Chem. Phys. 81 (1984) 3684.

[38] W. Humphrey, A. Dalke, K. Schulten, J. Molec. Graphics 14 (1996) 33 (http://www.ks.uiuc.edu/Research/vmd/).

[39] A. Geissler, M. He, J.-M. Benoit, P. Petit, J. Phys. Chem. C 114 (2010) 89.

[40] Y. Shibuta, J.A. Elliott, Chem. Phys. Lett. 427 (2006) 365.

[41] H.N.G. Wadley, X. Zhou, R.A. Johnson, M. Neurock, Prog. Mater. Sci. 46 (2001) 329.

[42] X.W. Zhou, H.N.G. Wadley, R.A. Johnson, D.J. Larson, N. Tabat, A. Cerezo, A.K. Petford-Long, G.D.W. Smith, P.H. Clifton, R.L. Martens, T.F. Kelly, Acta Mater. 49 (2001) 4005.

[43] M.S. Daw, M.I. Baskes, Phys. Rev. B. 29 (1984) 6443.

[44] ASM Metals Handbook, Vol. 2, Properties and Selection: Nonferrous Alloys and Special-Purpose Materials, 10th ed. (American Society for Metals, Metals Park, OH, 1992).

[45] N. Sasaki, K. Kobayashi, M. Tsukada, Phys. Rev. B 54 (1996) 2138.

[46] D.W. Brenner, Phys. Rev. B 42 (1990) 9458.

[47] D.W. Brenner, O.A. Shenderova, J.A. Harrison, S.J. Stuart, B. Ni, S.B. Sinnott, J. Phys.: Condens. Matter 14 (2002) 783.

[48] A.C.T. van Duin, S. Dasgupta, F. Lorant, W.A. Goddard III, J. Phys. Chem. A 105 (2001) 9396.

[49] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids (Clarendon Press, Oxford, 1987).

[50] D. Frenkel, B. Smit, Understanding Molecular Simulation (Academic Press, London, 2002).

[51] NVIDIA CUDA C Programming Guide (v. 3.2), http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/, Oct. 2010.

[52] CUDA University Courses, University of Illinois : ECE 498AL, http://developer.nvidia.com/object/cuda_training.html

[53] C. Petzold, Programming Windows, 5th ed. (Microsoft Press, 1998).

[54] D. Shreiner, The Khronos OpenGL ARB Working Group, The Official Guide to Learning OpenGL, Versions 3.0 and 3.1, 7th ed. (Addison-Wesley, 2009).

[55] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, P. Bourne, Nucleic Acids Research 28 (2000) 235 http://www.rcsb.org/pdb.

[56] G.H. Gilmer, H. Huang, C. Roland, Comput. Mater. Sci. 12 (1998) 354.

[57] M.J. Harvey, G. De Fabritiis, J. Chem. Theory Comput. 5 (2009) 2371.

[58] P.K. Jha, R. Sknepnek, G. Iván Guerrero-García, M. Olvera de la Cruz, J. Chem. Theory Comput. 6 (2010) 3058.

7. List of figures

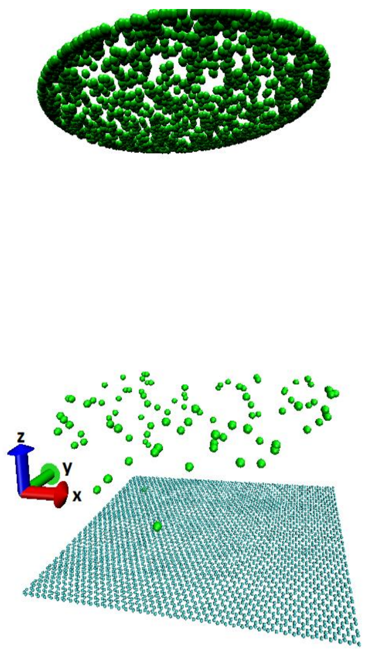


Figure 1: System in `Surface Growth` regime, the deposition of aluminium atoms is shown. Al and C atoms are denoted as green and cyan balls, respectively. Al atoms which are arranged in a circle at the upper part of the figure correspond to atoms which have not been injected into the simulation cell, located below. This arrangement, however, does not correspond to the true z-coordinate of non-injected Al atoms, and serves only the demonstrative purpose. All snapshots in this work were produced with Visual Molecular Dynamics software [38].

