Molecular dynamic simulation of water vapor interaction with various types of pores using hybrid computing structures *

V.V.Korenkov[1] E.G.Nikonov[1] M.Popovičová[2]

[1]Joint Institute for Nuclear Research,
141980 Dubna, Moscow Region, Russia

[2]University of Prešov,
str. Konštantinova 16, 080 01 Prešov, Slovakia

The 8th International Conference
"Distributed Computing and Grid-technologies in Science and Education"
Dubna, 10 - 14 September 2018

*This work was supported by the JINR project No. 05-6-1118-2014/2019, protocol No. 4596-6-17/19, and used HybriLIT resources.

Introduction

In this work, a study of efficiency of various implementations algorithms for MD simulation of water vapor interaction with individual pore is carried out.
A great disadvantage of MD is its requirement of a relatively large computational effort and long time in simulations.
These problems can be drastically reduced by parallel calculations.
In this work we investigate dependence of time required for simulations on different parameters, like number of particles in the system, shape of pores, and so on. The results of parallel calculations are compared with the results obtained by serial calculations. Two-dimensional and three-dimensional models of the pore are used for comparative analysis of parallel and serial calculations.

Molecular dynamics model

In classical molecular dynamics, the behavior of an individual particle is described by the Newton equations of motion[a], which can be written in the following form

$$m_i \frac{d^2 \vec{r_i}}{dt^2} = \vec{f_i}, \tag{1}$$

where i – a particle number, $(1 \leq i \leq N)$, N – the total number of particles, m_i – particle mass, $\vec{r_i}$ – coordinates of position, $\vec{f_i}$ – the resultant of all forces acting on the particle. This resultant force has the following representation

$$\vec{f_i} = -\frac{\partial U(\vec{r_1}, \ldots, \vec{r_N})}{\partial \vec{r_i}} + \vec{f_i}^{ex}, \tag{2}$$

where U – the potential of particle interaction, $\vec{f_i}^{ex}$ – a force caused by external fields.

[a] *Gould H., Tobochnik J., Christian W.,* An Introduction to Computer Simulation Methods, Chapter 8. Third edition, 2005, pp. 267-268.

Molecular dynamics model

For a simulation of particle interaction, we use the **Lennard-Jones potential**[a]

$$U(r) = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \tag{3}$$

with $\sigma = 3.17$Å and $\varepsilon = 6.74 \cdot 10^{-3}$ eV.
Here r – the distance between the centers of the particles, ε – the depth of the potential well, σ – the distance, where the energy of interaction becomes equal to zero. Parameters ε and σ are characteristic for each sort of atom. The minimum of the potential is reached when $r_{min} = \sigma \sqrt[6]{2}$.
It is the most used to describe the evolution of water in liquid and saturated vapor form.

[a] *J. E. Lennard-Jones,* On the Determination of Molecular Fields. // Proc. Roy. Soc. — 1924, — vol. A 106, — P. 463—477.

Molecular dynamics model

Equations of motion (1) were integrated by **Velocity Verlet method**[a]. Within the framework of the Velocity Verlet method integrating the equations of motion is performed as follows:

- At the beginning of each step values $r(t),\ v(t),\ f(t)$ at the time t are defined or calculated in the previous step.
- First, the coordinates of the particle's new location are calculated at time $t + \Delta t$, then the velocities of particles are calculated at time $t + \frac{\Delta t}{2}$

$$\begin{aligned} r(t+\Delta t) &= r(t) + \Delta t\ v\,(t) + \frac{(\Delta t)^2}{2} a(t), \\ v\left(t + \frac{\Delta t}{2}\right) &= v(t) + \frac{\Delta t}{2}\ \frac{f(t)}{m}. \end{aligned}$$

[a] *Verlet L.,* Computer 'experiments' on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. // Phys. Rev. — 1967, — vol. 159, — p. 98–103.

Molecular dynamic model

- Next, the forces $f(t)$ acting on the particles are recalculated at time $t + \Delta t$.
- Finally, the values of velocities are calculated at time $t + \Delta t$

$$v(t + \Delta t) = v\left(t + \frac{\Delta t}{2}\right) + \frac{\Delta t}{2}\ \frac{f(t + \Delta t)}{m}.$$

Molecular dynamic model

In our work, Berendsen thermostat[a] is used for temperature calibration and control. This thermostat uses alternating nonlinear friction in the equations of motion and is realized by the following equations.

$$\begin{aligned} \frac{dr_i(t)}{dt} &= v_i(t), \\ \frac{dv_i(t)}{dt} &= \frac{f_i(t)}{m_i} - \lambda(t)v_i(t). \end{aligned} \tag{4}$$

[a] *H.J.C.Berendsen, J.P.M.Postma, W.F.van Gunsteren, A.DiNola, J.R.Haak,* Molecular dynamics with coupling to an external bath. // J. Chem. Phys. — 1984, — vol. 81, — P. 3684–3690.

Molecular dynamic model

The coefficient of the velocity recalculation $\lambda(t)$ at every time step t

$$\lambda(t) = \left[1 + \frac{\Delta t}{\tau_B}\left(\frac{T_0}{T(t)} - 1\right)\right]^{\frac{1}{2}}. \tag{5}$$

depends on the so called "rise time" of the thermostat τ_B which belongs to the interval $[0.1, 2]$ ps.
τ_B describes strength of the coupling of the system to a hypothetical heat bath. For increasing τ_B, the coupling weakens, i.e. it takes longer to achieve given temperature T_0 from current temperature $T(t)$.

The Berendsen algorithm is simple to implement and it is very efficient for reaching the desired temperature from far-from-equilibrium configurations.

Molecular dynamic model

Initial concentrations were obtained from the density of water vapor at the appropriate pressure and density at a given temperature using known tabulated data. The pressure in the pore was controlled using the formula based on virial equation[a].

$$P = \frac{1}{3V}\left(\langle 2K\rangle - \left\langle \sum_{i<j} r_{ij}\cdot f\left(r_{ij}\right)\right\rangle\right).$$

Here V is the pore volume, $\langle 2K\rangle$ is the doubled kinetic energy averaged over the ensemble, $f\left(r_{ij}\right)$ is the force between particles i and j at a distance r_{ij}.

[a] Frenkel2002 *Frenkel D., Smith B.,* Understanding molecular simulation : from algorithms to applications. Second edition, Academic Press, 2006, 658 pp.

Computational algorithm for molecular dynamic simulation

For molecular dynamic simulation we used the code written in **CUDA C**. The program does not require a lot of memory. We only keep co-ordinates, speeds and forces for each particle. One of the main problems of molecular dynamic simulation is a large number of particles and time steps. Therefore it is necessary to use parallel calculations. The code for our simulations was implemented on heterogeneous computing cluster *HybriLIT*.

Computational algorithm for molecular dynamic simulation

The code contains four functions that are paralleled and which are performed on the **GPU**. This is a function for calculating the forces (i.e., acceleration) for individual particles, which calculates the interactions between all particles (F1). There are two functions to calculate new coordinates and speeds for each particle. We need two functions (F2 and F3) to calculate them because we need forces acting on particles at two different time moments. Finally, we use the Berendsen thermostat in the program that runs parallel to the **GPU** too (F4).

Computational algorithm for molecular dynamic simulation

We use natural parallelism for molecular dynamic simulations. The force calculations and velocity/position updates can be done simultaneously for all particles.
There are two basic ideas how to achieve parallelism. The goal in each is to divide computations evenly across the processors so as to extract maximum effect.
In the first class of methods a subgroup of particles is assigned to each processor. This method is called an particle-decomposition of the workload. The processor performs all calculations on its particles no matter where they move in the simulation domain.

