INFO 0939: Project 1 – Due on October 20, 2020

Updated 06/10/2020

(Intermediate deadline: October 6, 2020)

Goals of the Project

- Refresh your C language knowledge.
- Experiment with the SLURM job scheduler.
- Write your first parallel program using OpenMP.
- Implement a Brownian motion model to study the behavior of dioxygen gas.

Introduction

The kinetic theory of gases is a simple model of the behavior of gases, which has allowed to establish many important concepts in thermodynamics. Gas is described by the model as plenty of identical particles having the same mass and size, traveling at different velocities and exchanging energy only by elastic collisions between them. Under these assumptions, the motion of the particle can be represented by the laws of classical mechanics.

In this project, you will study dioxygen gas using a Brownian motion model. In particular, you will verify using a brute-force computational simulation that

- the dioxygen molecule speeds follow the Maxwell-Boltzmann distribution (see Figure 1);
- under the assumptions of the kinetic theory of gases, the dioxygen is a perfect gas;
- and the temperature is proportional to the kinetic energy of the system.

The (unknown) pressure p (in pascal) and the temperature T (in kelvin) of the gas are given by the following formulas:

$$p = \frac{Dm\langle v^2\rangle}{3}, \tag{1}$$

$$T = \frac{m\langle v^2\rangle}{3k_B}, \tag{2}$$

where D is the molecule density (assumed in this project to be fixed, with $D = 5 * 10^{27}$ molecules/m^3), m is the mass of a dioxygen molecule ($m = 5.314 * 10^{-26}$ kg), $\langle v^2\rangle$ is the average squared particle speed ($\langle v^2\rangle = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{v}_i\|^2$, with $\mathbf{v}_i$ the velocity of the particle p_i, $i = 1, \ldots, N$) and k_B is the Boltzmann constant ($k_B = 1.38064852 * 10^{-23}$ J/K).

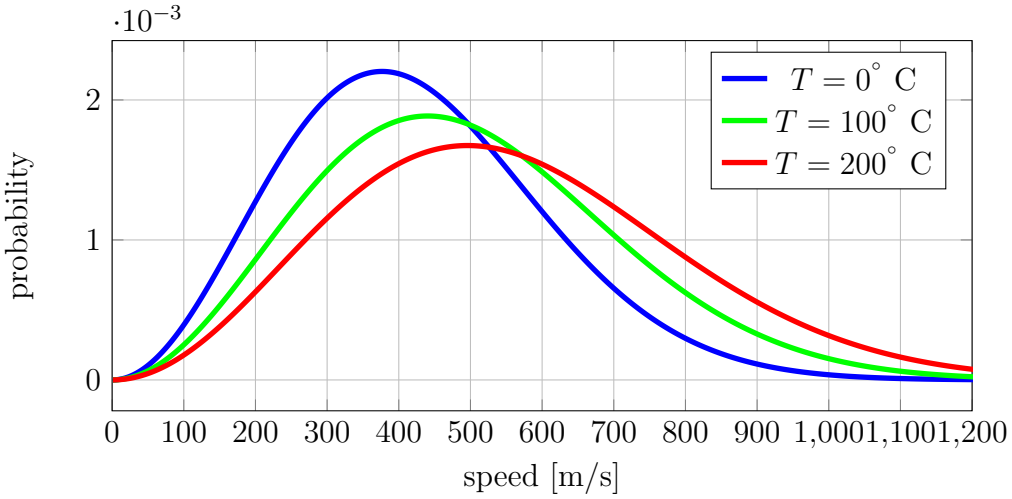

Figure 1: Maxwell-Boltzmann speed distribution of dioxygen gas at 0, 100 and 200 °C.

In order to compute the (unknown) velocities, you will simulate the time evolution of the position a (large) number of molecules in a cubic subdomain of side-length l (given in pico-meter, with 1 pm $= 1 * 10^{-12}$ m) of a gas container, given initial positions and velocities for all the particles. To simulate this time evolution, you will use the algorithm described below that updates the velocities at I time steps separated by $\Delta t = 1 * 10^{-14} s$.

The algorithm

The algorithm takes as input the size of the cubic domain l (in pico-meter), the number of time steps I, and the initial kinetic energy K of molecules (in electron-volt, eV).

- Knowing the kinetic energy K, compute the initial particle speed:

$$v_0 = \sqrt{\frac{2Kq_e}{m}}, \tag{3}$$

  where q_e is the electron electric charge ($q_e = 1.602176635 * 10^{-19}$ C) that gives the conversion between electron-volt to joule.

- Allocate memory to store the velocity and the position of N particles, with $N = D{*}l^3$. The position and the velocity are both stored as three double precision floating point numbers corresponding to the three spatial coordinates.

- Initialize the position (x_i, y_i, z_i) and the velocity $\mathbf{v}_i = (v_0 \sin(\theta_i)\cos(\phi_i), v_0 \sin(\theta_i)\sin(\phi_i), v_0 \cos(\theta_i))$ for each particle p_i, $i = 1, \ldots, N$, by considering that $x_i \in [0, l]$, $y_i \in [0, l]$, $z_i \in [0, l]$, $\theta_i \in [0, \pi]$ and $\phi_i \in [0, 2\pi]$ are (pseudo) random numbers generated by the `rand` function. It is not a problem if

particles "overlap" (see below); it is also advised to work with velocities in meter per second and positions in pico-meter in order to avoid numerical issues.