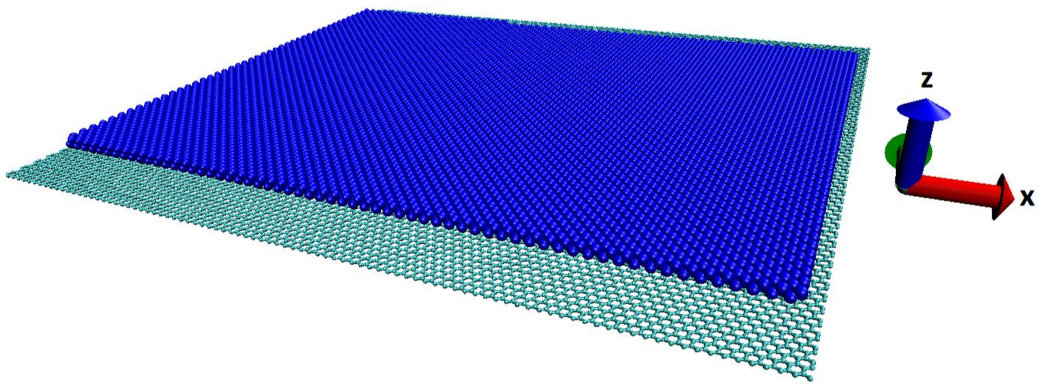


Figure 2: Initial configuration of the system in `Shear` regime for 11000 Ni atoms (blue balls) adsorbed on a graphene layer containing 16928 atoms.

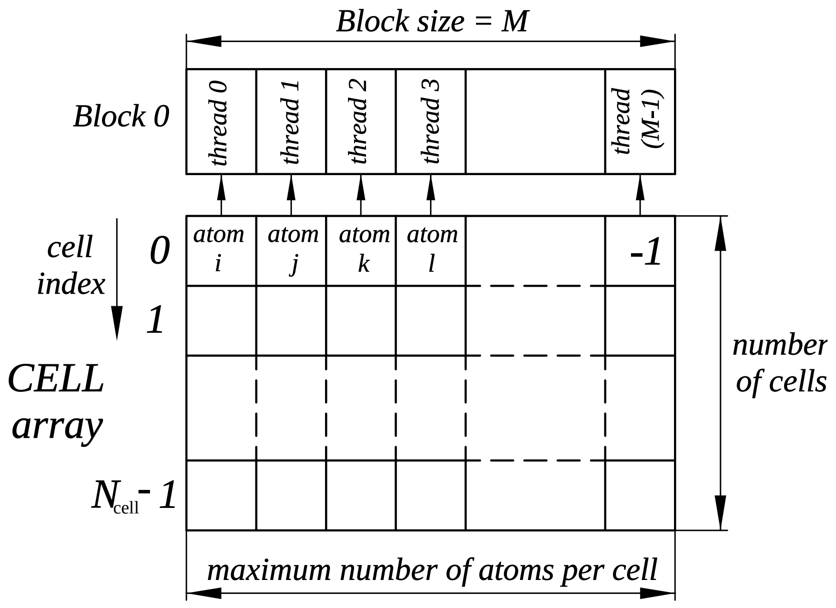


Figure 3: Schematic view of CELL array. All threads of the block 0 read atoms from the row which corresponds to the cell 0. Indexes of atoms in this figure are arbitrarily chosen.

