INAF HPC course 2024

hands-on session

Jacobi solver

Overview

Partial differential equations play an important role in many branches of science and engineering. The Poisson problem is a partial differential equation (PDE) that is widely applied in many applications.

We use this problem as a means for describing the features of MPI, OpenMP, CUDA, OpenACC that can be used in solving PDEs.

The Poisson problem

The Poisson problem in two space dimensions x and y takes the form:

$$\nabla^2 u = \frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y} = f(x,y), \text{ in the interior}$$

$$u(x,y) = g(x,y), \text{ on the boundary}$$

An approximate solution can be found using a square mesh (also called a grid) consisting of points *(xi , yi),* given by

$$x_i = \frac{i}{n+1},\ i = 0,..,n+1 \qquad y_i = \frac{j}{n+1},\ j = 0,..,n+1$$

with n+2 points along each edge of the mesh.

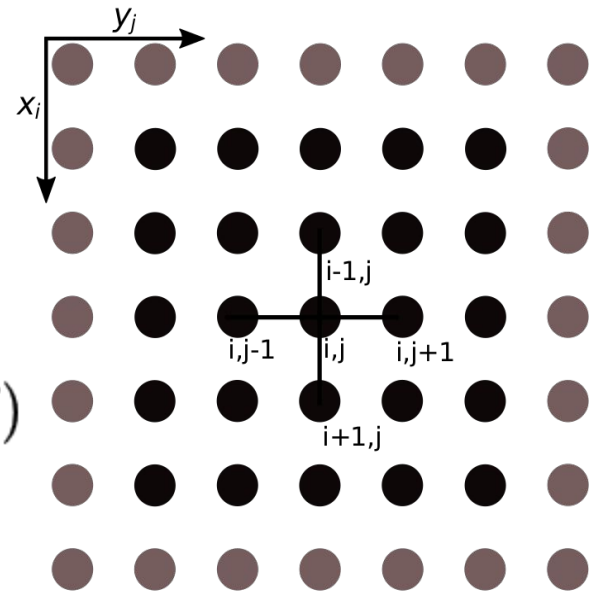
Jacobi's iterative method

The approximate solution *u* is found only at the points (*xi* , *yi*). Using the mesh we get the discretized equation,

$$u_{i-1,j} + u_{i,j+1} + u_{i,j-1} + u_{i+1,j} - 4u_{i,j} = h^2 f(i,j)$$

where h = 1 / (n+1).

Our aim is to solve the latest equation everywhere on the mesh.



Approximation for 2-D Poisson problem, with n = 5. The boundaries of the domain are shown in gray.

Jacobi's iterative method

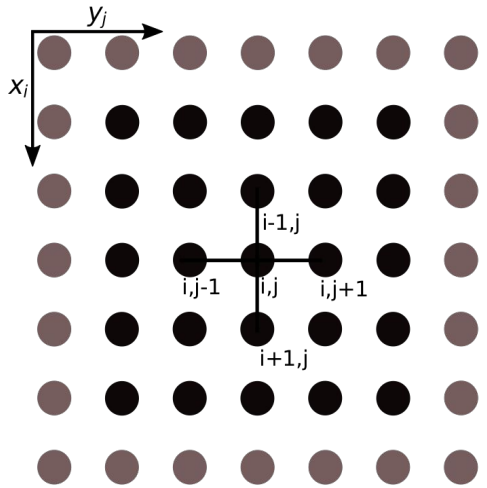
One approach is to rewrite the previous equation as,

$$u_{i,j} = 0.25(u_{i-1,j} + u_{i,j+1} + u_{i,j-1} + u_{i+1,j} - h^2 f_{i,j})$$

iterate by choosing values for all mesh points u(i,j) and then replacing them by using

$$u_{i,j}^{k+1} = 0.25(u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} + u_{i+1,j}^{k} - h^2 f_{i,j})$$

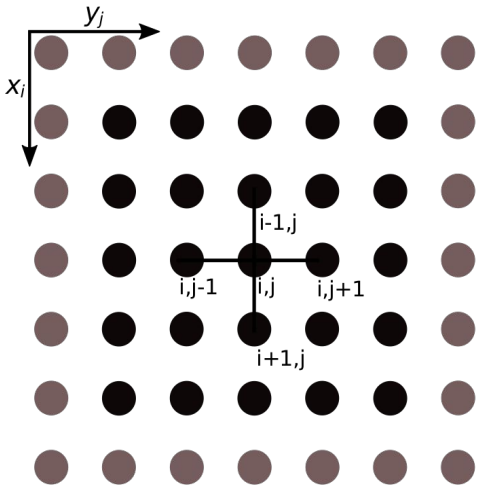
This convergent method is known as *Jacobi iteration.*



Approximation for 2-D Poisson problem, with n = 5. The boundaries of the domain are shown in gray.

Core algorithm of the Jacobi iteration

```
#define N  5

double u[N + 2][N + 2], unew[N + 2][N + 2];

for (int i=1 ; i<=N ; i++)

     for (int j=1 ; j<=N ; j++)

          unew[i,j] = 0.25 * (u[i-1][j] + u[i][j+1] +

                              u[i][j-1] + u[i+1][j] -

                              h * h * f[i][j]);
```



Approximation for 2-D Poisson problem, with n = 5. The boundaries of the domain are shown in gray.

Algorithm Implementation: serial code

Here's a sketch on how your code should be correctly written:

- set grid indices (input parameter of the program);
- allocate memory for 2D arrays;
- initialize solution array with boundaries conditions (u[i][j] = 0 in the interior points);
- start iterating (until convergence is reached);
  - assign boundary conditions;
  - update 2D solution;
  - compute residual;
- write solution to disk.

Problem details (serial code proposed as start point)

- Find the steady-state temperature distribution of the rectangular plate $0 \le x \le 1$, $0 \le y \le 1$, subject to the following Dirichlet boundary conditions (x, y arrays are the I.C.);
- compute the residuals through (at every iteration)

$$\epsilon = \sum_{ij} h^2 \|u_{ij}^{k+1} - u_{ij}^{k}\|$$

- quit iteration loop when $\epsilon < 10^{-5}$;
- using a grid 128 x 128 the convergence is achieved in ≅ 7316 iterations.

$$\begin{cases} u(i,0) & = 1 - y[i] \\ u(i,N+1) & = y[i] * y[i] \\ u(N+1,j) & = 1 - x[j] \\ u(0,j) & = x[j] \end{cases}$$