Computational algorithm for molecular dynamic simulation

The second group of methods is called a spatial decomposition of the workload. It means that parts of the physical simulation domain is assigned to each processor. Each processor only works with the particles in its subdomain.
Our program uses an particle-decomposition method. One command provides processing of a large amount of data that depends on how the block is defined in the program. The pseudo code for all four parallel functions is in Fig. 1 - 4

Computational algorithm for molecular dynamic simulation

```
__global__void F1( ) {

int tid = threadIdx.x + blockIdx.x*blockDim.x;
while(tid < N) {

// Derive a_tid(t + Δt) from the interaction potential using r(t + Δt)

tid += blockDim.x*gridDim.x;
}
}
```

Figure 1: The function for the calculation of acceleration, related to forces of each particle on the device

Computational algorithm for molecular dynamic simulation

```
__global__void F2( ) {

int tid = threadIdx.x + blockIdx.x*blockDim.x;
while(tid < N) {

/* Calculate r_tid(t + Δt) = r_tid(t) + v_tid(t) * Δt + 0.5 * a_tid(t) * Δt^2
v_tid(t + 0.5 * Δt) = v_tid(t) + 0.5 * a_tid(t) * Δt   */

tid += blockDim.x*gridDim.x;
}
}
```

Figure 2: The function for calculating a position and first part of velocity for each particle performed on the device

Computational algorithm for molecular dynamic simulation

```
__global__void F3( ) {

int tid = threadIdx.x + blockIdx.x*blockDim.x;
while(tid < N) {

// Calculate v_tid(t + Δt) = v_tid(t + 0.5 * Δt) + 0.5 * a_tid(t + Δt) * Δt

tid += blockDim.x*gridDim.x;
}
}
```

Figure 3: The function for calculating the second speed fraction for each particle performed on the device

Computational algorithm for molecular dynamic simulation

```
__global__void F4( ) {

int tid = threadIdx.x + blockIdx.x*blockDim.x;
while(tid < N) {

// Calculate v_tid(t) = v_tid(t) * λ(t)

tid += blockDim.x*gridDim.x;
}
}
```

Figure 4: The function for speed adjustment to ensure the required temperature (The Berendsen thermostat)

Computational algorithm for molecular dynamic simulation

Other calculations are performed on the host. General scheme of the calculation algorithm for two- and three-dimensional molecular dynamic simulation is shown in Fig. 5.

Computational algorithm for molecular dynamic simulation

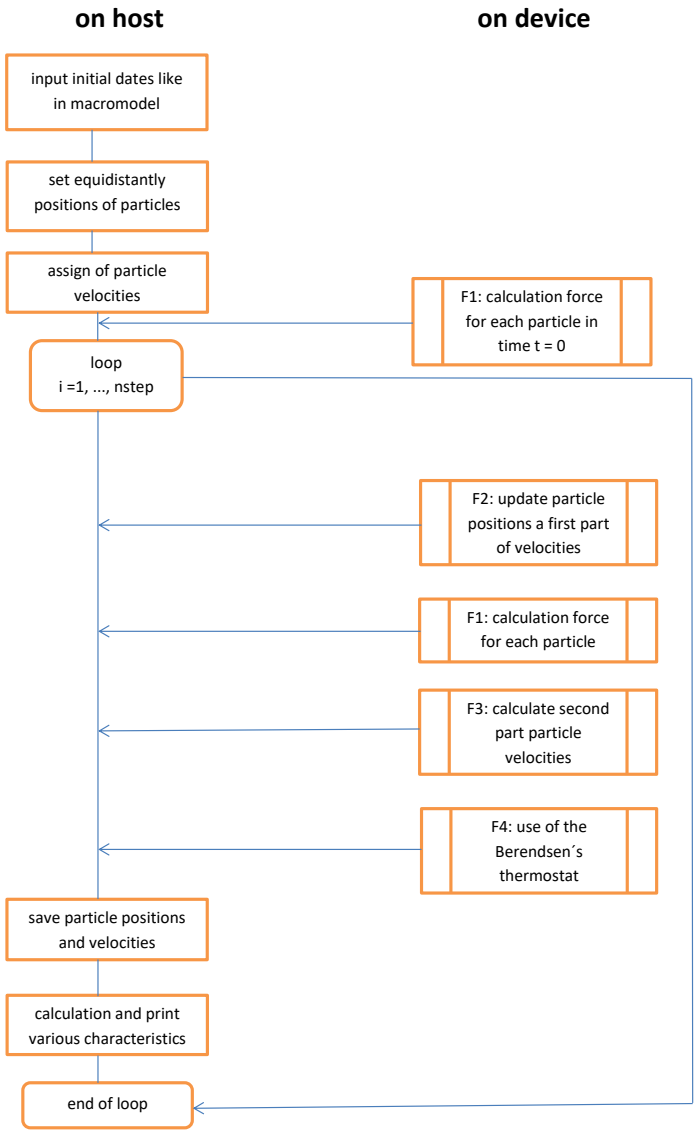

Figure 5: Computational scheme for molecular dynamic simulation

Computational algorithm for molecular dynamic simulation

In this work we compare the temporal realization of these four functions F_1, F_2, F_3, F_4 on the **GPU** and the **CPU**. The total time of parallel computing consists of two parts, that is the time needed directly to calculate on the **GPU** (pure **GPU** time) and the time needed to complete these calculations on the **CPU** because some algorithms performed on the **GPU** must be completed by the **CPU**. In this work, total **GPU** time will indicate the sum of these two times.

2D molecular dynamic simulation

We consider the pore with dimensions $l_x = 1\mu m$, $l_y = 1\mu m$.

