P. Hadjidoukas, E. Dermatas, E. Gallopoulos

Winter semester 2022-23

Set 1 - MPI and PDEs

Issued: October 27, 2022

Heat flow in a medium can be described by the diffusion equation

$$\frac{\partial \rho(\boldsymbol{r},t)}{\partial t} = D\nabla^2 \rho(\boldsymbol{r},t), \tag{1}$$

where $\rho(\boldsymbol{r},t)$ is a measure for the amount of heat at position $\boldsymbol{r}$ and time t and the diffusion coefficient D is constant.

Let's define the domain Ω in two dimensions as $\{x,y\} \in [-1,1]^2$. Equation 1 then becomes

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

Equation 2 can be discretized with a central finite difference scheme in space and explicit Euler in time to yield:

$$\frac{\rho_{r,s}^{(n+1)} - \rho_{r,s}^{(n)}}{\delta t} = D\left(\frac{\rho_{r-1,s}^{(n)} - 2\rho_{r,s}^{(n)} + \rho_{r+1,s}^{(n)}}{\delta x^2} + \frac{\rho_{r,s-1}^{(n)} - 2\rho_{r,s}^{(n)} + \rho_{r,s-1}^{(n)}}{\delta y^2}\right) \tag{3}$$

where $\rho_{r,s}^{(n)} = \rho(-1 + r\delta x, -1 + s\delta y, n\delta t)$ and $\delta x = \frac{2}{N-1}$, $\delta y = \frac{2}{M-1}$ for a domain discretized with $N \times M$ gridpoints.

We use open boundary conditions

$$\rho(x,y,t) = 0 \quad \forall\, t \geq 0 \,\text{and}\, (x,y) \notin \Omega \tag{4}$$

and an initial density distribution

$$\rho(x,y,0) = \begin{cases} 1 & |x,y| < 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

An OpenMP implementation of the 2D diffusion equation is provided in `pde/diffusion2d_omp.c`. The parallelization is applied to the routines that initialize and advance the system. The `compute_diagnostics` routine computes an approximation to the integral of ρ over the entire domain.

An MPI implementation of the same problem can be found in `pde/diffusion2d_mpi.c`. The code decomposes the domain using tiling decomposition scheme (described in the lecture notes), i.e. distributes the rows evenly to the MPI processes. Moreover, `pde/diffusion2d_mpi_nb.c` provides an equivalent implementation that uses non-blocking communication to achieve overlap

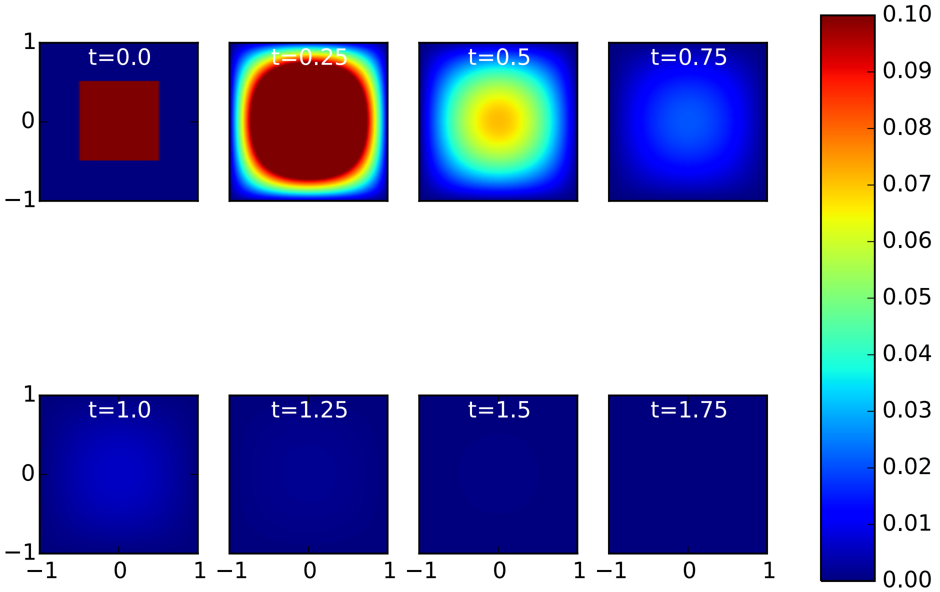

Figure 1: Time evolution of the system for a grid of 128x128 points.

of data transfer with computation. The MPI version of the `compute_diagnostics` routine is based on the MPI reduction operation.

A indicative series of snapshots of the simulation in time are shown in the figure.

The provided MPI solutions implement a 2D domain decomposition that divides the square domain of size NxN into P rectangular tiles (stripes) along the $y-$ direction, with one tile per process. Then the maximum number of elements that need to be communicated between neighboring processes is $2N$, since two boundary rows of grid-points need to be communicated to two neighboring processes. This applies for all tiles that are not on the domain boundaries of the domain (since in our exercise, the boundary conditions are not periodic so data do not need to be communicated at the domain boundaries). For the boundary tiles the message communication size is N since only one row of data needs to be communicated.

Another approach is to split the grid into square tiles with one tile per process. The total amount of communication per process if the domain is divided into square tiles is $2[N_x/P_x + N_y/P_y]$ where $N_{x,y}$ and $P_{x,y}$ is the number of grid points and processes in each direction. As we are doing sparse matrix-vector multiplication the computation scales as N_xN_y/P_xP_y. If we keep the total grid size N fixed and increase the total number of processes P the computational load per process will decrease as $1/P$. The communication will stay constant for rectangular tiles, but will decrease as $1/\sqrt{P}$ for square tiles. Hence finite difference discretization should scale much better if we use square tiles rather than rectangular.

Question 1: 2D Diffusion and MPI

a) Become familiar with the provided parallel implementations of the 2D diffusion equation.

b) Implement the domain decomposition that uses square tiles. Assume that the number of MPI processes divide evenly the domain. Verify that the implementation is correct by comparing your results with those generated by the provided codes. You can use any of the two MPI implementations that are available for the rectangular tile based domain decomposition (i.e. with blocking or non-blocking communication).

c) Report the execution time for your implementation on 1 and 4 processors, for $D = 1$, $L = 1$, $T = 1000$, $dt = 1e - 9$, for $N = 1024$, $N = 2048$ and $N = 4096$. For benchmarking, you must build the execution with `make perf=1`, to disable the computation of diagnostics. You are allowed to perform any further optimizations to the code.

Question 2: Checkpointing with MPI I/O

In the provided skeleton code `mpiio/diffusion2d_mpi_nb_io.cpp`, you find a slightly extended version of the MPI code for the 2D diffusion problem. The main function includes a checkpoint and restart mechanism, based on the `write_density_mpi()` and `read_density_mpi()` routines. Use MPI I/O in both routines to generate a single **binary** file for all density values and then read it in order to restart the simulation. The code uses the initial (striped based) domain decomposition, where each process works on a subset of the rows.

Hints:

- You do not need to store the two rows that contain the ghost data of each rank.
- You can initially implement the mechanism for a single MPI process and regular I/O (binary or not).
- Given that you write only doubles in the output file, you can use a binary reader or the printdata.sh script to check your implementation. Adjust the problem size and the number of ranks to simplify your checks.
- Verify your solution for larger number of processes by comparing the output binary file with that generated by the single process experiment (but for the same problem size and number of steps).