Lab 4 TDDC78 – Particle Simulation

Jonathan Bosson, jonbo665

1 Program Description

This application simulates a high number of particles in a box, all with initial velocities and positions. Particles can collide with each other and the boxes' walls. The pressure is calculated from the number of collisions the particles have with the walls. Using **MPI** with multiple threads a total pressure is then calculated at the end of the simulation.

The box is divided into as many pieces as there are threads. The same number of particles are generated for each thread with a pseudo-random location and velocity. In each thread will simulate a number of time steps. Each step we check if any particle collides with any other particle in the same thread. If a collide happens does the particles interact with each other according to the laws of physics. If a particle collide with a wall will it increase the total momentum in that piece of the box. If a particle during a time step has travelled into another part of the box belonging to another thread it will be sent to that thread and removed from the particle list on the current thread.

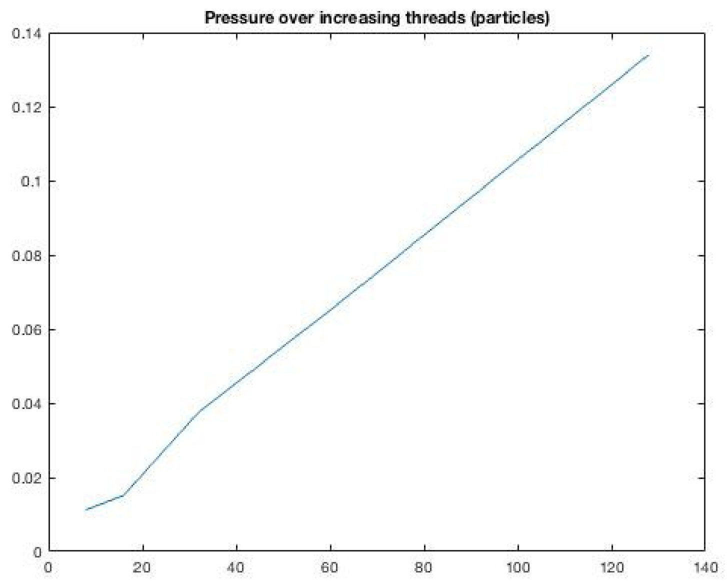
After all time steps have been completed is the **MPI_Reduce** called a couple of times to sum up the total momentum, number of sent particles and to calculate the average temperature.

2 Results

Particles sent in is equal to 500 times the number of threads used. Temperature is gotten from taking the average of the square of the randomized radius at the initial creation of particles, divided by 2 and is given in Kelvin. R is the universal gas constant that is calculated from the formula pV = nRT, ie R = pV/nT. It can be seen in the table below that once we get a high number of threads does the resulting R constant vary less and less, which implies that something is right. In a figure below it can be seen that the pressure increases linearly with the number of threads (and in turn particles).

The number of threads does not affect the execution time in a substantial way. This is since the program doesn't divide work amongst the threads but additional threads simply run the application parallel, resulting in double the amount of computation. The more threads that are used the longer the time will be since **MPI** needs more resources to send variables amongst the processors. The fact that this difference is small proves that the program is working well in a parallel environment.

Threads	8	16	32	64	128
Time	1.07983	1.29	1.2334	1.312	1.42641
Pressure (p)	$1.12 * 10^{-2}$	$1.51 * 10^{-2}$	$3.75 * 10^{-2}$	$6.90 * 10^{-2}$	$1.34 * 10^{-1}$
Area (V)	$10^4 * 10^4$	$10^4 * 10^4$	$10^4 * 10^4$	$10^4 * 10^4$	$10^4 * 10^4$
Particles (n)	4000	8000	16000	32000	64000
Temperature (T)	399.0	365.8	407.5	390.43	393.61
R	0.7035	0.5164	0.5756	0.5523	0.5336



Below is a table of how the number of sent particles amongst the threads relates to the number of threads. Although it varies in each execution, we can see that it relates very closely to the formula 62.5 * numThreads * 2^{k-3}, or 62.5 * 2^{2k-3} where the number of processors is equal to 2^k. This is specially clear with a higher number of active threads. From this we can draw the conclusion that we don't want more than 32000 used particles per thread, since after that there are more sent particles than particles used per thread.

k	0	1	2	3	4	5	6	7
Threads	1	2	4	8	16	32	64	128
Sent Particles	0	23	117	462	2015	7993	32771	135913
Estimation	0	31.25	125	500	2000	8000	32000	128000