Beware that this part of the algorithm must not be made parallel because the `rand` function contains locks.

- Then for each time step, from 1 to I:
  - For each particle p_i do:
    - Search for its neighboring particles, defined as particles that are at most at a distance d. The parameter d is the kinetic diameter of the dioxygen molecule ($d = 346$ pm). Particles that overlap, i.e. for which the distance between them is smaller than ~~the square of~~ the machine epsilon (that is approximately equal to $2*10^{-16}$), are not considered as neighboring particle and are simply ignored.
    - For each neighbor particle p_j (no matter the order in which they are considered), update the velocity $\mathbf{v}_i$ of particle p_i and the velocity $\mathbf{v}_j$ of particle p_j according to:

$$\mathbf{v}_i^{\text{new}} = \mathbf{v}_i - \frac{(\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j)}{\|\mathbf{x}_i - \mathbf{x}_j\|^2}(\mathbf{x}_i - \mathbf{x}_j), \tag{4}$$

$$\mathbf{v}_j^{\text{new}} = \mathbf{v}_j - \frac{(\mathbf{v}_j - \mathbf{v}_i) \cdot (\mathbf{x}_j - \mathbf{x}_i)}{\|\mathbf{x}_j - \mathbf{x}_i\|^2}(\mathbf{x}_j - \mathbf{x}_i). \tag{5}$$

      These are the equations of a two-body elastic collision in three dimensions.
  - Once the velocity is updated for each particle, update their position using

$$\mathbf{x}_i^{\text{new}} = \mathbf{x}_i + \Delta t \, \mathbf{v}_i, \tag{6}$$

    and check that each particle remains in the box. If a particle goes out of the box, re-enter it from the opposite side.

At the end of the algorithm:

- Save the speed of each particle ($\|\mathbf{v}_i\|$) into a `CSV` file named `speed.csv` structured as described in Figure 2;
- Compute the average square speed $\langle v^2 \rangle = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{v}_i\|^2$;
- Compute the pressure p and the temperature T using (1) and (2) and print them to the screen.

$$\begin{bmatrix} \|\mathbf{v}_1\| \\ \|\mathbf{v}_2\| \\ \|\mathbf{v}_3\| \\ \|\mathbf{v}_4\| \\ \vdots \\ \|\mathbf{v}_N\| \end{bmatrix}$$

Figure 2: Structure of the output file `speed.csv`.

Instructions

Intermediate deadline

By group of **two students** (this is mandatory) you are asked to implement a C program that

1. takes from the command line the number of time steps I, the size of the box l in pico-meter and an initial particle kinetic energy K (realistic values are between 0.04 eV and 0.1 eV). A call to your program must thus look like `./brownian 10000 5000 0.04`;
2. saves the speed of each particle into a file `speed.csv`;
3. computes the pressure p and the temperature T;
4. is written in a single C file `main.c` (no header and no other source files).

Final deadline

Implement a multithreaded version of the code using OpenMP. This program should run on the CECI clusters: to this end, you should write a SLURM script that reserves a given number of cores on the supercomputer, with appropriate memory and job duration.

Write a report of maximum 5 pages where you:

1. describe your implementation;
2. study the effect of the loop scheduling on your parallel implementation;
3. analyze the efficiency (strong scaling) of your parallel implementation;
4. show that the dioxygen molecule speeds follow the Maxwell-Boltzmann distribution;
5. show that under the assumptions of the kinetic theory of gases, the dioxygen is a perfect gas, i.e. show that it follows $p = Dk_BT$;

6. show that $T = \alpha K$ (the temperature does not depend on the number of particles) with α a constant to determine.

Submit your C code on the Montefiore submission platform `https://submit.montefiore.ulg.ac.be/`. Note that no errors have to be reported during the automatic tests performed on this platform. **If your last submission generates error(s), a default grade of 0/20 will be attributed.**

When your code passes on the submission platform, send your report in PDF format together with your C code and SLURM submission script to `anthony.royer@uliege.be` and `cgeuzaine@uliege.be`. The files (report, C code, SLURM script) should be named

```
project1_Lastname1_Lastname2.pdf
project1_Lastname1_Lastname2.c
project1_Lastname1_Lastname2.sh
```

Remarks

The number of arguments and their values are passed as arguments (`argc` and `argv`) to the `main` routine: `int main(int argc, char **argv)`. For `./brownian 10000 10000 0.04`; `argc` will be 4, `argv[0]` will be `"./brownian"`, `argv[1]` will be `"10000"`, `argv[2]` will be `"10000"` and `argv[3]` will be `"0.04"`. The C library functions `atoi` and `atof` can be used to convert strings into integers and floating point values, respectively.

Make your code easily readable by other people and by "future you". This implies to:

- correctly indent your code;
- make things as simple as possible... while still being efficient;
- use pertinent, meaningful variable and function names, even if it makes them longer;
- make functions when necessary.

As usual, don't write the code all at once without testing. Write it part by part and test every small functionality separately.