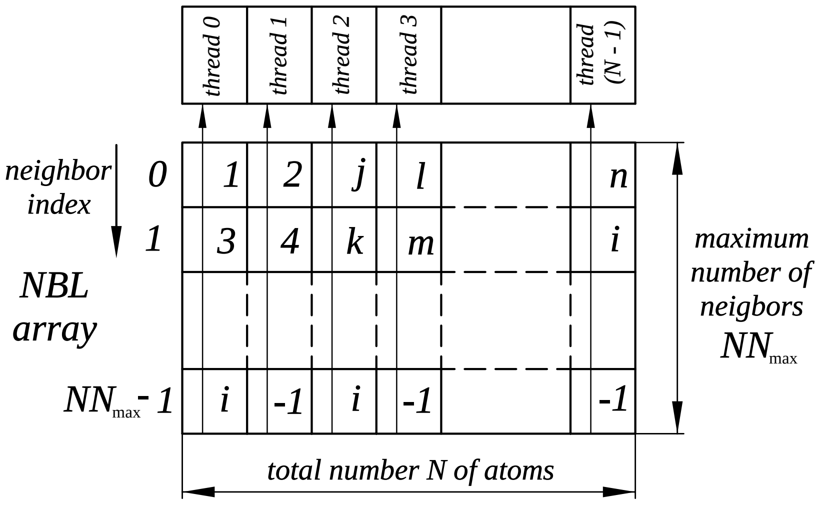


Figure 4: Schematic view of NBL array. Each thread reads neighbors of the associated atom from the corresponding column. Indexes of neighbors in this figure are arbitrarily chosen.

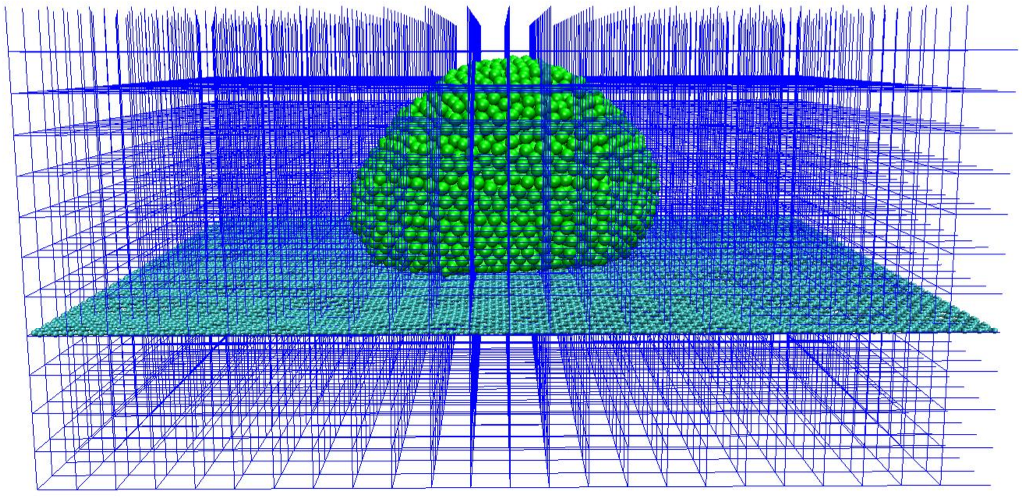


Figure 5: Subdivision of the simulation box into the cells for 16x16 graphene and 5000 gold atoms.