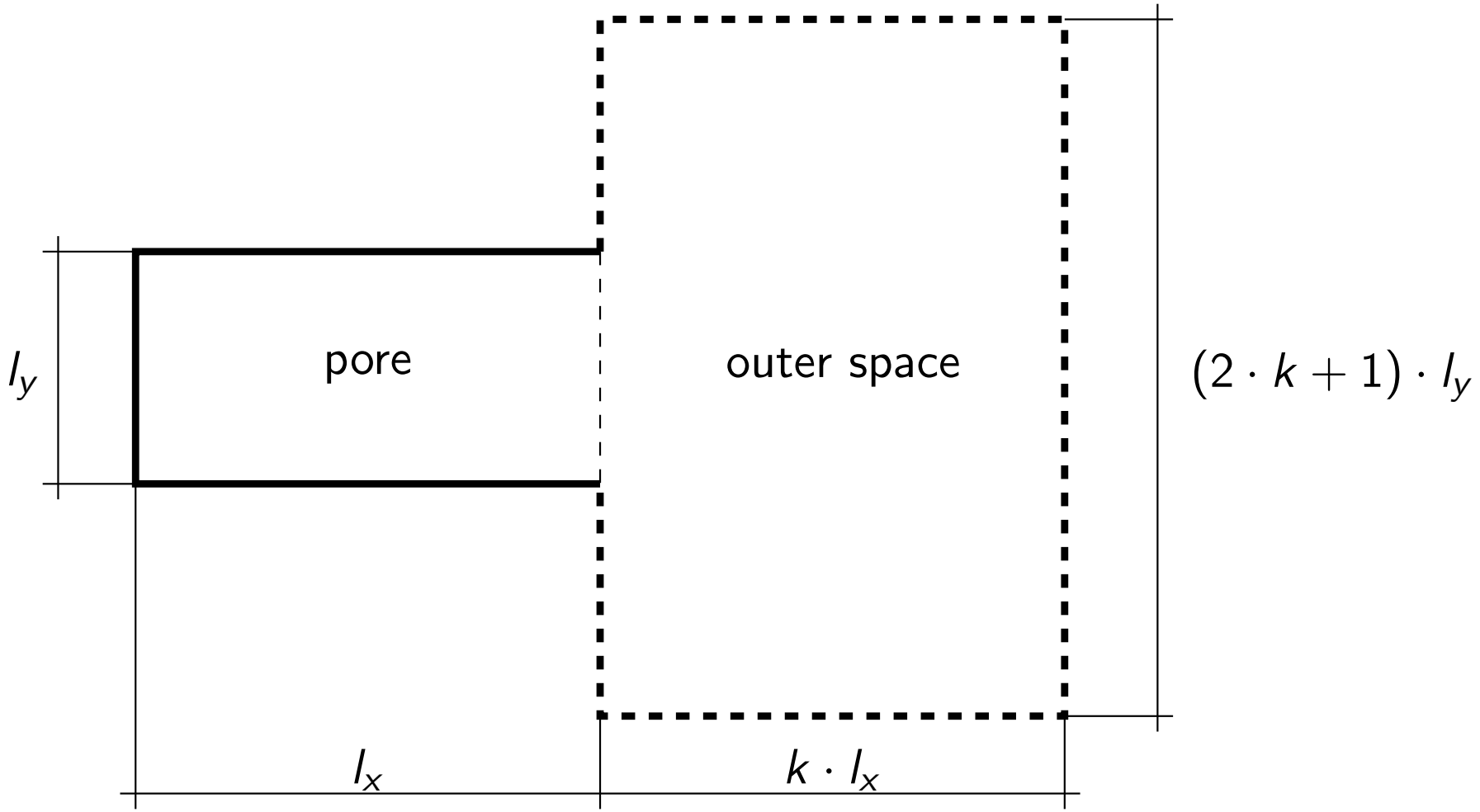

Figure 6: 2D pore and outer space.

2D molecular dynamic simulation

The outer space in this micro-model reflects as a space right to the pore, see Fig. 6 (dashed line) which size, one can change by means of the parameter k.

All sides of the outer space satisfy to the periodic boundary conditions. The left pore side reflects the inner molecules due to the boundary condition[a] but also provides the periodic boundary conditions for a part of outer space. There are 420 molecules of water vapor inside the pore which form saturated water vapor at temperature 35°C and pressure 5.62 kPa at the time $t = 0$. The value of parameter $k = 3$ means that the outer space volume for calculations is 21 times larger that the pore volume. There are 1764 molecules of water vapor in outer space corresponding to 20% saturated water vapor. The integration step is 0.016ps.

[a] *Nikonov E.G., Pavluš M., Popovičová M.*, 2D microscopic and macroscopic simulation of water and porous material interaction. — 2017, — arXiv:1709.05878 [physics.flu-dyn]

2D molecular dynamic simulation

First, we made several runs of our program with 69 blocks of 32 threads. Each implementation took 2000 time steps. We found that performing all the functions at every step in each run is the same with a small deviation. Time averages and deviations for each function are shown in table 1.

F	t_{CPU} (ms)		t_{GPU} (ms)	
	$\bar{t}$	σ_t	$\bar{t}$	σ_t
F1	112.575	2.372	4.479	0.110
F2+F3	0.124	0.005	0.135	0.009
F4	0.038	0.002	0.054	0.003

Table 1: Calculation time averages $\bar{t}$ and deviations σ_t for each function $F_i, i = 1 \div 4$. t_{CPU} – **CPU** calculation time. t_{GPU} – total **GPU** calculation time.

For this reason, we will further consider that all program runs take an average implementation of time.

2D molecular dynamic simulation

Furthermore, the total time of calculating the parallel portion of the code was examined, depending on the number of threads in the blocks (Fig. 7). We see that the minimum time has been reached for 128-threaded blocks (n = 7). Such a dependency pattern did not have all 4 functions that are executed in parallel. The main creator of this result was a function to calculate the potential (F1).

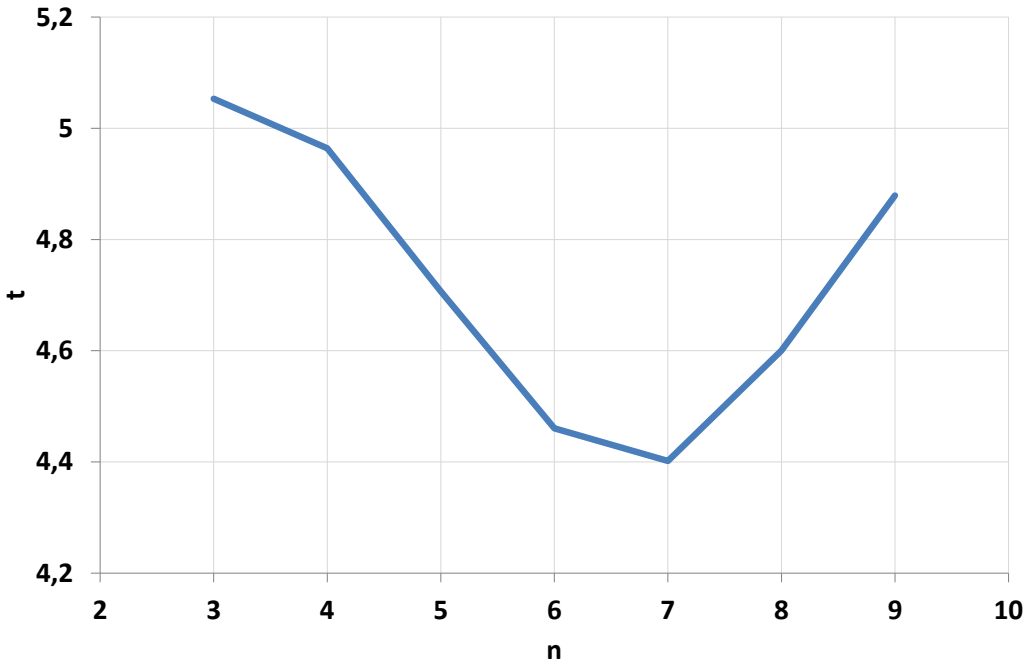

Figure 7: Dependence of the total GPU time calculation on the number of threads per block. t is the time in ms. The number of threads in the block is 2^n.

2D molecular dynamic simulation

The calculation time on CPU varies within the calculated standard deviation (Fig. 8). When comparing CPU and GPU total calculations time, we can see that GPU calculations were performed on average 24 times faster than CPU calculations.

