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

Fall semester 2021

Set 4 - Diffusion, PSE & OpenMP

Issued: November 12, 2021
Hand in (optional): November 26, 2021 12:00

Question 1: Diffusion (25 points)

The diffusion of a substance can be described by the equation

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

where c is the concentration of the substance at position (x, y) and at time t, and D is the diffusion constant. The diffusion process happens in the domain $|x| < L/2$ and $|y| < L/2$. The concentration is zero on the boundaries of the domain for $t \geq 0$. The initial concentration is

$$c(x,y,0) = \begin{cases} 1, & \text{if } |x| < L/4 \text{ and } |y| < L/4, \\ 0, & \text{otherwise.} \end{cases}$$

a) Write down the 2-dimensional discretized diffusion process for eq. (1). Assume a uniform grid with spacing h and a central finite difference scheme in space and forward Euler time integration. For the forward Euler integration assume time intervals of size dt. Make sure that you annotate all variables.

The corresponding discretized equation is:

$$c_{i,j}^{t+1} = c_{i,j}^{t} + D\frac{dt}{h^2}(c_{i+1,j}^{t} + c_{i-1,j}^{t} + c_{i,j+1}^{t} + c_{i,j-1}^{t} - 4c_{i,j}^{t}) \tag{2}$$

where $c_{i,j}^{t} = c(i*h, j*h, t)$ is the concentration at time t for $x = i*h$ and $y = j*h$.

Grading scheme:

- 5p: Correct final equation
- -1p: If annotations missing (e.g. $c_{i,j}^{t} = c(i*h, j*h, t)$)

Total: 5p

In the following subquestions, you will work with the codes provided in `/ex04/skeleton_code/q1/`. Please have a look at the README file for further information. We recommend compiling and running the code on the Euler cluster https://scicomp.ethz.ch/wiki/Main_Page.

b) Based on the discretization found in the previous subquestion, find the maximal timestep dt using the von Neumann stability analysis. Replace the hardcoded timestep ($t = 0.0001$) in the code with your solution.

We assume a solution of the form $c_{i,j,t} = \rho^t e^{ikx_i} e^{ikx_j} = \rho^t e^{ik(x_i+x_j)}$ and apply this to the discretization

$$\begin{aligned}\rho^{t+1} e^{ik(x_i+x_j)} &= \rho^t e^{ik(x_i+x_j)} + D\frac{dt}{h^2}\rho^t(e^{ik(x_i+h+x_j)} + e^{ik(x_i-h+x_j)} + e^{ik(x_i+x_j+h)} + e^{ik(x_i+x_j-h)} - 4e^{ik(x_i+x_j)}) \\ &= \rho^t e^{ik(x_i+x_j)} + D\frac{dt}{h^2}\rho^t(2e^{ik(x_i+h+x_j)} + 2e^{ik(x_i-h+x_j)} - 4e^{ik(x_i+x_j)}) \ .\end{aligned}$$

Dividing both sides by $\rho^t e^{ik(x_i+x_j)}$ gives

$$\rho = 1 + \frac{Ddt}{h^2}(4\cos(kh) - 4) \ .$$

For stability we must have $|\rho| \leq 1$. This yields the upper bound

$$dt_{max} = \frac{h^2}{4D}$$

The implementation can be found in `solution_code/q1/diffusion.cpp`.

Grading scheme:
Inside `advance`:

- 2p: Correct initial assumption and replacement.
- 1p: Solving for ρ.
- 1p: Finding ρ_{max}.
- 1p: Implementation.

Total: 5p

c) Based on the discretization found in the first subquestion, provide a cache-friendly implementation of the diffusion equation in the method `advance`. I.e. avoid copying of memory if possible and mind the access patterns of the memory. Blocking must not be implemented.

The implementation can be found in `solution_code/q1/diffusion.cpp`.

Grading scheme:
Inside `advance`:

- 3p: Correct implementation of diffusion
- 1p: Efficient memory access scheme inside for loops in order to benefit from data locality.
- 1p: Avoid copying memory of `c_tmp` into `c` but instead use `std::swap` or similar.

Total: 5p

d) Plot the total concentration as a function of time for $t \in [0, 0.5]$ using $D = 1$, $L = 2$ and $N = 100$. The concentration can be read from the file `diagnostics.dat` (column 0 and 1). Qualitatively explain the behaviour of the graph in less than 3 sentences, is this result expected?

An exemplary plotting script can be found in `solution_code/q1/plotdiagnostics.py`. The total concentration decreases over time because of the implied boundary conditions. The concentration outside the domain is zero, and since the concentration diffuses from high to low, the particles exit the domain of interest.

Grading scheme:
Plotting:

- 2p: Correct behaviour of graph, graph visible for $t \in [0, 0.5]$, x-axis and y-axis labelled, x and y-axis ticks visible and value range clear.
- -1p: Deduct one point if plot not complete (e.g. range is cut-off, or labels missing, or value range not clear).

