P. Koumoutsakos
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2018

Set 12 – Particle Strength Exchange, Cell Lists

Issued: December 14, 2018
Hand in (optional): December 24, 2018 23:59

Question 1: 2D Diffusion using Particle Strength Exchange (PSE)

Consider a scalar two-dimensional field $\phi(x, y, t)$ defined on a periodic domain $(x, y) \in [0, 1)^2$. We want to utilize the PSE method to solve the diffusion problem:

$$\frac{\partial \phi}{\partial t} = \nu \left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \right), \tag{1}$$

for some given initial condition $\phi(x, y, t = 0)$ and diffusion constant ν.

Instead of discretizing the field ϕ on a grid, we will use a collection of N *particles*. A particle represents a small "volume" of the field, and is defined by its position $\mathbf{x}_i$ and field value $\phi_i = \phi_i(t)$. In this exercise we assume the volume of each particle is equal ($V_i = V_{\text{total}}/N = 1/N$).

We rewrite Eq. (1) as a system of equations on particles:

$$\frac{\mathrm{d}\phi_i}{\mathrm{d}t} = \frac{\nu}{\varepsilon^2} \sum_j V_j \left(\phi_j - \phi_i\right) \eta_\varepsilon(\mathbf{x}_j - \mathbf{x}_i), \tag{2}$$

where $\eta_\varepsilon(\mathbf{r})$ is a kernel representing the laplacian operator, and ε a scale constant. In this exercise we consider the following kernel:

$$\eta_\varepsilon(\mathbf{r}) = \frac{1}{\varepsilon^2} \eta\left(\mathbf{r}/\varepsilon\right), \qquad \eta(\mathbf{r}) = \frac{4}{\pi} e^{-|\mathbf{r}|^2}. \tag{3}$$

a) You are given a skeleton code that initializes the particle positions and values. As a placeholder for the final timestep, the skeleton code simply decays the values ϕ_i in time. Get familiar with the skeleton code. Use `make run` and `make plot` to run the code and test the plotting script.

   Note: the plotting script requires numpy, matplotlib and ffmpeg.

b) Extend the provided skeleton code to compute the right-hand side of Eq. (2) using Eq. (3) for the kernel η_ε. Account for a periodic domain when computing the distance between two particles!

c) Implement the forward Euler scheme for the integration of the Eq. (2).

d) Considering the domain size and the number of particles, what is qualitatively the distance h between neighboring particles? Parameter ε determines the spread of the kernel. Run the code for different values of ε. Experiment with $\varepsilon \ll h$, $\varepsilon \approx h$ and $\varepsilon \gg h$. What do you observe for different cases?

Question 2: Cell lists (optional)

Cell lists are a data structure used to speed up the computation of short-range pairwise interactions between particles. In Question 1 the implementation of Eqs. (2) and (3) required $O(N^2)$ operations each timestep. Cell lists enable us to optimize the complexity to $O(N)$.

First, notice that the exponent in $\eta_\varepsilon(\mathbf{r})$ quickly vanishes as $|\mathbf{r}|$ is increased. For $\varepsilon = 0.05$ and $|\mathbf{r}| = 0.30$, we have $\eta_\varepsilon(\mathbf{r}) \approx 3 \times 10^{-16}$, which is close to the machine precision. Thus, if we set a distance threshold of $\Delta = 6\varepsilon$ and consider only interactions for pairs where $|\mathbf{x}_i - \mathbf{x}_j| < \Delta$, the result we get is virtually the same. We can even sacrifice a bit of accuracy and choose a threshold of $\Delta = 4\varepsilon$, such that $\eta_\varepsilon(\Delta) \approx 1.4 \times 10^{-7}$.

Now when we replace the long-range interactions with short-range interactions, in order to get the performance improvement, we need to be able to efficiently find all pairs of nearby particles. This is where cell lists come into play.

The idea is to split the domain into square cells whose side length is equal to Δ. For example, for $\varepsilon = 0.05$ and $\Delta = 4\varepsilon = 0.20$, the domain would be split into 25 cells. Because the cell size is exactly Δ, a particle i belonging to a cell k can only interact with particles of cell k and 8 neighboring cells (in 3D a cell would have 26 neighbors).

This way we reduced the number of (ordered) pairs we consider from N^2 to $9NK$, where K is the number of particles per cell. Thus, the complexity is now linear with the size of the problem[1]!

Your goal is to implement cell lists.

a) First, implement the data structure itself. Instead of storing all particles in one array, create a matrix of cells, and redistribute the particles into those cells.

b) Implement the traversal of each pair of nearby particles, i.e. all particles whose distance is less than Δ. In other words, implement a traversal over each particle i and each nearby particle j. Do not forget about periodic boundaries!

c) Test your code and compare the performance with the old $O(N^2)$ implementation. Experiment with increased size of the domain (or effectively, larger N and smaller ε).

d) If the particles are allowed to move in time (as in the case of Molecular Dynamics), they will be occasionally crossing from one cell to another. How would you implement that operation? Concretely, how do you insert a particle to a cell, and how do you remove it? Can you guarantee $O(1)$ complexity for both? Keep in mind that at some point you might want to utilize vectorization, so you need to keep the storage simple.

Question 3: Brain teasers (optional)

In this question we propose a few general problems and ask you to think of some good solutions.

a) We introduced cell lists as a data structure used to improve the performance of local interaction kernels. However, our appetite has grown and now we want to extend the capabilities of our simulations to large gravitational systems. Gravity is a long-range interaction, with force proportional to $1/r^2$, where r is the distance between two mass particles.

[1] Value K is determined by the nature of the problem (interaction range) and numerics (ε and h), but it is independent of the size of the domain. We therefore say that cell lists optimize $O(N^2)$ to $O(N)$ complexity, even though the underlying constant is large.

Think of a way to optimize this kind of kernel. Hint: think like an engineer, not as a mathematician.

b) Assume the pairwise interaction (the kernel) is expensive to compute, and that it is symmetric with respect to the order of particles. The exponential in Eq. (3) is a good example. The symmetry allows us to reduce the number of kernel computations by one half – when we compute the interaction effect of a particle A on the particle B, we get for free the effect of B on A.

   How would you incorporate this symmetry into cell lists? How does the interaction calculation algorithm look like?

c) The machines we are running our codes on have multiple CPU cores, and it would be a shame not to take advantage of that.

   How would you parallelize the computation of interactions based on cell lists? Of course, ensure you don't have a race condition. Ignore for the moment the symmetry trick from the previous subquestion.

   Now, what if we want to take advantage of all three ideas: cell lists, symmetry and parallelization? What are the problems now? Atomics and locks are one solution. Can you think of a solution that avoids atomics? Consider first the 1D case and generalize to 2D/3D.