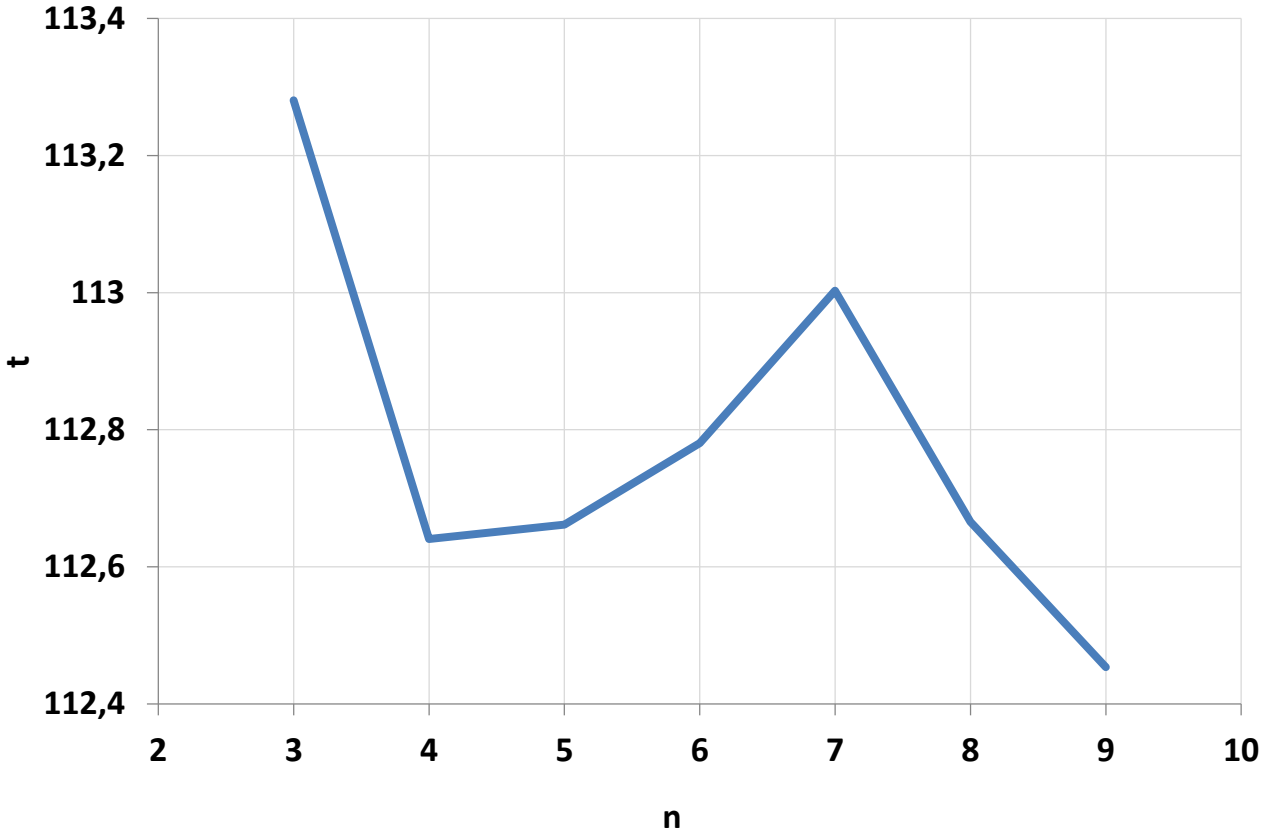

Figure 8: CPU total calculation time dependence on the number of threads in a block. t is the time in ms. The number of threads in the block is 2^n.

2D molecular dynamic simulation

Finally, we studied the calculation time for both platforms, depending on the number of particles in the pore while maintaining the ratio of the density in the pores and in the outer area of 5 : 1. It was used blocks with 128 threads. The results can be seen in table 2.

N	t_{CPU}	t_{GPU}	δ
100	5.125	1.577	30.775
200	20.238	2.957	14.610
300	45.588	4.371	9.589
400	81.017	5.541	6.839
500	126.854	7.040	5.550
600	182.441	8.317	4.559
700	248.107	9.660	3.894
800	325.447	10.563	3.246
900	410.330	12.675	3.089
1000	503.784	13.706	2.721
1100	614.173	15.175	2.471
1200	727.594	17.212	2.366
1300	855.108	18.956	2.217
1400	996.005	19.767	1.985
1500	1133.698	21.522	1.898

Table 2: The calculation time in dependence on the number of particles N in the pore. Calculation time on CPU t_{CPU} and total calculation time on GPU t_{GPU} are given in ms. $\delta = \frac{t_{GPU}}{t_{CPU}} \cdot 100\%$.

2D molecular dynamic simulation

On Fig.9, we can see as it grows advantage of parallel computing when the number of particles increases.

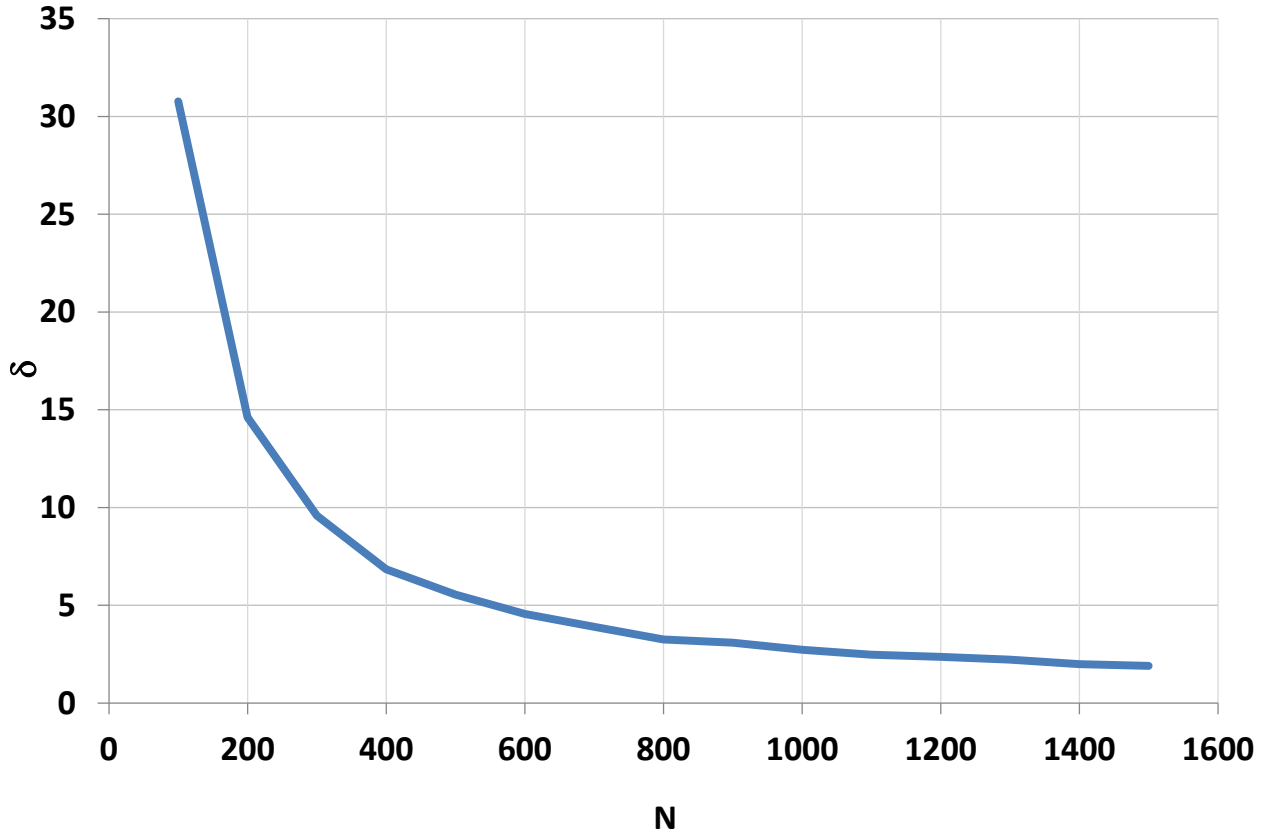

Figure 9: Comparison of the calculation time on the CPU and total GPU time depending on the number of particles in the pore expressed in percent $\delta = \frac{t_{GPU}}{t_{CPU}} \cdot 100\%$. N is the number of particles in the pore.