Explanation:

- 1p: For correct argument.

Total: 3p

e) Parallelize the diffusion process (your implementation from subquestion 1c) in the method `advance` using OpenMP.

The implementation can be found in `solution_code/q1/diffusion.cpp`.

Grading scheme:
Inside `advance`:

- 2p: `#pragma omp parallel for` around outer loop (or similar). Consider cache-friendliness of your parallelizaition approach.

Total: 2p

f) Parallelize the integration of the concentration (marked with `TODO` in `compute_diagnostics`) and the calculation of the histogram (marked with `TODO` in the method `compute_histogram`) using OpenMP.

The implementation can be found in `solution_code/q1/diffusion.cpp`.

Grading scheme:
Inside `compute_diagnostics`:

- 1p: Correct reduction of variable `amount` (e.g. `#pragma omp parallel for reduction(+:amount)`, or manual reduction with critical section, or similar).

Inside `compute_histogram`:

- 2p: Correct reduction of variables `max_c` and `min_c` (e.g. `#pragma omp parallel for reduction(m`
or manual reduction with critical section, or similar).
- 2p: Fully parallel histogram computation (e.g with local histogram computation and parallel reduction of local histograms, or with a user declared reduction operator on the histogram vector, or similar).

Total:5p

Question 2: Particle Strength Exchange (20 points)

Consider the diffusion equation eq. (1) from the previous exercise. We want to utilize the particle strength exchange (PSE[1]) method to solve this diffusion problem. Instead of discretizing the field $c(x, y, t)$ 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 $\phi_i = \pi_i(t)$. In this exercise, we assume the volume of each particle is equal $V_i = V_{total}/N = L^2/N$. We rewrite eq. (1) as a system of equations on particles:

$$\frac{\partial \phi_i}{\partial t} = \frac{D}{\epsilon^2} \sum_{j=1}^{N} V_j(\phi_j - \phi_i)\eta_\epsilon(x_j - x_i)\,, \tag{3}$$

where $\eta_\epsilon(r)$ is a kernel representing the Laplacian operator, and ϵ a scale constant. In this exercise we consider the following kernel:

$$\eta_\epsilon(\mathbf{r}) = \frac{4}{\epsilon^2 \pi} e^{-\frac{1}{\epsilon^2}|r|^2}. \tag{4}$$

Assume the same initial and boundary conditions as in the previous exercise.

In the following subquestions, you will work with the codes provided in `/ex04/skeleton_code/q2/`. Please have a look at the README file for further information. We recommend compiling and running the code on the Euler cluster.

a) You are given a skeleton code that initializes the particle positions and values. Get familiar with the skeleton code. Use `make run` and `make plot` to run the code and the scripts.

No solution. **Total: 0p**

b) Extend the provided skeleton code to compute the right-hand side of eq. (3) using eq. (4) for the kernel η_ϵ. Reuse pair-wise quantities and reduce the number of operations. Implement the forward Euler scheme for the integration of the eq. (3).

The implementation can be found in `solution_code/q2/pse2d.cpp`. The forward euler scheme is implemented as:

$$\phi_i \leftarrow \phi_i + \delta t \cdot RHS_i.$$

Note that all RHS_i must be computed before updating any ϕ_i and that the particles on the boundaries are not updated.

Grading scheme:

- 2p: Correct Gaussian kernel.
- 2p: Correct nested for-loops.
- -1p: N^2 operations, not reusing calculations.
- 1p: Correct Euler scheme.
- -1p: If particles on boundaries updated.

Total: 5p

c) 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

[1] see lecture website for supplementary information on PSE

- $\epsilon << h$,
- $\epsilon \approx h$,
- and $\epsilon >> h$.

What do you observe for different cases? You can visualize the output of your runs with `make plot`.

The distance h is approximately $h \approx L/\sqrt{N}$ as we distribute $\sqrt{N}$ particles uniformly along each dimension. If $\epsilon << h$, the kernel fades out too quickly and there is no interaction between the particles. If $\epsilon >> h$, the kernel is so wide that it does not even differentiate particles according to their distance. In practice it means that all particles initially set to $\phi_i = 0$ will increase at the same rate, independent on their position, and all particles initially set to $\phi_i = 1$ will decrease at the same rate. The case $\epsilon \approx h$ shows the desired behavior of the diffusion kernel.

Grading scheme:

- 2p: Correct estimate h
- 3p: Correct argumentation (1p each)

Total: 5p

d) Parallelize the `timestep` method using OpenMP. For this, you need to modify the `Makefile` and include the library in your source code.