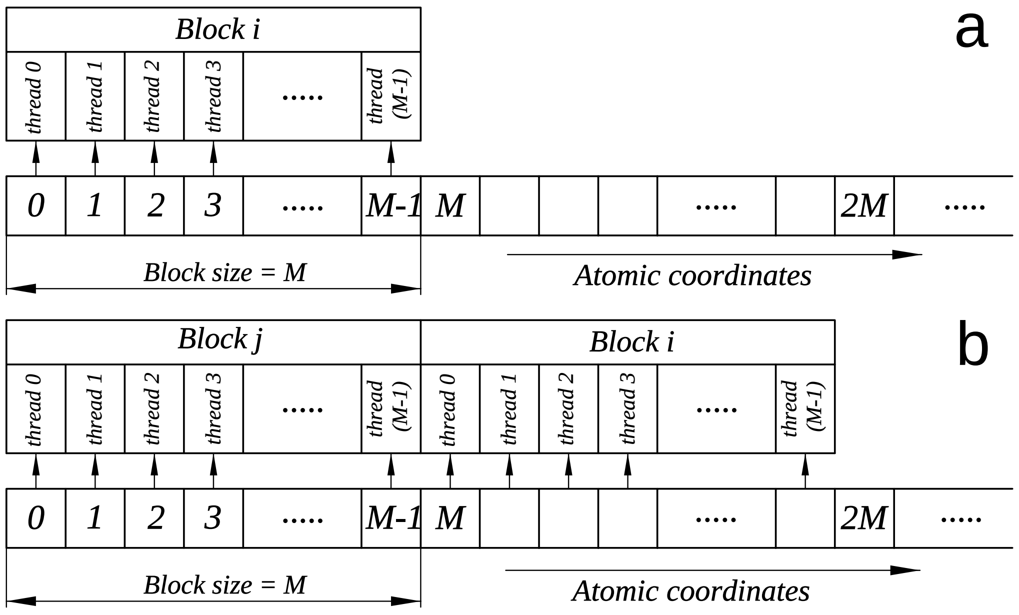


Figure 6: Schematic explanation of the binning algorithm. At first, all threads in a block i read one portion of M atomic coordinates and process them (a). On the second step, the threads in the block i read and process another portion of M coordinates, and threads in a block j work with previous portion of the coordinates (b). This process continues until all the atomic coordinates are processed by every block running on the GPU.

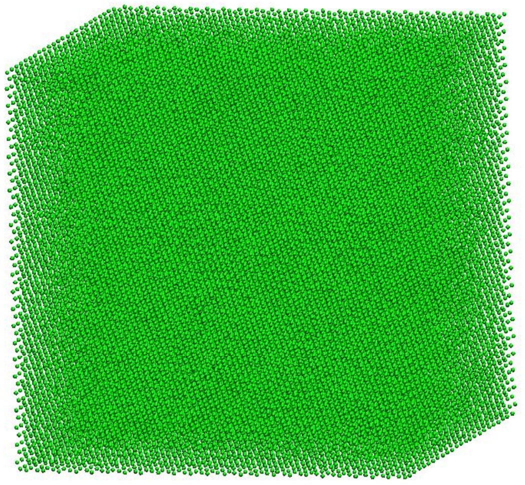
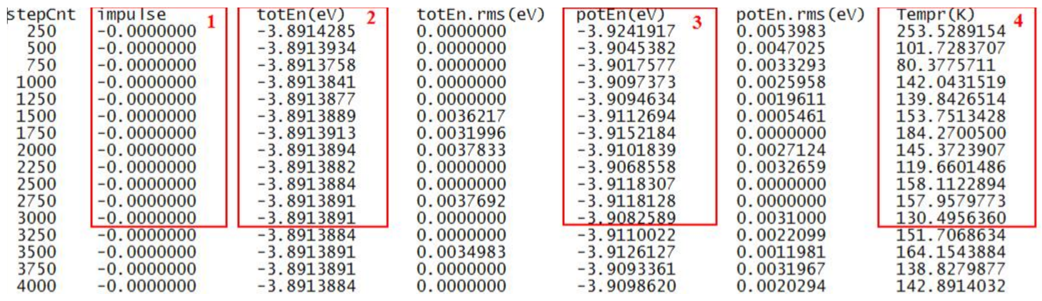