2D molecular dynamic simulation

Time needed for calculation on the CPU a total time on the GPU is compared on figure 10.

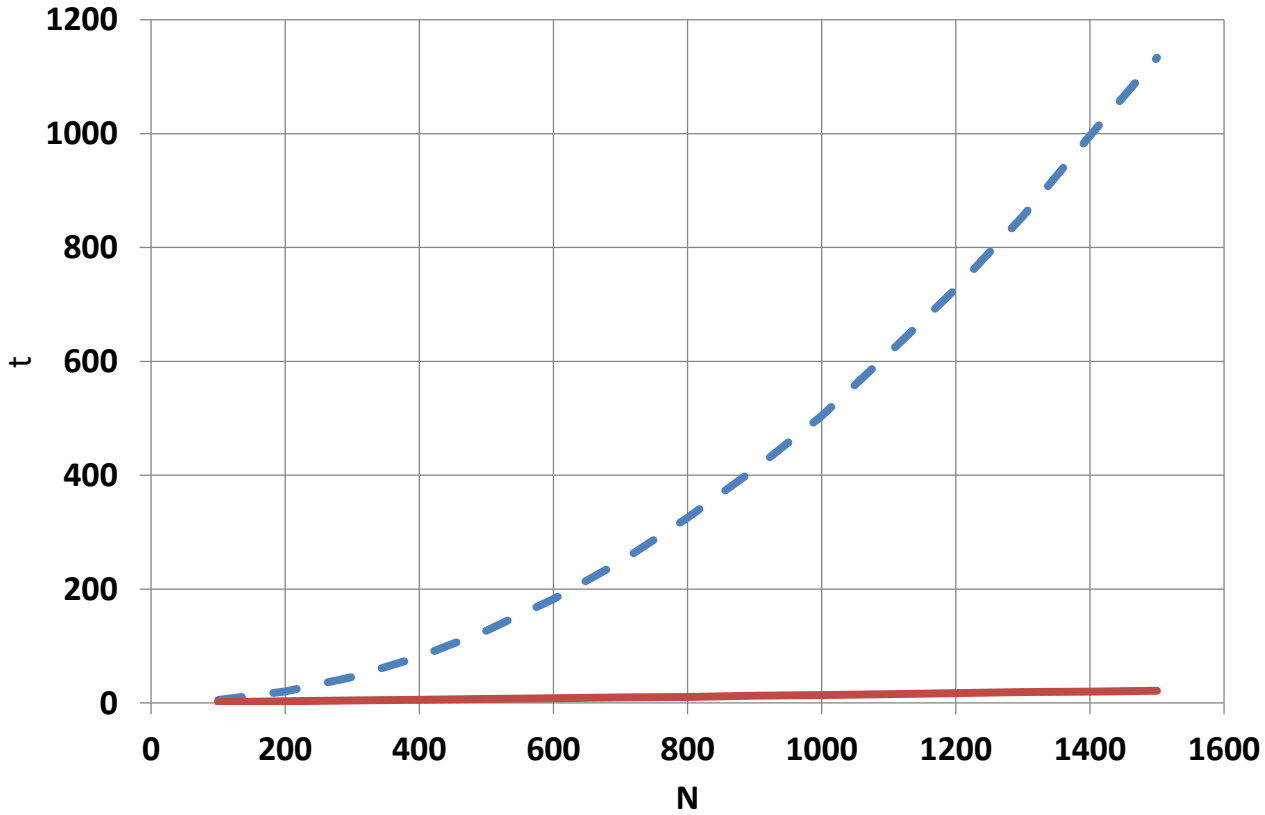

Figure 10: Comparison of computational time on the CPU and total GPU time depending on the number of particles in the pore. The dashed line is the CPU calculation time and the solid line is the GPU calculation time. N is the number of particles in the pore and t is the time in ms.

2D molecular dynamic simulation

The development of the GPU calculation time for blocks with different threads is shown on figure 11.

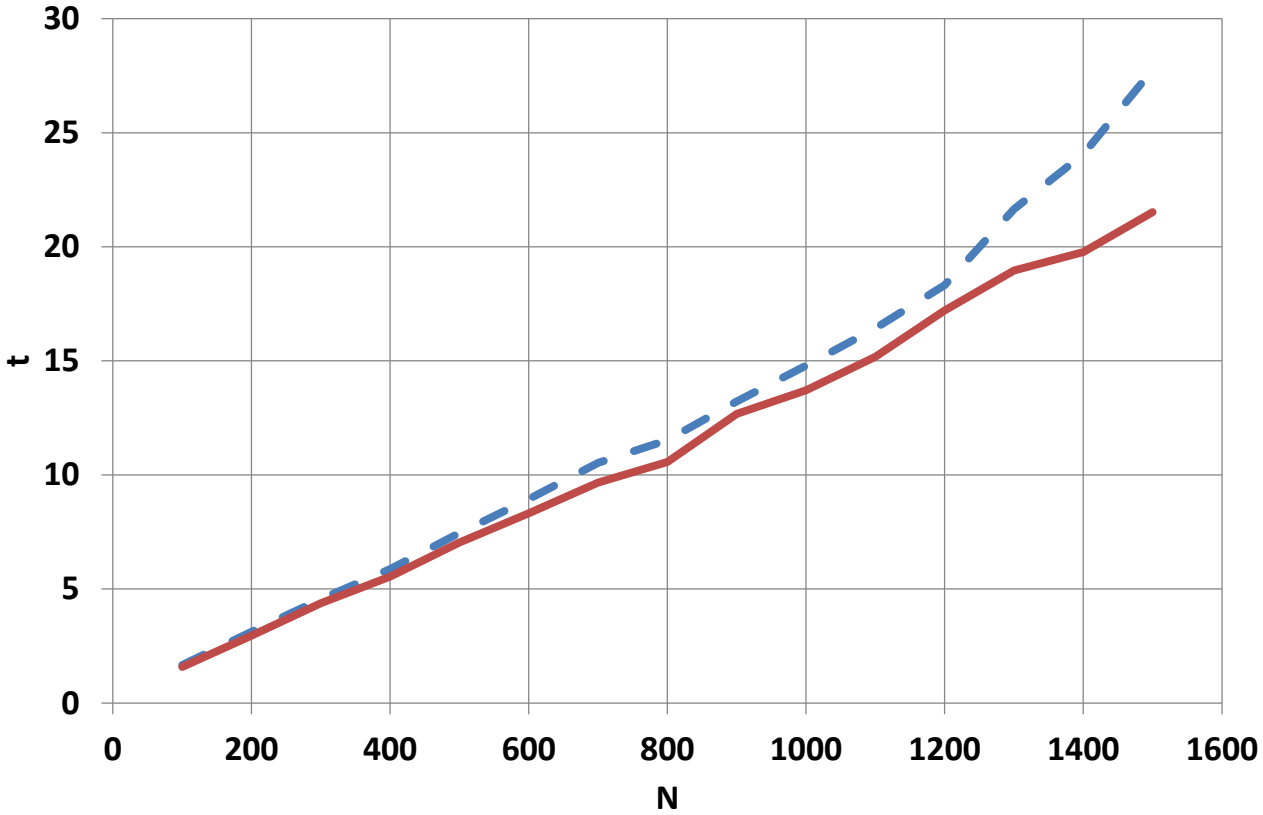

Figure 11: Total GPU calculation time for 128-threaded blocks (solid line) and 32-thread blocks (dashed line). N is the number of particles in the pore and t is the time in ms.