The implementation can be found in `solution_code/q2/pse2d.cpp`. Grading scheme:

- 1p: Correct makefile adaption, correct include
- 4p: Correct #pragma omp parallel for (2 times, 2p each),
- -1p: If atomic operation missing in inner loop (and pair-wise calculation implemented).

Total: 5p

e) Measure and plot the runtimes of your programs (diffusion and PSE) using 1...12 threads. For benchmarking on the Euler cluster, you can run `make stat` in an interactive shell or launch a job with `make statjob` (or `make statjobfull`). For plotting the runtimes run `make plotstat`. Answer the following: Which code (diffusion or PSE) scales better and how do you measure that? Do you consider strong or weak-scaling?

We consider strong-scaling as the number of processes are increased whilst the problem size remains constant. The PSE program scales better, although its slower (which is irrelevant, the programs are not even solving the same problem). For the PSE code we observe a speedup > 4 for 12 threads, whereas in the diffusion code we observe a speedup of ≈ 2.5. Note that your answer may vary based on your implementation, the compiler, hardware and if other processes were running on the same node.

Grading scheme:

- 2p for the plots (1p each)
- 1p strong-scaling
- 1p correct observation
- 1p justification

Total: 5p

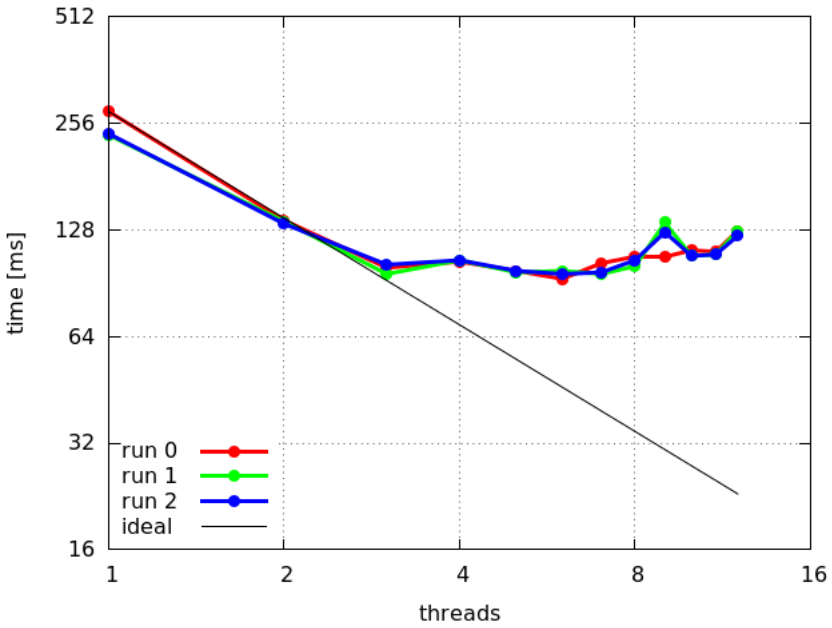

Figure 1: Runtimes of diffusion code (Q1) using 1..12 threads. Code ran on Euler using 12 cores.

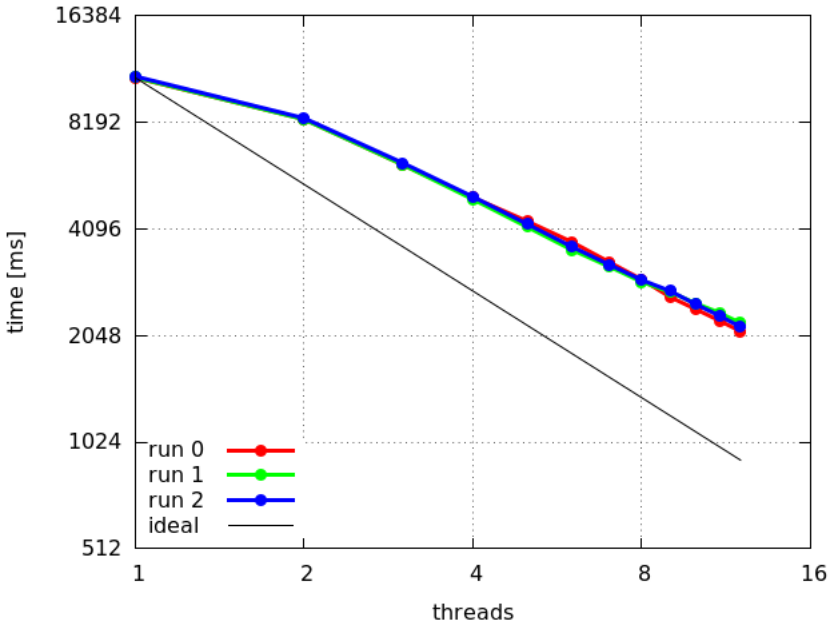

Figure 2: Runtimes of PSE code (Q2) using 1..12 threads. Code ran on Euler using 12 cores.