Figure 7: Snapshot of the system in `Bulk` regime containing 55296 gold atoms.



```
stepCnt  impulse     1  totEn(eV)   2  totEn.rms(eV)  potEn(eV)   3  potEn.rms(eV)  Tempr(K)     4
   250   -0.0000000     -3.8914285     0.0000000      -3.9241917     0.0053983      253.5289154
   500   -0.0000000     -3.8913934     0.0000000      -3.9045382     0.0047025      101.7283707
   750   -0.0000000     -3.8913758     0.0000000      -3.9017577     0.0033293      80.3775711
  1000   -0.0000000     -3.8913841     0.0000000      -3.9097373     0.0025958      142.0431519
  1250   -0.0000000     -3.8913877     0.0000000      -3.9094634     0.0019611      139.8426514
  1500   -0.0000000     -3.8913889     0.0036217      -3.9112694     0.0005461      153.7513428
  1750   -0.0000000     -3.8913913     0.0031996      -3.9152184     0.0000000      184.2700500
  2000   -0.0000000     -3.8913894     0.0037833      -3.9101839     0.0027124      145.3723907
  2250   -0.0000000     -3.8913882     0.0000000      -3.9068558     0.0032659      119.6601486
  2500   -0.0000000     -3.8913884     0.0000000      -3.9118307     0.0000000      158.1122894
  2750   -0.0000000     -3.8913891     0.0037692      -3.9118128     0.0000000      157.9579773
  3000   -0.0000000     -3.8913891     0.0000000      -3.9082589     0.0031000      130.4956360
  3250   -0.0000000     -3.8913884     0.0000000      -3.9110022     0.0022099      151.7068634
  3500   -0.0000000     -3.8913891     0.0034983      -3.9126127     0.0011981      164.1543884
  3750   -0.0000000     -3.8913891     0.0000000      -3.9093361     0.0031967      138.8279877
  4000   -0.0000000     -3.8913884     0.0000000      -3.9098620     0.0020294      142.8914032
```

Figure 8: Some data columns from the output file obtained in the `Bulk` regime.

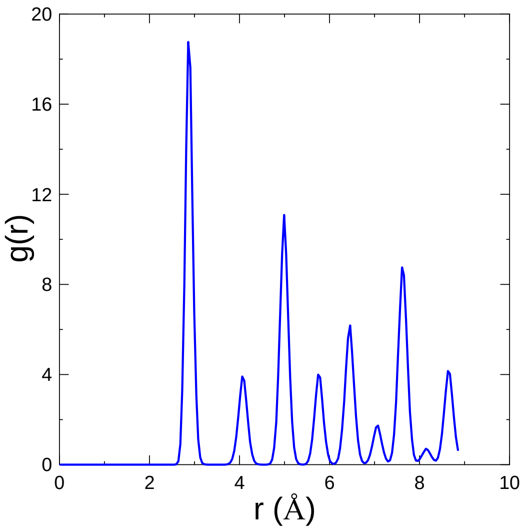


Figure 9: RDF for gold atoms in the `Bulk` regime.

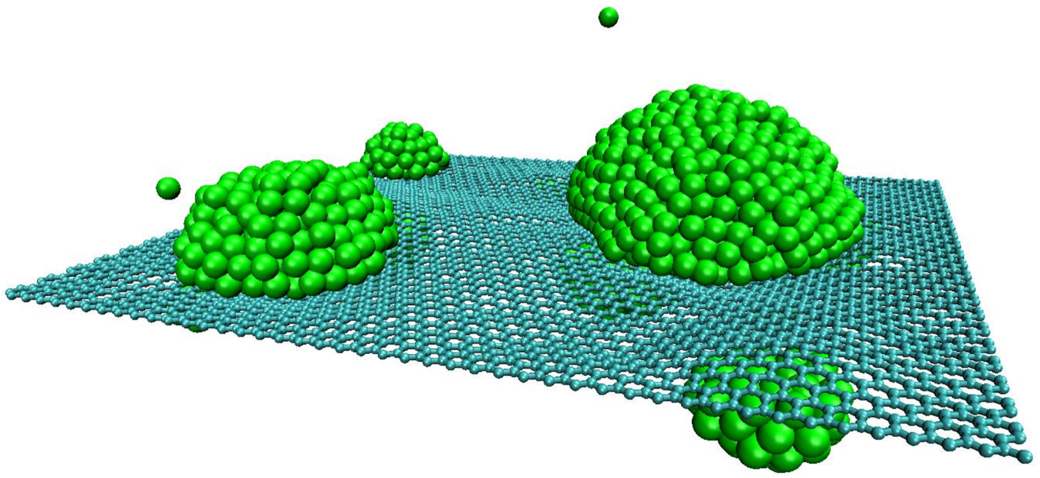


Figure 10: Snapshot of the Al (green balls) deposited on graphene (cyan balls) at the end of the simulation with parameters described in the text.