3D molecular dynamic simulation

In three-dimensional case[a] we made simulation for a pore in the shape of a prism of dimensions $l_x = 500$ nm, $l_y = 50$ nm, $l_z = 50$ nm. Five walls are isolated and there is no exchange of particles with outer space. The sixth wall is open. The external environment is illustrated by a prism which is 9 times bigger than the pore. The big prism satisfies periodic boundary conditions. This means that the particles which pass through one wall return to the system through the opposite wall.
Integration time step is $\Delta t = 0.016$ ps and evolution time 65.3 ns. For our purposes, we will again perform only 2000 time steps.

[a] *Nikonov E.G., Pavluš M., Popovičová M.*, Molecular dynamic simulation of water vapor interaction with blind pore of dead-end and saccate type. — 2017, — arXiv:1708.06216 [physics.flu-dyn]

3D molecular dynamic simulation

Consequently, we have considered the following input data for the drying process: There are 1000 molecules of water vapor inside the pore which form saturated water vapor at temperature 25^{o}C and pressure 3.17 *kPa*. There are 1800 molecules of water vapor in the outer area space corresponding to 20% saturated water vapor.
The simulation of this problem is solved using the **CUDA C** code according to the computational scheme on Fig. 5. Each of the 4 functions F1 - F4 is expanded to calculate the 3rd coordinate.
We first look at the dependence of the total computational time for parallel computing on the number of threads in the block, this dependence can be seen on Fig. 12. The minimum time has been reached for blocks with 64, 128 and 256 threads (n = 6, 7, 8). Again the main creator of this result was a function to calculate the potential (F1). GPU calculations were performed on average 21 times faster than CPU calculations.

3D molecular dynamic simulation

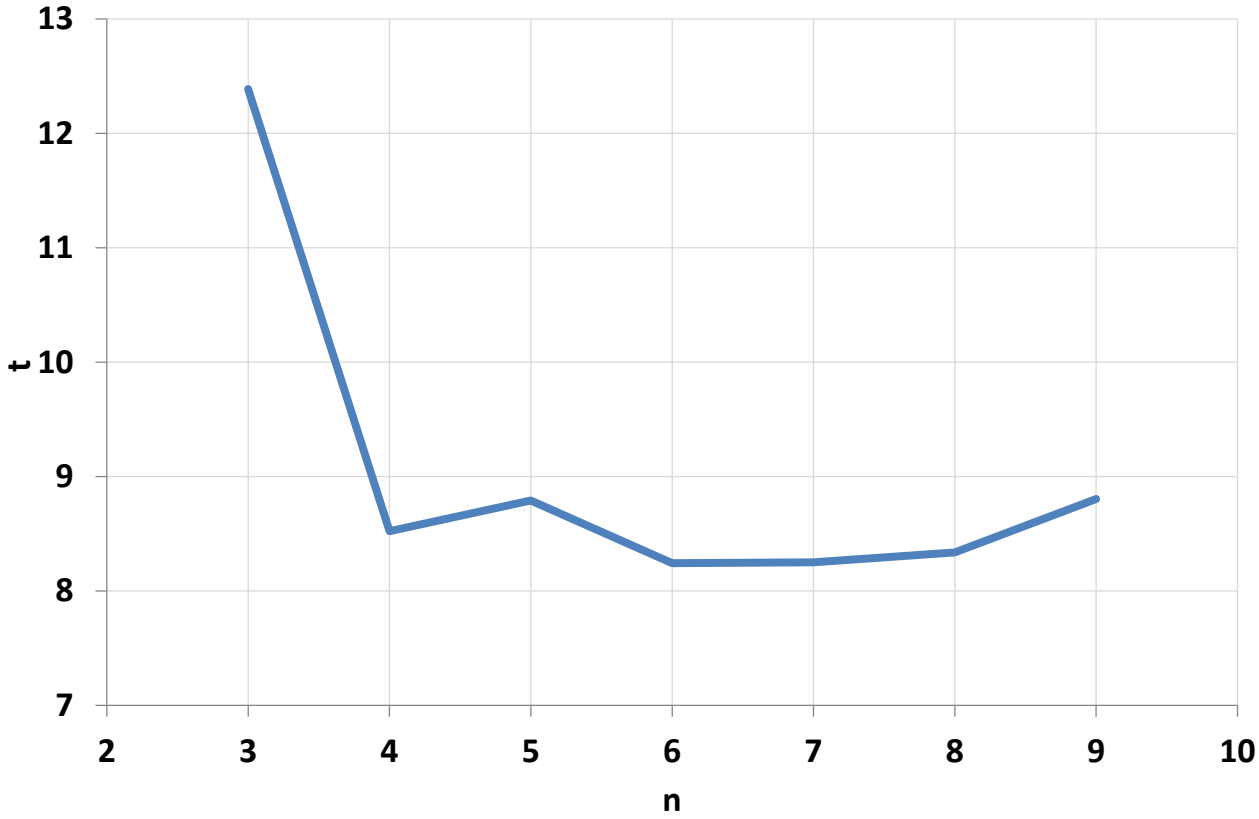

Figure 12: Dependence of the total GPU time calculation on the number of threads per block for 3D simulation. t is the time in ms. The number of threads in the block is 2^n.

3D molecular dynamic simulation

Development of computational time on the CPU is shown on Fig. 13.

