Heat equation in 2D

Heat diffusion

Heat flows in medium according to local temperature gradient by diffusing in to a certain equilibrium state. Such process is accurately represented with a partial differential equation for the temperature field. In it, the rate of change in time t of the temperature field $u = u(x, y, t)$, over two spatial dimensions x and y, is governed by

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u$$

where α is the constant heat conduction coefficient of the medium and

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

is the Laplacian partial differential operator.

Rewriting the above equation we have

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right).$$

This equation is well approximated by replacing the derivatives with finite difference discretizations. This way the temperature is described only at finite number of locations in space and time, which is desirable from the perspective of available finite memory in modern computers.

In this mini-app we will discretize all three dimensions x, y, and t, such that

$$\begin{aligned} \nabla^2 u = & \frac{u(i-1,j) - 2u(i,j) + u(i+1,j)}{(\Delta x)^2} \\ & + \frac{u(i,j-1) - 2u(i,j) + u(i,j+1)}{(\Delta y)^2} \end{aligned}$$

where $u(i, j)$ refers to the temperature at location with integer index i within the domain of x spaced by Δx and location with integer index j within the domain of y spaced by Δy.

Given an initial condition ($u(t = 0) = u^0$), one can follow the time dependence of the temperature field from state m to $m + 1$ over regular time steps Δt with explicit time evolution method:

$$u^{m+1}(i,j) = u^m(i,j) + \Delta t \alpha \nabla^2 u^m(i,j)$$

Note: The algorithm is stable only when

$$\Delta t < \frac{1}{2\alpha} \frac{(\Delta x \Delta y)^2}{(\Delta x)^2 (\Delta y)^2}$$

This equation expresses that the time evolution of the temperature field at a particular location depends on the value of the field at the previous step at the same location *and* four adjacent locations:

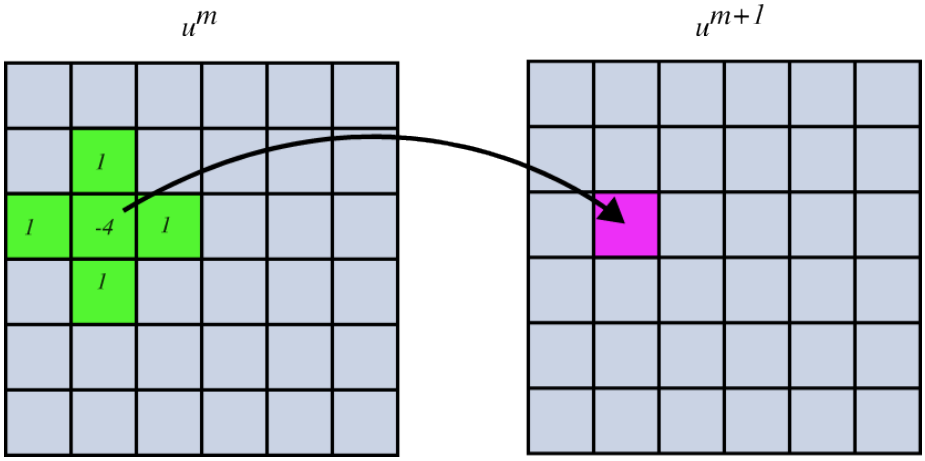

Figure 1: Heat distribution is updated from 5 cell indices (green) to the center of the cells (purple). Each cell in the grid corresponds to a (i, j) -combination.

Parallelization

The problem can be parallelized by diving the two dimensional temperature field to different workers, i.e. doing domain decomposition. With shared memory computers the parallelization is relatively straightforward, however, with distributed memory parallelization MPI some extra steps are needed.

We focus here on one dimensional decomposition. Due to different memory layout in Fortran and C/C++, the grid is divided into blocks of columns in Fortran or and into rows in C/C++ and each block is assigned to one MPI task.

The MPI tasks are able to update the grid independently everywhere else than on the boundaries – there the communication of a single column (or row) with the nearest neighbour is needed. This can be achieved by having additional ghost-layers that contain the boundary data of the neighbouring tasks. As the

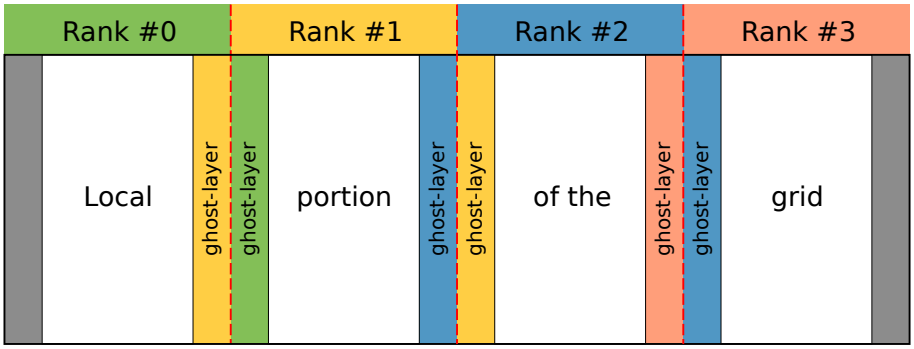

Figure 2: 2D domain decomposition

system is aperiodic, the outermost ranks communicate with only one neighbour, and the inner ranks with two neighbours.

Code

The solver carries out the time development of the 2D heat equation over the number of time steps provided by the user. The default geometry is a flat rectangle (with grid size provided by the user), but other shapes may be used via input files – a bottle is given as an example in common/bottle.dat.

Examples on how to run the binary: - `./heat` (no arguments - the program will run with the default arguments: 2000x2000 grid and 500 time steps) - `./heat bottle.dat` (one argument - start from a temperature grid provided in the file `bottle.dat` for the default number of time steps) - `./heat bottle.dat 1000` (two arguments - will run the program starting from a temperature grid provided in the file `bottle.dat` for 1000 time steps) - `./heat 4000 8000 1000` (three arguments - will run the program in a 4000x8000 grid for 1000 time steps)

The program will produce an image (PNG) of the temperature field after every 500 iterations. You can change the frequency by modifying the parameter `image_interval` in main.c (or main.F90). You can also visualize the images e.g. with the following command: `animate heat_*.png`.