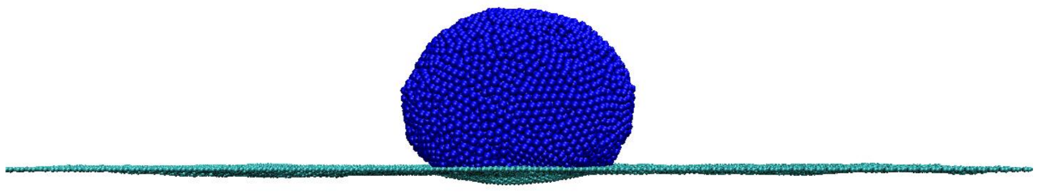


Figure 11: Side view of the formed Ni nanoparticle consisting of 13000 atoms.

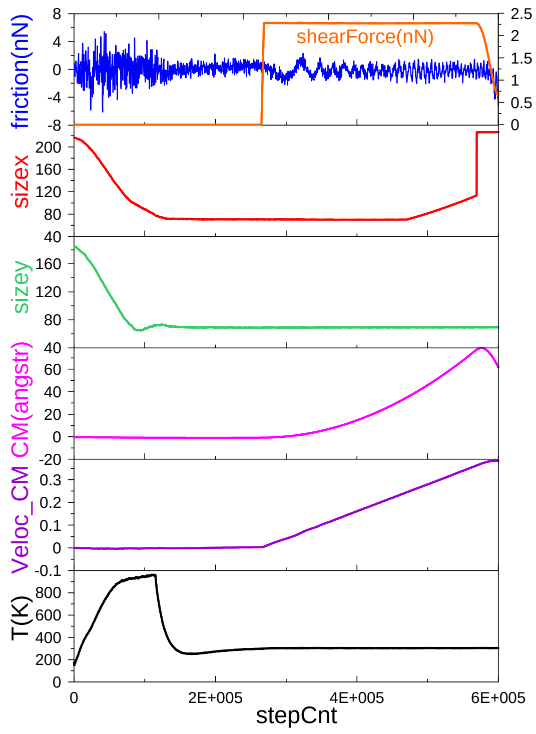


Figure 12: Time dependencies of the friction force, lateral sizes, x components of the coordinate and the velocity of the center of mass, and temperature for Ni nanoparticle described in the text.

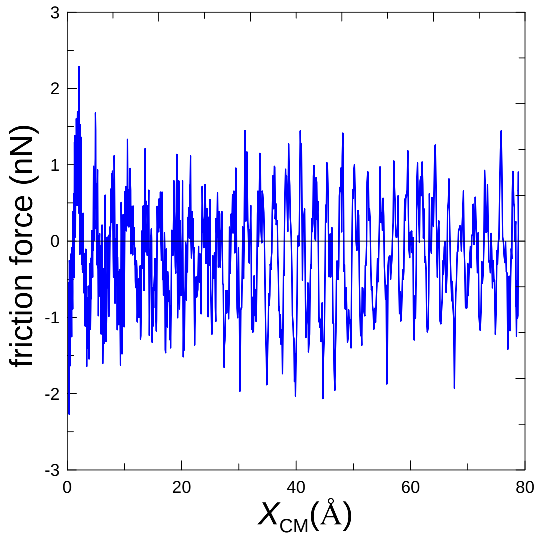


Figure 13: Dependence of the friction force on the lateral position of the center of mass of the Ni nanoparticle for the system discussed in the text.