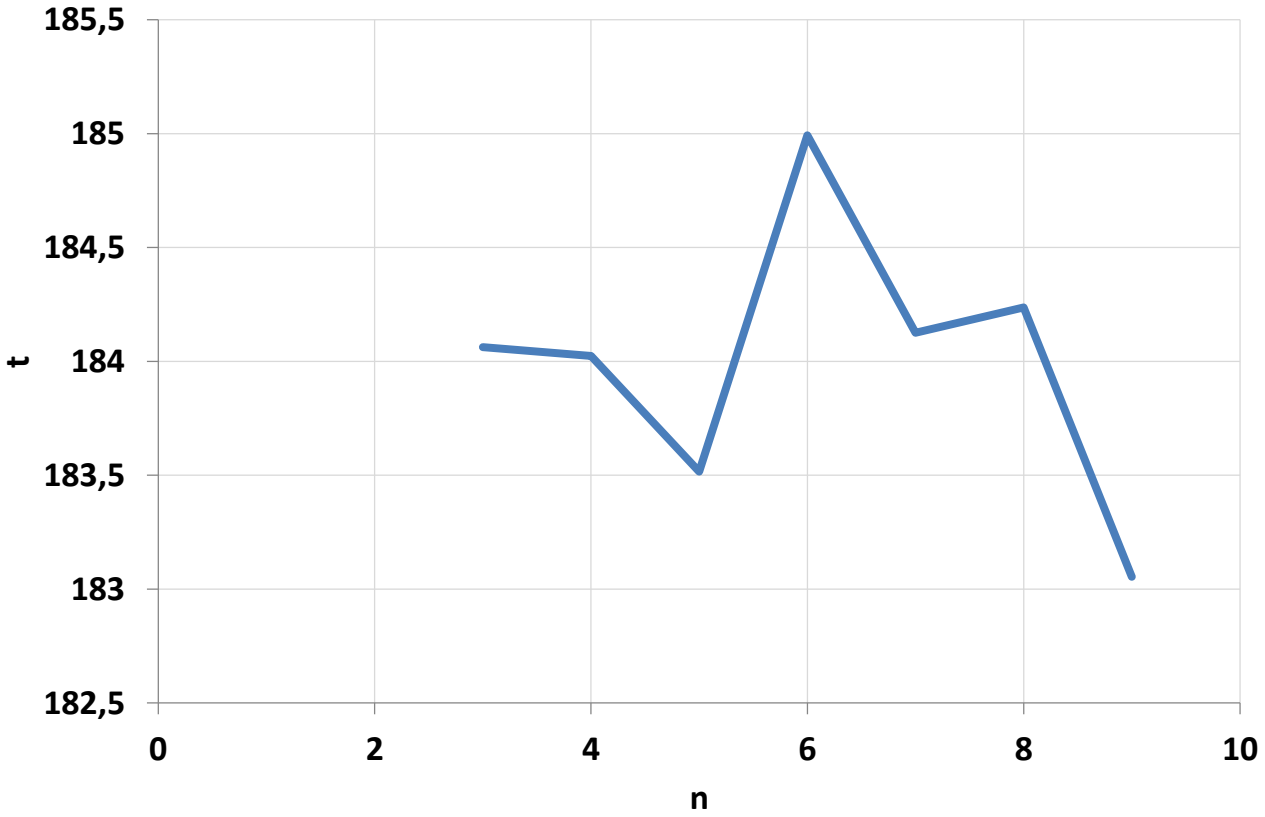

Figure 13: Development of computational time on the CPU depending on the number of threads in the block for 3D simulation. t is the time in ms. The number of threads in the block is 2^n.

3D molecular dynamic simulation

Furthermore, the calculation time of both platforms was investigated depending on the number of particles in the pore. For 3D simulation, the same ratio of particles inside the pores and in outside was maintained like for 2D simulation. 128-threaded blocks were used for the calculations. The results are shown in Table 3.

N	t_{CPU}	t_{GPU}	δ
100	1.993	1.042	52.279
200	7.417	1.776	23.944
300	16.605	2.504	15.077
400	29.462	3.374	11.453
500	46.049	4.115	8.936
600	68.389	4.829	7.061
700	90.600	5.571	6.149
800	118.330	6.162	5.207
900	150.655	7.276	4.830
1000	184.301	8.175	4.436
1100	223.147	9.035	4.049
1200	263.869	9.924	3.761
1300	313.163	10.501	3.353
1400	360.352	11.587	3.215
1500	412.590	12.434	3.014

Table 3: The calculation time in dependence on the number of particles N in the pore. Calculation time on CPU t_{CPU} and total calculation time on GPU t_{GPU} are given in ms. $\delta = \frac{t_{GPU}}{t_{CPU}} \cdot 100\%$.

3D molecular dynamic simulation

The advantage of parallel calculations for the increasing number of particles is shown on Fig. 14.

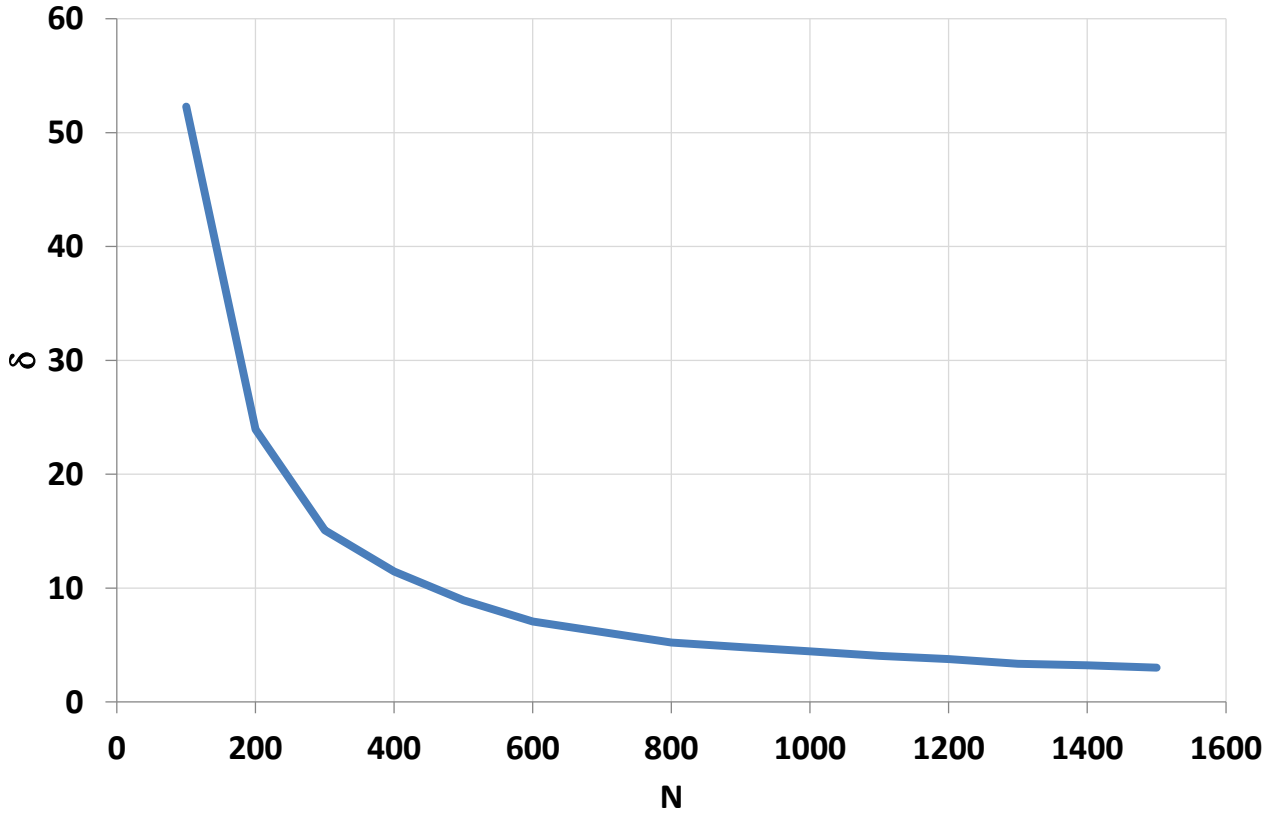

Figure 14: Comparison of the calculation time on the CPU and total GPU time depending on the number of particles in the pore expressed in percent $\delta = \frac{t_{GPU}}{t_{CPU}} \cdot 100\%$ for 3D simulation. N is the number of particles in the pore.

3D molecular dynamic simulation

Comparison of CPU time and total GPU time is depicted on Fig. 15.

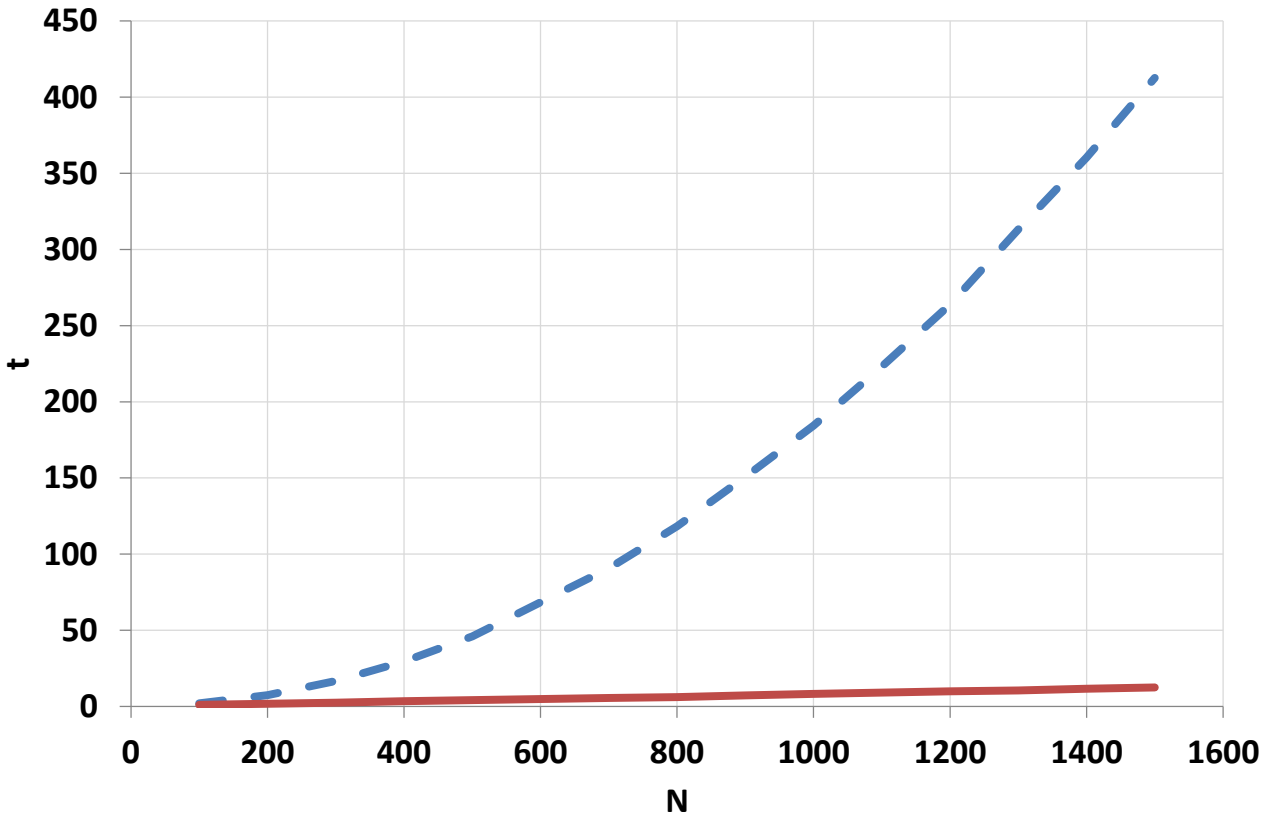

Figure 15: Comparison of computational time on the CPU and total GPU time depending on the number of particles in the pore. The dashed line is the CPU calculation time and the solid line is the GPU calculation time. N is the number of particles in the pore and t is the time in ms.

3D molecular dynamic simulation

The development of the GPU calculation time for blocks with different threads is shown on Fig. 16.

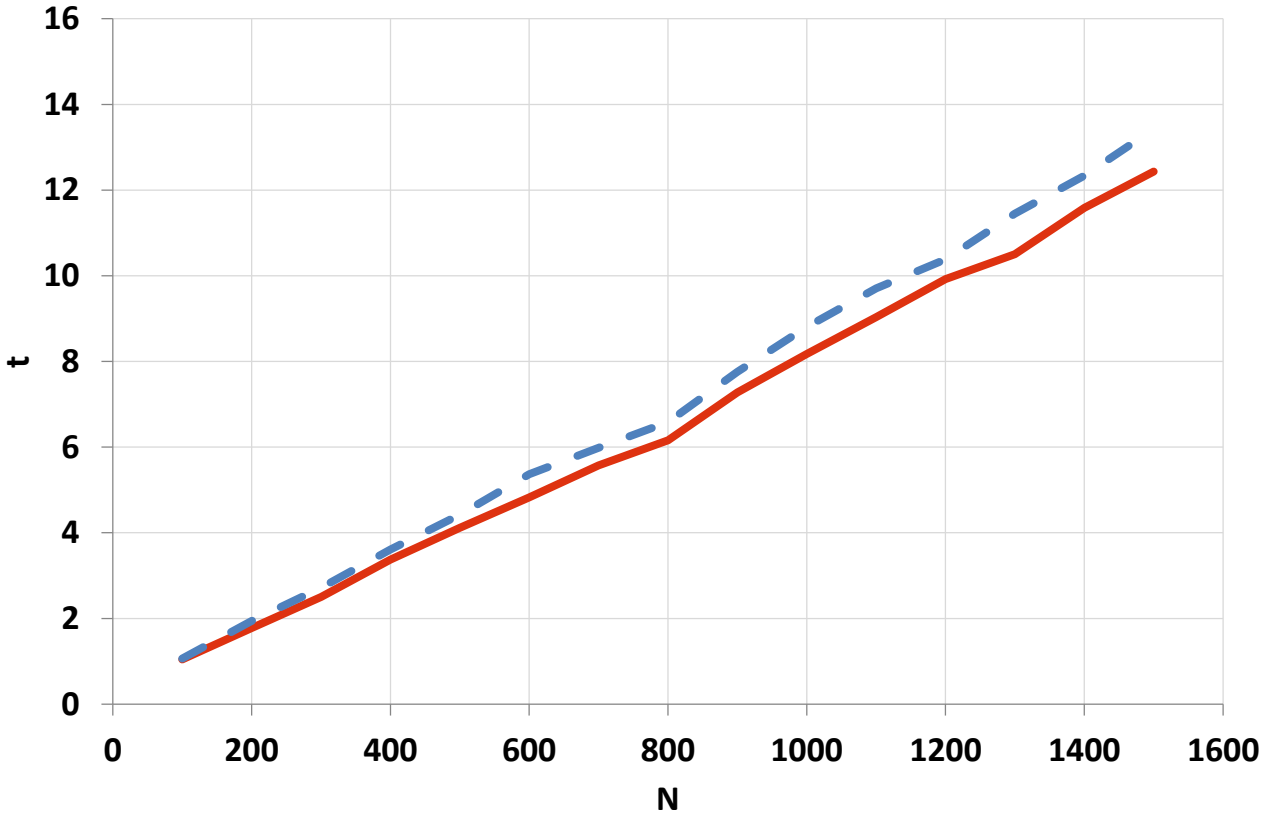

Figure 16: Total GPU calculation time for 128-threaded blocks (solid line) and 32-thread blocks (dashed line) for 3D simulation. N is the number of particles in the pore and t is the time in ms.

Comparison of 2D and 3D molecular dynamic simulations

For both simulations, the average time required to calculate one step was also calculated. In order to compare the two simulations, we have converted this time to one particle. The results are shown in Table 4. The time required to calculate one step for one particle for 2D and 3D simulation on **GPU** in both cases (pure **GPU** time and total **GPU** time) especially pure **GPU** time is 2 : 3. **CPU** time does not keep this ratio.

time	**2D**			**3D**		
	CPU	pure GPU	total GPU	CPU	pure GPU	total GPU
One Step	112.732	4.532	4.668	184.349	8.649	8.786
Per Particle	0.05162	0.00207	0.00214	0.06584	0.00309	0.00314

Table 4: Comparison of computation times per step and for one particle for both simulations.

Conclusions

As our investigations showed for both cases of 2D and 3D simulation, when paralleling the computations, there are some optimal value of number of threads in blocks such that the computation time becomes minimal in comparison with other values of this number of threads. In addition, it should be noted that, when parallelizing, the cost ratio of the computation time per particle for 2D and 3D modeling is equal 2/3 with high precision.